Hiroyuki Kitagawa
Yoshiharu Ishikawa
Qing Li
Chiemi Watanabe (Eds.)

LNCS 5982

Database Systems for Advanced Applications

15th International Conference, DASFAA 2010
Tsukuba, Japan, April 2010
Proceedings, Part II

2 Part II

Springer

Hiroyuki Kitagawa Yoshiharu Ishikawa
Qing Li Chiemi Watanabe (Eds.)

Database Systems for Advanced Applications

15th International Conference, DASFAA 2010
Tsukuba, Japan, April 1-4, 2010
Proceedings, Part II

Springer

Volume Editors

Hiroyuki Kitagawa
University of Tsukuba, Graduate School of Systems and Information Engineering
Tennohdai, Tsukuba, Ibaraki 305–8573, Japan
E-mail: kitagawa@cs.tsukuba.ac.jp

Yoshiharu Ishikawa
Nagoya University, Information Technology Center
Furo-cho, Chikusa-ku, Nagoya 464-8601, Japan
E-mail: ishikawa@itc.nagoya-u.ac.jp

Qing Li
City University of Hong Kong, Department of Computer Science
83 Tat Chee Avenue, Kowloon, Hong Kong, China
E-mail: itqli@cityu.edu.hk

Chiemi Watanabe
Ochanomizu University, Department of Information Science
2-1-1, Otsuka, Bunkyo-ku, Tokyo 112-8610, Japan
E-mail: chiemi@is.ocha.ac.jp

Library of Congress Control Number: 2010922033

CR Subject Classification (1998): H.2, H.3, H.4, H.5, C.2, J.1

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

ISSN	0302-9743
ISBN-10	3-642-12097-0 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-12097-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Message from the DASFAA 2010 Chairs

It is our great pleasure to welcome you to the proceedings of the 15th International Conference on Database Systems for Advanced Applications (DASFAA 2010). DASFAA is an international forum for academic exchange and technical discussions among researchers, developers, and users of databases from academia, business, and industry. DASFAA is a leading conference in the areas of databases, large-scale data management, data mining, and the Web. We are delighted to have held the 15th conference in Tsukuba during the cherry blossom season – the very best season of the year.

The call for papers attracted 237 research submissions from 25 countries / regions (based on the affiliation of the first author). Among them, 55 regular papers and 16 short papers were selected for presentation after a thorough review process by the Program Committee. The Industrial Committee, chaired by Hideko S. Kunii and Umesh Dayal, selected 6 industrial papers for presentation from 15 submissions and organized an industrial invited talk. The conference program also included 22 demo presentations selected from 33 submissions by the Demo Committee, chaired by Takahiro Hara and Kian-Lee Tan.

We are proud to have had two distinguished keynote speakers: Gerhard Weikum (Max-Planck Institute for Informatics) and Raghu Ramakrishnan (Yahoo! Research). Their lectures were the highlight of this conference. Tutorial Co-chairs, Kazutoshi Sumiya and Wookey Lee organized three tutorials by leading experts: Mining Moving Objects, Trajectory and Traffic Data (by Jiawei Han, Zhenhui Li, and Lu An Tang), Querying Large Graph Databases (by Yiping Ke, James Cheng, and Jeffrey Xu Yu), and Introduction to Social Computing (by Irwin King). A stimulating panel was organized by Panel Co-chairs Yasushi Kiyoki and Virach Sornlertlamvanich. This rich and attractive conference program boasts conference proceedings that span two volumes in Springer's *Lecture Notes in Computer Science* series.

Beyond the main conference, Masatoshi Yoshikawa and Xiaofeng Meng, who chaired the Workshop Committee, put together workshops that were of interest to all. The workshop papers are included in a separate volume of proceedings also published by Springer in its *Lecture Notes in Computer Science* series.

DASFAA 2010 was jointly organized by the University of Tsukuba and the Database Society of Japan (DBSJ). It received in-cooperation sponsorship from the KIISE Database Society of Korea, the China Computer Federation Database Technical Committee, ARC Research Network in Enterprise Information Infrastructure, Asian Institute of Technology (AIT), "New IT Infrastructure for the Information-explosion Era," MEXT Grant-in-Aid for Scientific Research on Priority Areas, Japan, Information Processing Society of Japan (IPSJ), the Institute of Electronics, Information, and Communication Engineers (IEICE), Japan PostgreSQL Users Group, MySQL Nippon Association, and the Japanese

Firebird Users Group. We are grateful to the sponsors who contributed generously to making DASFAA 2010 successful. They are Beacon Information Technology Inc., Mitsubishi Electric Corporation, National Institute for Materials Science (NIMS), KDDI R&D Laboratories Inc., National Institute of Advanced Industrial Science and Technology (AIST), Ricoh Co., Ltd., NTT DATA Corporation, Hitachi, Ltd., Ricoh IT Solutions Co., Ltd., SRA OSS, Inc., Japan, and Nippon Telegraph and Telephone Corporation. We also appreciate financial support from the Telecommunications Advancement Foundation and Kayamori Foundation of Informational Science Advancement.

The conference would not have been possible without the support of many colleagues. We would like to express our special thanks to Honorary Conference Chair Yoshifumi Masunaga for his valuable advice on all aspects of organizing the conference. We thank Organizing Committee Chair Masaru Kitsuregawa and Vice Chair Miyuki Nakano, DBSJ Liaison Haruo Yokota, Publicity Co-chairs Jun Miyazaki and Hyoil Han, Local Arrangements Committee Chair Toshiyuki Amagasa, Finance Chair Atsuyuki Morishima, Publication Chair Chiemi Watanabe, and Web Chair Hideyuki Kawashima. We are grateful for the strong support from the DASFAA 2010 Geographical Area Chairs: Bonghee Hong (Korea), Li-Zhu Zhou (China), Jeffrey Xu Yu (Hong Kong), Ming-Syan Chen (Taiwan), Stéphane Bressan (Singapore), Vilas Wuwongse (Thailand), Krithi Ramamritham (India), James Bailey (Australia), Chen Li (America), and Peer Kröger (Europe). Our thanks go to all the committee members and other individuals involved in putting it all together.

Finally, we thank the DASFAA Steering Committee, especially the immediate past Chair, Kyu-Young Whang, and current Chair, Katsumi Tanaka, for their leaderships and encouragement.

April 2010

Hiroyuki Kitagawa
Yoshiharu Ishikawa
Qing Li

Organization

Honorary Conference Chair

Yoshifumi Masunaga	Aoyama Gakuin University, Japan

General Conference Chair

Hiroyuki Kitagawa	University of Tsukuba, Japan

Organizing Committee Chair

Masaru Kitsuregawa	The University of Tokyo, Japan

Organizing Committee Vice Chair

Miyuki Nakano	The University of Tokyo, Japan

DBSJ Liaison

Haruo Yokota	Tokyo Institute of Technology, Japan

Program Committee Co-chairs

Yoshiharu Ishikawa	Nagoya University, Japan
Qing Li	City University of Hong Kong, China

Industrial Committee Co-chairs

Hideko S. Kunii	Ricoh IT Solutions Co., Ltd., Japan
Umesh Dayal	HP Labs, USA

Tutorial Co-chairs

Kazutoshi Sumiya	University of Hyogo, Japan
Wookey Lee	Inha University, Korea

Panel Co-chairs

Yasushi Kiyoki	Keio University, Japan
Virach Sornlertlamvanich	NECTEC, Thailand

Demo Committee Co-chairs

Takahiro Hara	Osaka University, Japan
Kian-Lee Tan	National University of Singapore, Singapore

Workshop Committee Co-chairs

Masatoshi Yoshikawa	Kyoto University, Japan
Xiaofeng Meng	Renmin University, China

Publicity Co-chairs

Jun Miyazaki	Nara Institute of Science and Technology, Japan
Hyoil Han	LeMoyne-Owen College, USA

Local Arrangements Committee Chair

Toshiyuki Amagasa	University of Tsukuba, Japan

Finance Chair

Atsuyuki Morishima	University of Tsukuba, Japan

Publication Chair

Chiemi Watanabe	Ochanomizu University, Japan

Web Chair

Hideyuki Kawashima	University of Tsukuba, Japan

Geographical Area Chairs

Korea: Bonghee Hong	Pusan National University, Korea
China: Li-Zhu Zhou	Tsinghua University, China
Hong Kong: Jeffrey Xu Yu	Chinese University of Hong Kong, China
Taiwan: Ming-Syan Chen	National Taiwan University, Taiwan
Singapore: Stéphane Bressan	National University of Singapore, Singapore
Thailand: Vilas Wuwongse	Asian Institute of Technology, Thailand
India: Krithi Ramamritham	Indian Institute of Technology at Bombay, India
Australia: James Bailey	The University of Melbourne, Australia

America: Chen Li	University of California, Irvine and BiMaple, USA
Europe: Peer Kröger	Ludwig-Maximilians-Universität München, Germany

Best Paper Committee Co-chairs

Katsumi Tanaka	Kyoto University, Japan
Kyu-Young Whang	Korea Advanced Institute of Science and Technology (KAIST), Korea
Jianzhong Li	Harbin Institute of Technology, China

DASFAA Awards Committee

Tok Wang Ling (Chair)	National University of Singapore, Singapore
Kyu-Young Whang	Korea Advanced Institute of Science and Technology (KAIST), Korea
Katsumi Tanaka	Kyoto University, Japan
Kirthi Ramamirtham	Indian Institute of Technology at Bombay, India
Jianzhong Li	Harbin Institute of Technology, China
Dik Lun Lee	Hong Kong University of Science & Technology, China
Arbee L.P. Chen	National Chengchi University, Taiwan

Steering Committee

Katsumi Tanaka (Chair)	Kyoto University, Japan
Ramamohanarao Kotagiri (Vice Chair)	University of Melbourne, Australia
Kyu-Young Whang (Advisor)	Korea Advanced Institute of Science and Technology (KAIST), Korea
Yoshihiko Imai (Treasurer)	Matsushita Electric Industrial Co., Ltd., Japan
Kian-Lee Tan (Secretary)	National University of Singapore, Singapore
Yoon Joon Lee	Korea Advanced Institute of Science and Technology (KAIST), Korea
Qing Li	City University of Hong Kong, China
Krithi Ramamritham	Indian Institute of Technology at Bombay, India
Ming-Syan Chen	National Taiwan University, Taiwan
Eui Kyeong Hong	Univerity of Seoul, Korea
Hiroyuki Kitagawa	University of Tsukuba, Japan
Li-Zhu Zhou	Tsinghua University, China
Jianzhong Li	Harbin Institute of Technology, China
BongHee Hong	Pusan National University, Korea

Organizing Committee

Shuji Harashima	Toshiba Corporation, Japan
Atsushi Iizawa	Ricoh IT Solutions Co., Ltd., Japan
Minoru Inui	Beacon IT Inc., Japan
Tatsuo Ishii	SRA OSS, Japan
Hiroshi Ishikawa	Shizuoka University, Japan
Kyoji Kawagoe	Ritsumeikan University, Japan
Yutaka Kidawara	National Institute of Information and Communications Technology (NICT), Japan
Hajime Kitakami	Hiroshima City University, Japan
Isao Kojima	National Institute of Advanced Industrial Science and Technology (AIST), Japan
Kazunori Matsumoto	KDDI Lab., Japan
Masataka Matsuura	Fujitsu Ltd., Japan
Hirofumi Matsuzawa	IBM Japan, Japan
Shojiro Nishio	Osaka University, Japan
Makoto Okamato	Academic Resource Guide, Japan
Tetsuji Satoh	University of Tsukuba, Japan
Jun Sekine	NTT DATA Corporation, Japan
Shigenobu Takayama	Mitsubishi Electric Cooporation, Japan
Takaaki Tasaka	SGI Japan, Ltd., Japan
Yoshito Tobe	Tokyo Denki University, Japan
Masashi Tsuchida	Hitachi, Ltd., Japan
Masashi Yamamuro	NTT Corporation, Japan
Kazumasa Yokota	Okayama Prefectural University, Japan

Program Committee

Toshiyuki Amagasa	University of Tsukuba, Japan
Masayoshi Aritsugi	Kumamoto University, Japan
James Bailey	University of Melbourne, Australia
Ladjel Bellatreche	Poitiers University, France
Boualem Benatallah	University of New South Wales, Australia
Sourav Bhowmick	Nanyang Technological University, Singapore
Athman Bouguettaya	CSIRO, Australia
Chee Yong Chan	National University of Singapore, Singapore
Lei Chen	Hong Kong University of Science and Technology, China
Ming-Syan Chen	National Taiwan University, Taiwan
Reynold Cheng	University of Hong Kong, China
Gao Cong	Aalborg University, Denmark
Bin Cui	Peking University, China
Alfredo Cuzzocrea	ICAR-CNR / University of Calabria, Italy
Zhiming Ding	Chinese Academy of Sciences, China
Gill Dobbie	University of Auckland, New Zealand

Guozhu Dong	Wright State University, USA
Jianhua Feng	Tsinghua University, China
Ling Feng	Tsinghua University, China
Sumit Ganguly	Indian Institute of Technology Kanpur, India
Yunjun Gao	Zhejiang University, China
Lukasz Golab	AT&T Labs, USA
Vivekanand Gopalkrishnan	Nanyang Technological University, Singapore
Stéphane Grumbach	INRIA, France
Wook-Shin Han	Kyungpook National University, Korea
Takahiro Hara	Osaka University, Japan
Kenji Hatano	Doshisha University, Japan
Wynne Hsu	National University of Singapore, Singapore
Haibo Hu	Hong Kong Baptist University, China
Seung-won Hwang	POSTECH, Korea
Mizuho Iwaihara	Waseda University, Japan
Ramesh C. Joshi	Indian Institute of Technology Roorkee, India
Jaewoo Kang	Korea University, Korea
Norio Katayama	National Institute of Informatics, Japan
Yutaka Kidawara	National Institute of Information and Communications Technology, Japan
Myoung Ho Kim	Korea Advanced Institute of Science and Technology (KAIST), Korea
Markus Kirchberg	Institute for Infocomm Research, A*STAR, Singapore
Hajime Kitakami	Hiroshima City University, Japan
Jae-Gil Lee	IBM Almaden Research Center, USA
Mong Li Lee	National University of Singapore, Singapore
Sang-goo Lee	Seoul National University, Korea
Sang-Won Lee	Sungkyunkwan University, Korea
Wang-Chien Lee	Pennsylvania State University, USA
Cuiping Li	Renmin University, China
Jianzhong Li	Harbin Institute of Technology, China
Ee-Peng Lim	Singapore Management University, Singapore
Xuemin Lin	University of New South Wales, Australia
Chengfei Liu	Swinburne University of Technology, Australia
Jiaheng Lu	Renmin University, China
Sanjay Madria	University of Missouri-Rolla, USA
Nikos Mamoulis	University of Hong Kong, China
Weiyi Meng	Binghamton University, USA
Xiaofeng Meng	Renmin University, China
Jun Miyazaki	Nara Institute of Science and Technology, Japan
Yang-Sae Moon	Kangwon National University, Korea
Yasuhiko Morimoto	Hiroshima University, Japan
Miyuki Nakano	University of Tokyo, Japan
Wolfgang Nejdl	L3S / University of Hannover, Germany

Masato Oguchi	Ochanomizu University, Japan
Tadashi Ohmori	University of Electro-Communications, Japan
Makoto Onizuka	NTT Cyber Space Laboratories, NTT Corporation, Japan
Satoshi Oyama	Hokkaido University, Japan
HweeHwa Pang	Singapore Management University, Singapore
Jian Pei	Simon Fraser University, Canada
Wen-Chih Peng	National Chiao Tung University, Taiwan
Evaggelia Pitoura	University of Ioannina, Greece
Sunil Prabhakar	Purdue University, USA
Tieyun Qian	Wuhan University, China
Krithi Ramamritham	Indian Institute of Technology Bombay, India
Uwe Röhm	University of Sydney, Australia
Shourya Roy	Xerox India Innovation Hub, India
Yasushi Sakurai	NTT Communication Science Laboratories, NTT Corporation, Japan
Simonas Saltenis	Aalborg University, Denmark
Monica Scannapieco	University of Rome, Italy
Markus Schneider	University of Florida, USA
Heng Tao Shen	University of Queensland, Australia
Hyoseop Shin	Konkuk University, Korea
Atsuhiro Takasu	National Institute of Informatics, Japan
Kian-Lee Tan	National University of Singapore, Singapore
David Taniar	Monash University, Australia
Egemen Tanin	University of Melbourne, Australia
Jie Tang	Tsinghua University, China
Yufei Tao	Chinese University of Hong Kong, China
Vincent S. Tseng	National Cheng Kung University, Taiwan
Anthony K.H. Tung	National University of Singapore, Singapore
Vasilis Vassalos	Athens University of Economics and Business, Greece
Guoren Wang	Northeastern University, China
Jianyong Wang	Tsinghua University, China
Jiying Wang	City University of Hong Kong, China
Wei Wang	University of New South Wales, Australia
Raymond Chi-Wing Wong	Hong Kong University of Science and Technology, China
Vilas Wuwongse	Asian Institute of Technology, Thailand
Jianliang Xu	Hong Kong Baptist University, China
Haruo Yokota	Tokyo Institute of Technology, Japan
Ge Yu	Northeastern University, China
Jeffrey Xu Yu	Chinese University of Hong Kong, China
Rui Zhang	University of Melbourne, Australia
Aidong Zhang	University at Buffalo, SUNY, USA
Yanchun Zhang	Victoria University, Australia
Aoying Zhou	East China Normal University, China

Industrial Committee

Rafi Ahmed	Oracle, USA
Edward Chang	Google, China and University of California Santa Barbara, USA
Dimitrios Georgakopoulos	CSIRO, Australia
Naoko Kosugi	NTT Corporation, Japan
Kunio Matsui	Nifty Corporation, Japan
Mukesh Mohania	IBM Research, India
Yasushi Ogawa	Ricoh Co. Ltd., Japan
Makoto Okamoto	Academic Resource Guide, Japan
Takahiko Shintani	Hitachi, Ltd., Japan

Demo Committee

Lin Dan	Missouri University of Science and Technology, USA
Feifei Li	Florida State University, USA
Sanjay Kumar Madria	Missouri University of Science and Technology, USA
Pedro Jose Marron	University of Bonn, Germany
Sebastian Michel	Max-Planck-Institut für Informatik, Germany
Makoto Onizuka	NTT CyberSpace Laboratories, NTT Corporation, Japan
Chedy Raissi	National University of Singapore, Singapore
Lakshmish Ramaswamy	The University of Georgia, Athens, USA
Lidan Shou	Zhejiang University, China
Lei Shu	Osaka University, Japan
Tomoki Yoshihisa	Osaka University, Japan
Koji Zettsu	National Institute of Information and Communications Technology, Japan
Xuan Zhou	CSIRO, Australia

Workshop Committee

Qiang Ma	Kyoto University, Japan
Lifeng Sun	Tsinghua University, China
Takayuki Yumoto	University of Hyogo, Japan

Local Arrangements Committee

Kazutaka Furuse	University of Tsukuba, Japan
Takako Hashimoto	Chiba University of Commerce, Japan
Yoshihide Hosokawa	Gunma University, Japan
Sayaka Imai	Sagami Women's University, Japan

Kaoru Katayama	Tokyo Metropolitan University, Japan
Shingo Otsuka	National Institute for Materials Science, Japan
Akira Sato	University of Tsukuba, Japan
Tsuyoshi Takayama	Iwate Prefectural University, Japan
Hiroyuki Toda	NTT Corporation, Japan
Chen Han Xiong	University of Tsukuba, Japan

External Reviewers

Sukhyun Ahn
Muhammed Eunus Ali
Mohammad Allaho
Parvin Asadzadeh
Seyed Mehdi Reza Beheshti
Stéphane Bressan
Xin Cao
Ding Chen
Keke Chen
Shiping Chen
Yi-Ling Chen
Hong Cheng
Van Munin Chhieng
Taewon Cho
Jaehoon Choi
Tangjian Deng
Pham Min Duc
Takeharu Eda
Yuan Fang
Chuancong Gao
Shen Ge
Nikos Giatrakos
Kazuo Goda
Jian Gong
Yu Gu
Adnene Guabtni
Rajeev Gupta
Tanzima Hashem
Jenhao Hsiao
Meiqun Hu
Guangyan Huang
Oshin Hung
Rohit Jain
Bin Jiang
Lili Jiang
Qingsong Jin
Kaoru Katayama
Yoshihiko Kato
Hideyuki Kawashima
Hea-Suk Kim
Kyoung-Sook Kim
Georgia Koloniari
Susumu Kuroki
Injoon Lee
Jinseung Lee
Jongwuk Lee
Ki Yong Lee
Ki-Hoon Lee
Sanghoon Lee
Sunwon Lee
Yutaka I. Leon-Suematsu
Kenneth Leung
Guoliang Li
Jianxin Li
Jing Li
Lin Li
Xian Li
Yu Li
Bingrong Lin
Xin Lin
Yimin Lin
Xingjie Liu
Yifei Liu
Haibing Lu
Min Luo
Jiangang Ma
Chris Mayfield
Debapriyay Mukhopadhyay
Tiezheng Nie
Sarana Nutanong
Ardian Kristanto Poernomo
Yinian Qi
Meena Rajani
Harshana Randeni
Gook-Pil Roh
Jong-Won Roh
Seung Ryu
Sherif Sakr
Jie Shao
Zhitao Shen
Wanita Sherchan
Reza Sherkat
Liangcai Shu
Chihwan Song
Kazunari Sugiyama
Keiichi Tamura
Takayuki Tamura
Masashi Toyoda
Mayumi Ueda
Muhammad Umer
Daling Wang
Yousuke Watanabe
Ling-Yin Wei
Jemma Wu
Hairuo Xie
Kefeng Xuan
Masashi Yamamuro
Zenglu Yang
Jie (Jessie) Yin
Peifeng Yin
Tomoki Yoshihisa
Naoki Yoshinaga
Weiren Yu
Kun Yue
Dana Zhang

Rong Zhang
Shiming Zhang
Wenjie Zhang
Geng Zhao
Xiaohui Zhao
Bin Zhou
Rui Zhou
Xiangmin Zhou
Xuan Zhou
Gaoping Zhu
Ke Zhu
Qijun Zhu

Organizers

University of Tsukuba   The Database Society of Japan (DBSJ)

In Cooperation with

KIISE Database Society of Korea
The China Computer Federation Database Technical Committee
ARC Research Network in Enterprise Information Infrastructure
Asian Institute of Technology (AIT)
"New IT Infrastructure for the Information-explosion Era", MEXT (Ministry of Education, Culture, Sports, Science and Technology) Grant-in-Aid for Scientific Research on Priority Areas, Japan
Information Processing Society of Japan (IPSJ)
The Institute of Electronics, Information, and Communication Engineers (IEICE)
Japan PostgreSQL Users Group
MySQL Nippon Association
The Japanese Firebird Users Group

Sponsoring Institutions

Platinum Sponsors

BeaconIT, Japan

MITSUBISHI ELECTRIC CORPORATION, Japan

Gold Sponsors

National Institute for Materials Science (NIMS), Japan

KDDI R&D Laboratories Inc., Japan

National Institute of Advanced Industrial Science and Technology (AIST), Japan

Silver Sponsors

Ricoh Co., Ltd., Japan

NTT DATA CORPORATION, Japan

Hitachi, Ltd., Japan

Bronze Sponsors

Ricoh IT Solutions Co., Ltd., Japan

SRA OSS, Inc., Japan

Table of Contents – Part II

B^s-tree: A Self-tuning Index of Moving Objects

Nan Chen, Lidan Shou, Gang Chen, Ke Chen, and Yunjun Gao

College of Computer Science, Zhejiang University, China
{cnasd715,should,cg,chenk,gaoyj}@cs.zju.edu.cn

Abstract. Self-tuning database is a general paradigm for the future development of database systems. However, in moving object database, a vibrant and dynamic research area of the database community, the need for self-tuning has so far been overlooked. None of the existing spatio-temporal indexes can maintain high performance if the proportion of query and update operations varies significantly in the applications. We study the self-tuning indexing techniques which balance the query and update performances for optimal overall performance in moving object databases. In this paper, we propose a self-tuning framework which relies on a novel moving object index named B^s-tree. This framework is able to optimize its own overall performance by adapting to the workload online without interrupting the indexing service. We present various algorithms for the B^s-tree and the tuning techniques. Our extensive experiments show that the framework is effective, and the B^s-tree outperforms the existing indexes under different circumstances.

Keywords: spatio-temporal database, moving object, index, self-tuning.

1 Introduction

With the rapid progress in hardware and software technologies, the database systems and applications are becoming increasingly more complex and dynamic, however at lower costs than ever. As a result, the traditional solution of employing database administrators to maintain and fine-tune the DBMS for higher performance has appeared to be inefficient and uneconomical. The self-tuning functionality of databases is expected to be the substitute for DBAs. Intuitively, a self-tuning database provides practical solutions to adjust itself automatically for optimal performance with minimal human intervention. In the past years, a lot works (e.g., [3]) have been done to extend the self-tuning capabilities of database systems. However, the existing works so far are mainly restricted to traditional static databases.

1.1 Motivation

The recent emergence of numerous moving object database applications, such as traffic control, meteorology monitoring, mobile location computing and so on, has called on for self-tuning techniques in the moving object databases as well. In the numerous techniques proposed for managing moving object databases, index design appears to be the spotlight. Therefore, we shall look at the self-tuning techniques for moving object indexes. The problem that we consider for tuning is the update and query performance.

We observe that the update and query performances of moving object indexes display interesting patterns if we classify the existing indexes into two major categories regarding the data representations that they use [5]. The first class includes R/R*-tree-based indexes, such as the TPR-tree [9] and the TPR*-tree [11]. This class is characterized by good query performance but expensive update costs. The second class includes techniques that employ space partitioning and data/query transformations to index object positions, such as the B^x-tree [8]. As it is based on the B+-tree, the B^x-tree has good update performance. However, it does not achieve satisfactory query efficiency, as shown in [13]. In addition, the second class has better concurrency performance than the first, because, as shown in [8,6], the concurrency control in a R/R*-tree-based index is more complex and time-consuming than that in a B+-tree-based index.

The above observation leads to the implication that the current moving-object indexing techniques encounter difficulty when handling variable needs in the proportion of query and update operations. Given some examples of these applications, an aviation monitoring system may need more query operations, and oppositely make fewer updates to the database. While an animal position tracking system may require far more updates than queries. In addition, there are also applications which require variable ratio of updates and queries at different time. For example, in a traffic monitoring system of a city, the proportions of updates and queries may vary widely by time. It may receive increased updates during the rush hour, and process more queries at the report time.

To solve the problem with these dynamic applications, we need a self-tunable index structure which is able to strike a balance between the performance of queries and updates, thereby achieving good overall performance for different proportion of updates and queries. In addition, the self-tuning of the index should not interrupt the index service. Based on the discussion in the above, none of the existing moving object indexes can satisfy such requirements readily. They suffer from either poor query performance or large update costs, and does not have the ability of self-tuning. In fact, the performance of queries and that of updates usually affect each other, and are often anti-correlated in most of the existing indexes. Therefore, they are not appropriate for these environments. We believe this problem has so far been overlooked.

1.2 Overview of the Proposed Techniques

In this paper, we propose an online self-tuning framework for indexing moving-objects based on a novel data structure called B^s-tree. The B^s-tree provides the functionality of self-tuning, and can adaptively handle both queries and updates efficiently. Our framework uses an *update parameter* α and a *query parameter* β to tune the performance of the B^s-tree. Figure 1 shows the components of the proposed self-tuning framework. When an update arrives, the Key Generator Module (KGM) computes the index keys according to the update parameter α of the index, inserts them into the B^s-tree and reports the update cost to the Online Tuning Module (OTM). On the other hand, when a query arrives, the Query Executor Module (QEM) processes the query according to the query parameter β of the index and reports the query cost to OTM. OTM monitors the performance and controls the tuning.

The proposed B^s-tree has many desirable features. It can provide self-tuning for optimal overall performance without interrupting the indexing service. When the query

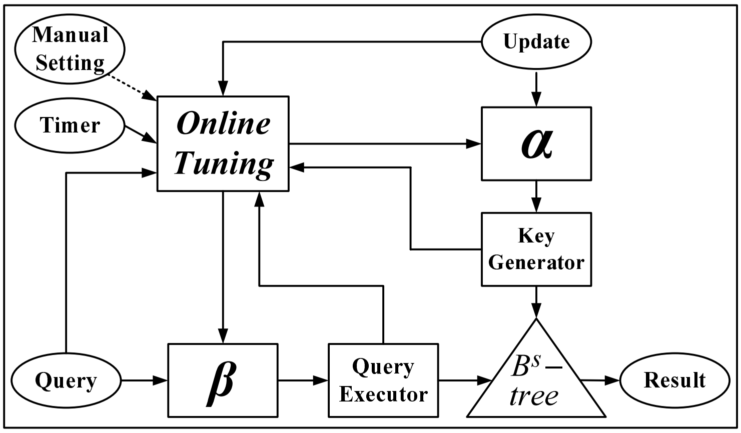


Fig. 1. Online Self-tuning Framework

cost dominates the overall performance, the B^s-tree achieves considerably higher performance of queries, at the expense of slightly more update costs. In contrast, if the update cost dominates the overall performance, the B^s-tree will sacrifice some query performance to obtain lower update costs. Our extensive experiments show that the B^s-tree outperforms the existing indexes in different environments. Another advantage of the B^s-tree is that it is based on the classical B+-tree. Therefore it can be easily implemented in the existing DBMSs and is expected to have good concurrency performance. Table 1 summarizes the properties of the B^s-tree compared to the previous indexes.

Table 1. Comparison of predictive indexes of moving objects.

	Query Cost	Update Cost	DBMS integration	Self-Tuning	Concurrency
B^x	HIGH	LOW	EASY	NO	HIGH
TPR*	LOW	HIGH	HARD	NO	LOW
B^s	LOW	LOW	EASY	YES	HIGH

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the structure of the B^s-tree, and presents the associated query and update operations. Section 4 analyzes the I/O cost of the B^s-tree, and introduces how the online self-tuning framework works. Section 5 presents the results of the experiments. Section 6 conludes the paper and discusses possible future work.

2 Related Work

Lots of index structures have been proposed to index the current and predicted future positions of moving objects. According to [5], these indexes can be classified into two major categories depending on the trees they are based on. The first category is the R/R*-tree-based indexes which base on the R-tree [7] and the R*-tree [1], such as the Time-Parameterized R-tree (TPR-tree) [9] and its variant, the TPR*-tree [11]. A node in the TPR-tree is represented as a MOving Rectangle (MOR), including a Minimum

Bounding Rectangle (MBR) and a velocity Bounding Rectangle (VBR). The extent of a MBR grows according to its VBR and it never shrinks, although it is generally larger than strictly needed. This guarantees that every MBR always bounds the objects that belong to it at all times considered. The update algorithm of the TPR-tree is exactly the same as those of the R*-tree, by simply replacing the four penalty metrics of the R*-tree with their integral counterparts. Based on the same structure, the TPR*-tree provides a new set of insertion and deletion algorithms aiming at minimizing a certain cost function. In the TPR-tree and the TPR*-tree, predictive query for a future time t is processed by visiting MBRs expanded to t. The TPR-tree and the TPR*-tree are deployed to solve a large number of spatio-temporal query problems (e.g., [2] for KNN queries). However, they have a major weakness: the update costs are expensive.

The second category of indexes relies on space partitioning and data/query transformations, such as the B+-tree based indexes [8] [5] [13]. The B^x-tree [8] is the first effort to adapt the B+-tree to index moving objects. It partitions the time axis into equal intervals, called phases, and partitions the data space into uniform cells. An index partition of the B+-tree is reserved for each phase. When an insertion operation of a moving object arrives, the B^x-tree computes the position of this object at the end timestamp of the next phase, and transforms this position to a value of the space-filling curve, such as the Hilbert-curve and Z-curve. This value is then indexed in the corresponding partition of the B+-tree. To process a range query, the B^x-tree checks each existing partition for qualifying objects. Specifically, in one partition, the B^x-tree enlarges the query window to the end timestamp of the corresponding phase. It first enlarges the query window according to the minimum and maximum velocities of all moving objects, and then makes the enlargement exacter using the space velocity histogram which records the minimum and maximum velocities of the space cells. Then the B^x-tree scans all multiple ranges of index keys which fall within the enlarged query window, and checks the corresponding moving objects to finds the ones satisfying the query. Since each partition of the B+-tree has to be checked, the query processing of the B^x-tree is not efficient, as shown by both the experiment results of [13] and those of ours.

The B^{dual}-tree in [13] also uses a B+-tree to index moving objects. However, it indexes objects in a dual space instead, considering both location and velocity information to generate index keys. The B^{dual}-tree maintains a set of MORs for each internal entry, and uses R-tree-like query algorithms as in the TPR/TPR*-tree. A range query searches the subtree of an internal entry only if any of its MORs intersects with the query region. By partitioning the velocity space, the B^{dual}-tree improves the query performance of the B^x-tree. However, maintaining the MORs introduces high computation workload, which slows down the fast update and high concurrency of the B+-Tree. It has worse update performance than the B^x-tree, and worse query performance than the TPR*-tree. In addition, by modifying the update and query algorithms of the B+-tree, the B^{dual}-tree can no longer be readily integrated into existing DBMSs.

Besides the query and update performance, there are some other issues with the moving object indexes. [4] proposes a series of B+-tree-based indexes structures to handle the problem of *doubtful positive*. When the update durations of moving objects are highly variable, the result of a predictive query may contain frequently updated objects whose states would be updated much earlier than the future query timestamp. These result objects are apparently doubtful and are called doubtful positives. The work in [4] introduces the concept of *prediction life period* for every moving object to handle predictive queries with no doubtful positive. In addition, different indexes in [4] have different balance of update performance and query performance. However, these indexes have no ability of self-tuning. The DBA has to decide to choose which index to satisfy the application. And, when the requirement of the application changes, the DBA has to stop the index service, delete the current index, choose and rebuild another index. The ST^2B-tree [5] is a self-tuning index of moving objects, based on the B+-tree. However, the problem it addresses is the varying distribution of moving objects over time and space, while our work focuses on the change of the proportion of query and update operations.

3 The B^s-tree

3.1 Index Structure

The B^s-tree is built on the B+-tree without any change to the underlying B+-tree structure and insertion/deletion algorithms, thus is readily implementable in existing DBMS. In order to support B-link concurrency control, each internal node contains a pointer to its right sibling. A moving object is transformed to one or more 1D keys and indexed by the B^s-tree. We now introduce how the transformation works.

As [4], in order to handle the problem of doubtful positive, we also define a *prediction life period*, t_{per} for each moving object, as the interval from its last update timestamp to the next update timestamp. It is the future within which the current information of this moving object is predicted to be right, and it can be updated. Similar to all the above existing works of moving object indexes, moving objects in the B^s-tree are modeled as linear functions of time. Therefore, the position of a moving object at time t is given by $O = (\boldsymbol{x}, \boldsymbol{v}, t_{ref}, t_{per}) = \boldsymbol{x}(t) = (x_1(t), x_2(t), ..., x_d(t))$, where $t_{ref} < t < t_{ref} + t_{per}$. Here, $\boldsymbol{x}$ and $\boldsymbol{v}$ is the location and velocity of the moving object at the reference time t_{ref}. Thus, $\boldsymbol{x}(t) = \boldsymbol{x} + \boldsymbol{v} * (t - t_{ref})$. For each query, the *query interval* indicates how far the query "looks" into the future. If the query interval of a query exceeds t_{per} of a moving object, this object is considered as a doubtful positive and will not appear in the result.

We partition the time axis into intervals of duration T_m, and then sub-partition each T_m into n equal-length sub-intervals of duration, each at length of T_{sam}, called *phases*. We label each phase with a timestamp t_{lab} which is the end time point of this phase. Unlike the B^x-tree which samples a moving object exact once, the B^s-tree samples a moving object one or more times. We introduce an *update parameter* α for the B^s-tree, where $1 \leq \alpha \leq n + 1$. When an insertion of a moving object arrives, the B^s-tree computes the positions at t_{lab} of every α phases located between $t_{lab} = [t_{ref}]_l$ and $t_{lab} = [t_{ref} + t_{per}]_l$, where $[t]_l$ means the nearest future t_{lab} of t. For each sampling, the B^s-tree maps the position to a signal-dimensional value, called KEY_{sam}, using the Hilbert curve. This value is concatenated with the phase number of its t_{lab}, called KEY_{lab}, to generate the index key values B_{value}, which is then inserted to a B+-tree. The relationship and computation methods are as follow:

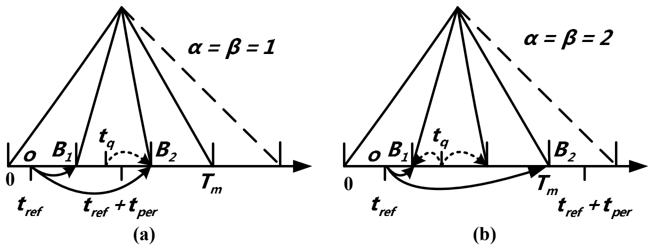


Fig. 2. An example of Insertion and Query in the B^s-tree

$$B_{value} = KEY_{lab} \oplus KEY_{sam},$$
$$KEY_{lab} = (t_{lab}/(T_m/n)) \bmod (n+1),$$
$$KEY_{sam} = Curve(\boldsymbol{x}(t_{lab})).$$

In figure 2, the solid lines with arrowheads show an insertion case of the B^s-tree when $\alpha = 1$ and $\alpha = 2$. We can see how a moving object is sampled, according to its prediction life period t_{per} and the update parameter α. The B^s-tree contains data belonging to $n+1$ phases, and thus it includes $n+1$ sub-trees, or partitions. As time passes, repeatedly the oldest partition expires, and a new one is appended. Therefore, the index rolls over. An update of a moving object deletes its old sampled values and inserts new ones. Except maintaining the space velocity histogram H_s as the B^x-tree, the B^s-tree also maintain a partition velocity histogram H_p, which stores the maximum and minimum velocities of the moving objects that inserted into each partition.

3.2 Query Algorithms

A range query $Q_r(R, t_q)$ retrieves all moving objects which intersect the query region R at the query time t_q, which does not precede the current time. We introduce a *query*

```
        **Algorithm Range Query** $Q_r(R, t_q)$
Input: $R$ is the query window and $t_q$ is the query time.
Output: All objects in $R$ at $t_q$ within their $t_{per}$s
1. $t_{lab} \leftarrow [t_q]_l$
2. For $k \leftarrow 1$ to $\beta$
3.      $R' = ExpandWindow(R, t_q, t_{lab}, H_p, H_s)$
4.      For each key range in $R'$
5.          For each moving object in the key range
6.              Compute its position at $t_q$
7.              If it is in $R$ at $t_q$ within its $t_{per}$
8.                  Add it to the result set
9.          Find the next key range
10.         If not found
11.             Break
12.     $t_{lab} \leftarrow [t_q]_l - T_{sam}/n$
13.Return result set
```

Fig. 3. Range Query Algorithm

parameter β for the B^s-tree. Unlike the B^x-tree which expands query windows to each partition, the B^s-tree only expands query windows β times to the phase which t_q belongs to and the $\beta - 1$ ones before. In figure 2, the dashed lines with arrowheads show the phases which a query window is expanded to when $\beta = 1$ and $\beta = 2$. We will discuss the relationship between α and β in section 4.2.

Figure 3 shows the range query algorithm of the B^s-tree. The algorithm searches the partition which the query time belongs to and the $\beta - 1$ ones before (Line 2). For each partition, it expands the query window from the query time t_q to the end timestamp t_{lab} of the corresponding phase (Line 3). Here, H_p indicts the partition velocity histogram and H_s indicts the space velocity histogram. The enlargement first expand the window according to the maximum and minimum velocities of the corresponding phases in H_p to get a preliminary window R_{pre}, and then finds the minimum and maximum velocities in the cells that R_{pre} intersects, using H_s, to get the exacter expanded window R'. After the enlargement, the algorithm can get several *key ranges* in R', which are one-dimensional intervals of index values falling within the expanded query window R'. For each key range, a range query of the B+-tree is executed (Line 4). For each object found in the key range, the algorithm checks whether it is in R at t_q within its t_{per}. If so, the object satisfies the query and is added to the result set (Line 5-8).

A KNN query $Q_{KNN}(p, t_q, k)$ retrieves k moving objects for which no other moving objects are nearer to o at t_q. As in the B^x-tree, a KNN query of the B^s-tree is handled as incremental range queries $Q_r(R, t_q)$ until exact k nearest neighbors are found. The initial search range is a region centered at p_t with extension $r = D_k/k$, where p_t is the position of p at t_q. If the KNNs are not found in this region, it extends the search radius by r as in [12]. D_k is estimated by the equation as in [8] [5]:

$$D_k = \frac{2}{\sqrt{\pi}}[1 - \sqrt{1 - \sqrt{k/N}}].$$

Note that, during the incremental range queries, the query window of a range query will contain the former one. Therefore, for the KNN queries of the B^s-tree, we record the expanded window of a range query for each partition. When processing the next range query, the area covered by the former expanded window does not need to be checked.

4 Self-tuning of the B^s-tree

In this section, we first analyze the update and query I/O cost of the B^s-tree. And then basing on the analysis, we introduce the online self-tuning framework of the B^s-tree.

4.1 Update and Query Performance Analysis

First, let us focus on the update performance of the B^s-tree. For a B+-based-tree, the height of the tree is usually significantly important for the update performance. Assuming that the B^s-tree maintains N values, has the fan-out of F, and its nodes are filled with at least $F/2$ values. So the height is $H = \log_{F/2} N + 1$ at the most. Since sampling a moving object for one or more times, the B^s-tree may ask for a bigger N than the B^x-tree. However, with a big F which is usually in the hundreds, such a bigger N will seldom affect H. As most indexes of moving objects, update of a moving object in the B^s-tree is a deletion operation followed by an insertion operarion. Therefore, the total update I/O cost of the B^s-tree is the sum of the deletion and insertion cost:

$$IO_{upd} = IO_{del} + IO_{ins} = \left(\left\lceil \frac{\left[t_{ref} + t_{per}\right]_l - \left[t_{ref}\right]_l}{T_{sam} * \alpha} \right\rceil + \left\lceil \frac{\left[t'_{ref} + t'_{per}\right]_l - \left[t'_{ref}\right]_l}{T_{sam} * \alpha'} \right\rceil \right) * H.$$

Here, $\lceil a \rceil$ means the smallest integer which is no smaller than a. Note that, for a moving object, among different updates, the prediction life period t_{per} and the update parameter α can change. We can see that the I/O cost of an update is only several times the tree height. Since H and T_{sam} are fixed, and t_{ref} and t_{per} are depended on the moving object itself, the update parameter α is the most important issue of the update cost. The update performance increases with α. We can tunes the update performance by tuning α.

Second, let us turn to the query performance of the B^s-tree. The B^s-tree handles range queries using the method of query window enlargement. It expands the query window for β times and traverses β partitions of the tree. Therefore, the total query I/O cost is the sum of the cost of the β partition traverses:

$$IO_{upd} = \sum_{i=1}^{\beta} \left(\sum_{j=1}^{m} \left(H - 1 + R_j \right) \right).$$

Here, for each partition traverse, m is the number of key ranges falling in the expanded window. For each key range, there is a unique search followed by a range search in the B+-tree. Specially, the I/O cost of a unique search is $H - 1$ internal nodes of the index, while the I/O cost of the range search is R_j leaf nodes of the index. Figure 4 gives an range query example of the figure 2 (b) where $\alpha = \beta = 2$. O is the position of the moving object o at the reference time. B_1 is its first key value sampled in the B^s-tree, while B_2 is not visible in figure 4. The shadowed rectangle is the original query window. It is expanded twice, one to the phase t_q belongs to and one for the phase before. In the first enlargement, there are two key ranges without containing any sampling value of o. While, in the second enlargement, after two unique-range scans of the B+-tree, it finds B_1. There are totally two enlargement and four unique-range scans for this range query.

The total cost of a range query is decided by the number of partition traverses: β. While, for each partition traverse, the cost is decided by the number and length of the

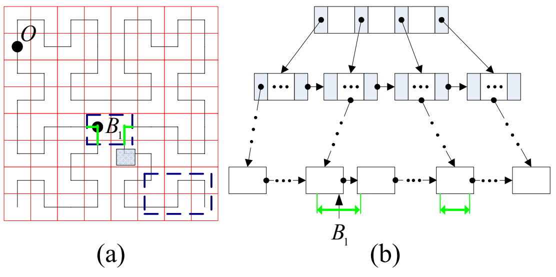


Fig. 4. An example of Query Analysis

key ranges. The larger the query window is expanded, the more and longer the key ranges it contains. The enlargement velocities and the enlargement time interval are the two issues of the query window enlargement. For the first issue, except the space velocity histogram used by the B^x-tree, the B^s-tree also uses the partition velocity histogram to make the enlargement velocities exacter. As for the second issue, the *enlargement time interval* set S_{eti} of the β enlargement is estimated as follow:

$$S_{eti} = \{\left|t_q - [t_q]_l\right|, T_{sam} - \left|t_q - [t_q]_l\right|, ..., (\beta - 1) * T_{sam} - \left|t_q - [t_q]_l\right|\}.$$

Here, $\left|t_q - [t_q]_l\right|$ is the enlargement time interval of the phase which t_q belongs to, while the $k * T_{sam} - \left|t_q - [t_q]_l\right|$ is the enlargement time interval of the kth phase of the $\beta - 1$ phases before. Therefore, the biggest enlargement time interval is $MAX(\left|t_q - [t_q]_l\right|, (\beta - 1) * T_{sam} - \left|t_q - [t_q]_l\right|)$. The enlargement time interval increases with β. The bigger β is, the larger a query window is expanded, and the higher the partition traverse cost is. In summary, we can see that the query parameter β is the most important issue of the range query costs. It not only decides the times of query window enlargement, but also effects how large a query window is expanded to.

As for KNN queries of the B^s-tree, since a KNN query is handled as incremental range queries, the I/O cost can be approximately estimated as the I/O cost of the last range query. However, it will be a little bigger. This is because, although the range queries in a KNN query of the B^s-tree do not check the same key values repeatedly, a continuous key range within the expanded window of the last range query may be divided by different range queries. Some additional unique scans are asked for, and some leaf nodes shared by different range queries are accessed several times. However, for the cost of KNN queries, the query parameter β is still the most important issue. To sum up, we can tunes the query performance of the B^s-tree by tuning β.

In addition, from the above analysis, we can also observe that since $T_{sam} = T_m/n$, n effects both update performance and query performance. A larger n results in more partitions and may require more update costs in the B^s-tree. While a smaller n results in larger query window enlargement, which increases the query cost. In this paper we choose $n = 3$, as the previous experiments of B+-based-tree of moving objects in [4] show that it is an appropriate balance value.

4.2 Self-tuning Framework

From the above analysis, we find that the update and query performance of the B^s-tree can be tuned by the update parameter α and the query parameter β. However, α and β can not be changed arbitrarily. We should guarantee the correctness of the queries.

Theorem 1. *As long as β is no less than all α of all moving objects, the query correctness of the B^s-tree can be guaranteed. That is, given a query, the B^s-tree can find all moving objects which satisfy the query in their t_{per}s.*

Proof: For a moving object $o(t_{ref}, t_{per}, \alpha_o)$, where α_o is its current update parameter, it will be sampled at following time:

$$\{[t_{ref}]_l, [t_{ref}]_l + \alpha_o * T_{sam}, [t_{ref}]_l + \alpha_o * 2 * T_{sam}, ..., [t_{ref}]_l + \alpha_o * m * T_{sam}\}.$$

Here, $[t_{ref}]_l + \alpha_o * m * T_{sam}$ is the biggest t_{lab} which is not bigger than $t_{ref} + t_{per}$. Given a future query $Q(t_q)$ where $t_{ref} \leq [t_q]_l \leq t_{ref} + t_{per}$, assuming that o satisfies Q at t_q and the query parameter $\beta < \alpha_o$, if $[t_q]_l = [t_{ref}]_l + (\alpha_o * i + j) * T_{sam}$ where $i < m$ and $\beta \leq j < \alpha_o$, the B^s-tree will not search any partition which o is sampled to, therefore will not find o and the result is not correct. In reverse, if β is no less than all α of all moving objects, the B^s-tree will not omit any partition which any result object is sampled to. As long as the partition is traversed, the method of query window enlargement guarantees that any moving object which satisfies the query in this partition will be found. Proved. ■

Note that, if α of the B^s-tree changes, the moving objects which are already in the index have the old α, while a new insertion or update is according to the new α. Therefore, in the index, the moving objects may have different α. According to Theorem 1, β of the B^s-tree should be equal to the biggest α of all moving objects to guarantee the query correctness. Bigger β is meaningless, as it increases query costs. If all moving objects in the B^s-tree have the same α, we call the B^s-tree is in a *steady state*, and $\beta = \alpha$. Also note that, there is some restriction for the longest update interval of a moving object. In the B^s-tree, the time interval T_m should be larger than most $t_{per}s$ of all moving objects in the system. In addition, similarly to [8] [5], for those rare moving objects who are not updated in the last T_m, they are "flushed" to new partitions. The new positions are estimated using their last updated positions and velocities. Their new $t_{per}s$ are the rest $t_{per}s$. However, their α could be changed. Thus, the longest update or flush interval of a moving object is restricted to T_m.

Now we introduce how the online self-tuning framework works. In the framework, the cost we focus on is the I/O cost, since it is usually more important than the CPU cost. Recall that, as shown in figure 1, OTM is the key module which monitors the performance of the B^s-tree and controls the tuning. OTM works in a "self-learning and self-tuning" way. And it has three tasks. First, it maintains a histogram $H_{rec} = \{m, n, C_{upd}, C_{que}\}$ which includes the number and average cost of updates and queries during the recent period of time T_{rec}, represented as m, n, C_{upd}, C_{que}. T_{rec} is the "self-learning" time window. It should be a multiple of T_{sam}, $T_{rec} = l * T_{sam}$. Specially, in order to get H_{rec}, OTM maintains the number and total cost of updates and queries of the l most recent T_{sam}. Thus, H_{rec} can be easily carried out at the end timestamp of each T_{sam}. T_{rec} can be set according to different applications, and it can be reset at any time without breaking the index service. In this paper, we set $T_{rec} = 2 * T_{sam} = 2 * n * T_m$.

Second, OTM also maintains a set S_{base} which includes the average cost of updates and queries for each steady state of the B^s-tree, ranging $\alpha = \beta$ from 1 to $n + 1$.

$$S_{base} = \{C^1_{upd}, C^1_{que}, C^2_{upd}, C^2_{que}, ..., C^{n+1}_{upd}, C^{n+1}_{que}\}$$

For initialization, we give predictive values for the elements in S_{base}. Then, when in a steady state, the corresponding elements of the current $\alpha = \beta$ are updated by C_{upd}, C_{que} in H_{rec} at the end timestamp of each T_{sam}. The initialized predictive values in S_{base} can be given by experiments. We give the guide line of setting the initialized S_{base} in the experiment department.

Third, OTM adjusts the B^s-tree in a self-tuning way. As shown in figure 1, there is a Timer trigger for OTM. At the end timestamp of each T_{sam}, OTM not only updates H_{rec} and S_{base}, but also starts a *checking procedure*. It decides weather to tune the update parameter α and query parameter β of the index. The overall performance is the average performance of each operation. It is estimated by the following equation:

$$C_{all} = \frac{m * C^{\alpha}_{upd} + n * C^{\beta}_{que}}{m + n}.$$

Here, m is the number of updates and n is the number of queries. OTM uses the current m and n in H_{rec}, and different C^{α}_{upd} and C^{β}_{que} in S_{base} for each $\alpha = \beta$ to compute C_{all}. Then, it chooses $\alpha = \beta$ which gets the lowest C_{all}. In order to avoiding thrash, if this C_{all} is 15% lower than the C_{all} with the current $\alpha = \beta$, OTM will decide to tune α and β, and enter a *tuning procedure*. Note that, the checking procedure is efficient, since there are only a little computation and comparison during the checking procedure.

For the tuning procedure, the current update parameter and query parameter are represented as α_{cur} and β_{cur}, while the target update parameter and query parameter which OTM decides to tune to is represented as α_{tar} and β_{tar}. In addition, α_{tun} and β_{tun} are the update parameter and query parameter during the tuning procedure. Since the current and target states both are steady states, we have $\alpha_{cur} = \beta_{cur}$ and $\alpha_{tar} = \beta_{tar}$. If $\alpha_{tar} < \alpha_{cur}$, at the beginning of the tuning procedure, OTM sets $\alpha_{tun} = \alpha_{tar}$ and $\beta_{tun} = \alpha_{cur}$. During the tuning procedure, the B^s-tree will include different moving objects with α_{cur} or α_{tar}. Thus, β_{tun} has to be equal to the biggest update parameter of all available partitions, α_{cur}, to guarantee the query correctness. Note that, a moving object will be updated or flushed at least once during the last T_m. Therefore, after time period of T_m, the update parameter of all moving object will be the new one α_{tar}, the B^s-tree will reach a steady state, and OTM tunes the query parameter to $\beta_{tar} = \alpha_{tar}$. While, on the other hand, if $\alpha_{tar} > \alpha_{cur}$, at the beginning of the tuning procedure, OTM sets $\alpha_{tun} = \beta_{tun} = \alpha_{tar}$. After time period of T_m, the B^s-tree reaches a steady state, with no need to reset the query parameter. The length of the tuning procedure is T_m. During the tuning procedure, OTM does not update H_{rec} and S_{base}, and does not enter the checking procedure. When the tuning procedure finishes, OTM resets H_{rec} to empty and continues its work. Note that, the index service is not broken.

Although OTM has the ability of self-tuning, it allows for manual configuration. This enhances the flexibility and usability of the B^s-tree. The administrator can set $\alpha_{tar} = \beta_{tar}$ at any time. OTM will then enter the tuning procedure at the end timestamp of the current T_{sam}. On the other hand, the administrator can also “freeze” the B^s-tree to avoid too frequent tuning. As a result, OTM will not enter the checking and tuning procedures. In addition, the initial value of $\alpha = \beta$ when creating the B^s-tree, is also customizable and should be set according to specific applications.

5 Experiments

In this section, we experimentally compare the B^s-tree with the B^x-tree [8] and the TPR*-tree [11], which are the most representative B+-tree and R-tree based indexes for moving objects. All experiments are implemented in the C++ language, and conducted on a 2.6GHz Pentium 4 Personal Computer with 1GB memory, running Windows XP Professional. The page size and index node size are both set to 4 KB.

We use a data generator similar to the one used by the B^x-tree [8]. The objects move in the space domain of $1000 * 1000$. The initial object positions are generated randomly, so are the moving directions. The moving speed in each dimension is selected randomly from -3 to 3. The update frequencies (indicated by t_{per}) of the moving objects are variable among various moving objects. 35% of the moving objects have $t_{per}s$ ranging from 0 to $1/3T_m$, while another 35% are in range $(1/3T_m, 2/3T_m]$. And the remaining 30% moving objects are in $(2/3T_m, T_m]$. As the experiments in [8], T_m is set to 120 time units. For the B^x-tree and the B^s-tree, we choose $n = 3$ and use the Hilbert-curve as the space-filling curve. For the TPR*-tree, the horizon H (how far the queries can "see" in the future) is set to 60 time units. For each dataset, we execute same 100 predictive queries and evaluate their average cost. The query time q_t ranges from 1 to H. For range queries, the side length of query windows is chosen from 10 to 50.

5.1 Basic Performance

In this set of experiments, we study the basic query and update performance of the B^s-tree, the B^x-tree and the TPR*-tree. Figure 5 $(a)-(d)$ show the average number of I/Os and CPU time for each range query and update at various dataset sizes, ranging from $100K$ to $500K$. For a more clear view, we represent the query and update performance of the B^s-tree for different steady states. B^s-tree(j) indicts the B^s-tree in the steady state with $\alpha = \beta = j$, from 1 to $n+1 = 4$. As expected, when $\alpha = \beta = 4$, the query and update performance of the B^s-tree is very similar with that of the B^x-tree. This is because both the B^s-tree(4) and the B^x-tree sample once for an update and expand query windows $n + 1 = 4$ times for a range query. Therefore, in these figures, we omit the query and update performance result of the B^s-tree(4), using that of the B^x-tree instead. From these experiments, we can observe that the B^x-tree has better update performance than the TPR*-tree, while the TPR*-tree has better query performance than the B^x-tree. The B^s-tree has the ability to tune the query and update performance by different α and β. It

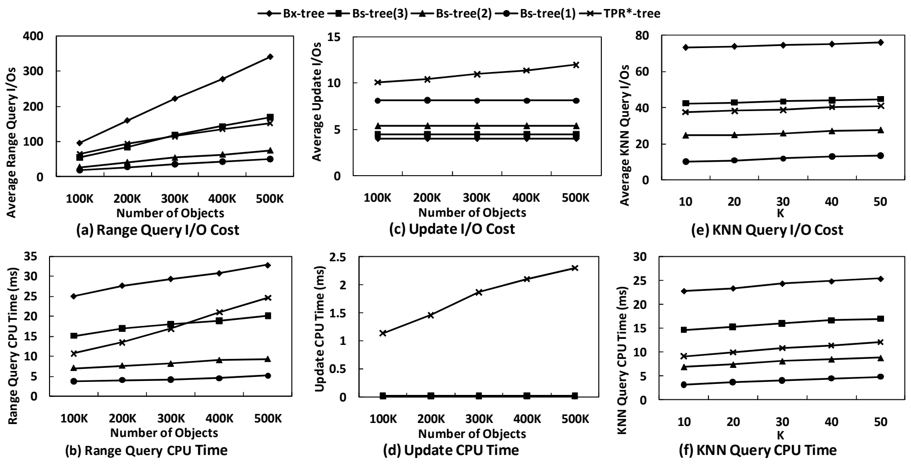


Fig. 5. Basic Query and Update Performance

can have almost the same query and update performance as the B^x-tree when update cost is the major part of the overall cost, while in other cases, it can pay slightly more update cost to achieve considerably higher performance of queries. Therefore, it “dominates“ the B^x-tree. In addition, when $\alpha = \beta = 2$ or $\alpha = \beta = 3$, the B^s-tree has both better query performance and update performance than the TPR*-tree. It “dominates“ the TPR*-tree. We can also see that the cost of queries increases with the date cardinality for all the three kinds of indexes. However, for the B^x-tree and the B^s-tree, the update cost is not apparently affected by the dataset size. In addition, for all the three kinds of indexes, the update cost is much lower than the query cost. In the rest experiments, the default dataset size is $100K$. Figure 5 (e) and (f) shows the average number of I/Os and CPU time per KNN query while varying K from 5 to 50. Observe that the effect of k is not very significant. The relationship of the performance of the three indexes is similar with that of the range query performance.

5.2 Overall Performance

In this experiment, we study the overall performance of the three trees by combining both update and query operations in the same workload. For the query operations, the range queries and KNN queries are mixed with the proportion of $5 : 1$, considering that the range queries are more common. We vary the ratio of the number of queries over the number of updates from $1 : 1000$ to $100 : 1$. Figure 6 shows the average overall I/Os and CPU time for each operation. For the B^s-tree, S_{base} of OTM is initialized using the results in subsection 5.1. The overall performance of the B^s-tree with different proportion of updates and queries was recorded when the B^s-tree reached the steady states. From this experiment, we can observe that the B^s-tree adjust its performance in a self-tuning way. When the total update cost is much higher than the total query cost, it automatically pays some more query cost to achieve lower update cost. While, oppositely, when the total query cost is much higher than the total update cost, it automatically pays some more update cost to achieve better query performance. In this way, the B^s-tree keeps good and smooth overall performance despite the change of the workload. It can be seen that in almost all cases, the B^s-tree outperforms the TPR*-tree and the B^x-tree in overall performance.

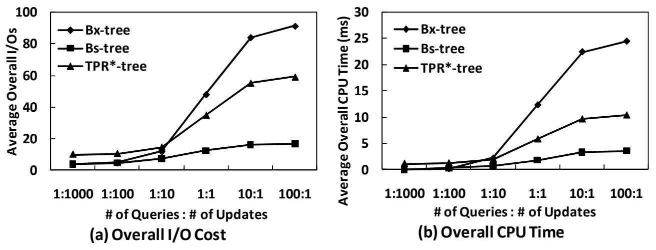


Fig. 6. Overall Performance

5.3 Self-tuning Performance

The above experiments show that the B^s-tree has good performance when it reaches the steady states. However, if the performance degrades markedly when the B^s-tree is in the tuning procedures, it is still not appropriate for real use. In this experiment, we study the self-tuning performance of the B^s-tree. We vary the proportion of the number of queries over the number of updates from $1 : 100$ to $100 : 1$, and then return to $1 : 100$. Correspondingly, $\alpha = \beta$ of the B^s-tree tunes from initial value 4 to 1, and then return to 4. Note that as shown in section 5.2, since the query cost is much higher than the update cost, the overall cost will increase with the ratio of queries, though it is much slighter for the B^s-tree than for the TPR*-tree and the B^x-tree. Figure 7 shows the average overall I/Os and CPU time for each operation. Here T indicts the performance when the B^s-tree is in the tuning procedures, while S indicts the performance when the B^s-tree reaches the steady states. We can see that the self-tuning cost of the B^s-tree is slight. During the tuning procedures, the B^s-tree pays only a little more cost than that of reaching the steady state later. And note that the length of a tuning procedure is only T_m. In addition, we can see that the extra tuning cost of the B^s-tree from a higher $\alpha = \beta$ to a lower $\alpha = \beta$ is slighter than that from a lower $\alpha = \beta$ to a higher $\alpha = \beta$, since the methods of tuning procedure are different. In conclusion, The B^s-tree can provide non-break index service with good and smooth performance.

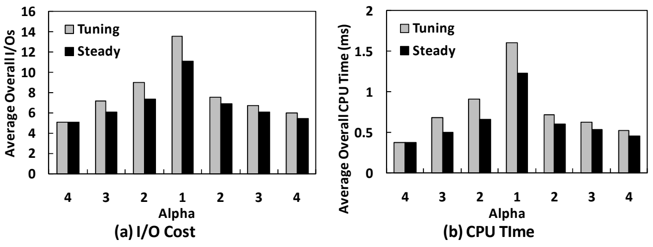


Fig. 7. Tuning Performance

5.4 Concurrency Performance

Finally, we study the concurrency performance, using a multi-thread program to simulate multi-user environments. To highlight the difference between the two B+-tree based indexes, we do not show the result for the TPR*-tree since it has been shown to be inefficient in a concurrent environment in [8] and [6]. The B-link technique [10] is used as the concurrency control technique for the B^x-tree and the B^s-tree. We use a workload varying the proportion of the number of query operations over the number of update operations from $1 : 100$ to $100 : 1$, and then return to $1 : 100$, with initial $\alpha = \beta = 4$. Figure 8 shows the average throughput and response time of the whole workload, while varying the number of threads from 1 to 6. Throughput is the rate at which operations could be served by the system and response time is the time interval between issuing an operation and getting the response from the system when the task is successfully completed. As expected, the B^s-tree outperforms the B^x-tree, basing the same B+-tree structure and the same concurrency control technique. This is because when the update operations dominate the overall performance, the B^s-tree with $\alpha = \beta = 4$ has almost the same update and query performance. While, in other cases, the B^s-tree with smaller $\alpha = \beta$ pay slight more update cost to achieve significantly higher query performance, which results much better overall performance than the B^x-tree.

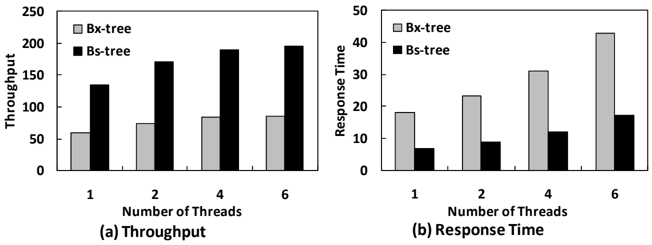


Fig. 8. Concurrent Performance

6 Conclusion

With the development of moving object databases, requirements to handle the applications, where the ratio of update and query operations varies widely with time, are becoming essential. In this paper, we propose a moving object index structure, namely B^s-tree. The B^s-tree has the ability of adapting its update and query performance to meet different requirements. We implement various algorithms for the B^s-tree. In addition, we present an online self-tuning framework which provides self-tuning for optimal overall performance without interrupting the indexing service. Our experiment studies show that, the B^s-tree achieves good and smooth overall performance with efficient self-tuning procedures. Therefore, the proposed B^s-tree is efficient and suitable for dynamic applications in which the proportion of query and update operations varies significantly by time. For future work, we would study other self-tuning techniques for moving object databases.

Acknowledgement

This work was supported in part by the National Science Foundation of China (NSFC Grant No. 60803003, 60970124) and by Chang-Jiang Scholars and Innovative Research Grant (IRT0652) at Zhejiang University.

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 Conference, Atlantic City, NJ, May 1990, pp. 322–331 (1990)
2. Benetis, R., Jensen, C.S., Karciauskas, G., Saltenis, S.: Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. VLDB J. 15(3), 229–249 (2006)
3. Chaudhuri, S., Narasayya, V.R.: An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In: VLDB, Athens, Greece, August 1997, pp. 146–155 (1997)
4. Chen, N., Shou, L.-D., Chen, G., Dong, J.-X.: Adaptive Indexing of Moving Objects with Highly Variable Update Frequencies. Journal of Computer Science and Technology (JCST) 23(6), 998–1014 (2008)
5. Chen, S., Ooi, B.C., Tan, K.-L., Nascimento, M.A.: S2TB-Tree: A Self-Tunable Spatio-Temporal B+-Tree Index for Moving Objects. In: SIGMOD Conference, Vancouver, BC, Canada, June 2008, pp. 29–42 (2008)

6. Guo, S., Huang, Z., Jagadish, H.V., Ooi, B.C., Zhang, Z.: Relaxed Space Bounding for Moving Objects: A Case for the Buddy Tree. SIGMOD Record (SIGMOD) 35(4), 24–29 (2006)
7. Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: SIGMOD Conference, Boston, Massachusetts, June 1984, pp. 47–57 (1984)
8. Jensen, C.S., Lin, D., Ooi, B.C.: Query and Update Efficient B+-Tree Based Indexing of Moving Objects. In: VLDB, Toronto, Ontario, Canada, August 2004, pp. 768–779 (2004)
9. Saltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the Positions of Continuously Moving Objects. In: SIGMOD Conference, Dallas, Texas, USA, May 2000, pp. 331–342 (2000)
10. Srinivasan, V., Michael, Carey, J.: Performance of B-Tree Concurrency Algorithmss. In: SIGMOD Conference, Denver, Colorado, May 1991, pp. 416–425 (1991)
11. Tao, Y., Papadias, D., Sun, J.: The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. In: VLDB, Berlin, Germany, September 2003, pp. 790–801 (2003)
12. Tao, Y., Zhang, J., Papadias, D., Mamoulis, N.: An Efficient Cost Model for Optimization of Nearest Neighbor Search in Low and Medium Dimensional Spaces. IEEE Trans. Knowl. Data Eng. (TKDE) 16(10), 1169–1184 (2004)
13. Yiu, M.L., Tao, Y., Mamoulis, N.: The Bdual-Tree: indexing moving objects by space filling curves in the dual space. VLDB J. (VLDB) 17(3), 379–400 (2008)

Privacy-Preserving Location Publishing under Road-Network Constraints

Dan Lin, Sashi Gurung, Wei Jiang, and Ali Hurson

Missouri University of Science and Technology
{lindan,sgy99,wjiang,hurson}@mst.edu

Abstract. We are experiencing the expanding use of location-based services such as AT&T TeleNav GPS Navigator and Intel's Thing Finder. Existing location-based services have collected a large amount of location data, which have great potential for statistical usage in applications like traffic flow analysis, infrastructure planning and advertisement dissemination. The key challenge is how to wisely use the data without violating each user's location privacy concerns. In this paper, we first identify a new privacy problem, namely *inference route* problem, and then present our anonymization algorithms for privacy-preserving trajectory publishing. The experimental results have shown that our approach outperforms the latest related work in terms of both efficiency and effectiveness.

1 Introduction

The extensive use of location-based services, such as AT&T TeleNav GPS Navigator, Sprint's Family Locator, and Intel's Thing Finder, have collected a large amount of location data. If information like vehicle IDs and moving directions on roads can be published, people in many fields will benefit from it. With respect to the public sector, traffic flow information can be extracted from published IDs and moving directions. Such information will play an important role in infrastructure construction and traffic light control. With respect to the business domain, traffic information can help decide the location of company branches, and also advertisements can be customized and disseminated at the most advantageous locations. With respect to our daily lives, traffic information is certainly useful for detecting and predicting traffic jam, and calculating better routes in an emergency (e.g., for ambulances). However, in the meantime, location privacy concerns [11,16] may hinder the development of such attractive usage of traffic information. It is well known that using a pseudonym is not sufficient to prevent the linkage of a published location to a real ID [5]. The key challenge is how to wisely use the location data without violating each user's privacy concerns. This problem is termed as *privacy preserving historical location data publishing*.

Historical location data forms a sequence of locations in chronological order, termed as *trajectory*. In general, one's trajectory consists of roads he has visited. For instance, in Figure 1, user u_1's trajectory can be represented as $IABC$ and user u_4's trajectory is ABD. Such road-network based trajectories are valuable in aforementioned applications. In privacy-preserving location publishing, the goal is to prevent adversaries from mapping published locations to a specific individual.

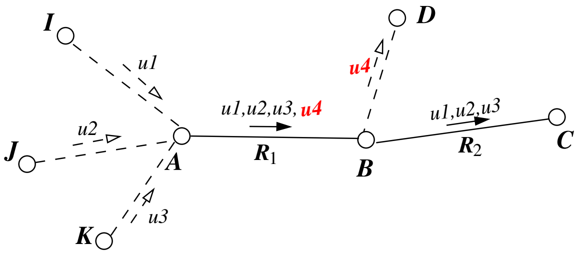


Fig. 1. An Example of Inference Route

One may think that a trajectory resembles a conventional sequential pattern. Hence, a naturally raised question is that if we can directly employ privacy preserving data publishing approaches [3,4,13,20] developed in non-spatial-temporal databases? The answer is negative, and the main reason is that a trajectory distinguishes itself from the conventional sequential patterns due to additional constraints (e.g., road-network information) which do not exist in the traditional sequences. More specifically, elements in traditional sequences are usually independent of one another, while the relationship of elements in the trajectory sequence is fixed under a particular road-network information. Therefore, we cannot use traditional algorithms to arbitrarily remove or replace elements in the sequences because such operations will create unrealistic trajectories consisting of non-connected road segments.

There have been several recent efforts [2,7,12,17] on anonymizing trajectories. Some work [17] considers trajectories as a sequence of landmarks, e.g., stores and museums, which ignore the paths connecting these places. Others [2,7,12] consider trajectories as a sequence of coordinates in Euclidean space but ignore the road-network constraints. Very few works considered the road-network constraint. The most recent one is by Pensa et al. [14], who anonymize road-network-based trajectories based on k-anonymity [15]. However, their approach may not preserve trajectory information as much as possible. This can be demonstrated by the example given below.

In [14], trajectories are stored and anonymized by using a prefix tree which may not be an appropriate structure to model the road-network. For instance, consider four users who leave their homes (I, J, K, D) and head for work. When k is 3 and the input to their algorithm is the following four trajectories: $u_1(IABC)$, $u_2(JABC)$, $u_3(KABC)$ and $u_4(ABD)$[1], their anonymization result will be an empty set since the prefix tree treats trajectories with different starting points independently. Such result obviously lost too much useful information. To achieve better information utility, an alternative way is to directly take partial trajectories as input, i.e., consider only busy roads with more than k users. In this case, the input becomes $u_1(ABC)$, $u_2(ABC)$, $u_3(ABC)$ and $u_4(AB)$, and the new anonymization result is : $u_1'(ABC)$, $u_2'(ABC)$, $u_3'(ABC)$ and $u_4'(AB)$, which is more meaningful than the previous empty set.

In addition, since road maps can be found everywhere, in the domain of privacy-preserving location publishing, it is reasonable to assume road-network information is available to any adversary. Thus, cautions are very much needed when publishing

[1] u_1, u_2, u_3 and u_4 can be thought as either a trajectory ID or a person's symbolic ID.

anonymized trajectories. For instance, let us continue from the previous example and assume that the road-network in Figure 1 is accessible to an adversary Bob. When u'_1, u'_2,u'_3 and u'_4 are published, using the road-network, Bob can infer that u'_4 was also travelled on the road segment $\overline{BD}$. Also, if Bob knows that Alice usually travels on $\overline{BD}$, then he can link u'_4 to Alice and consequently track Alice remaining trajectories in the published dataset. This *inference route problem* is caused by the fact that an adversary can infer someone's unpublished infrequent trajectories from the published location dataset. Because the inferred trajectories are infrequent, with high probability, these trajectories, combined with certain external knowledge, can be used to identify a particular individual's trajectory information in the published dataset. In general, given a threshold k, if the attacker can link any anonymous ID to Alice with probability greater than $\frac{1}{k}$ by using the above method, then we say there is an inference route problem.

In this paper, we address the problem of privacy-preserving location data publishing under the assumption that road-network data are public information. Our approach has three main properties: (1) it guarantees k-anonymity of published data, (2) it avoids the inference route problem, and (3) the anonymization results follow the road-network constraints. The basic idea is to employ a clustering-based anonymization algorithm to group similar trajectories and minimize the data distortion caused by anonymization through a careful selection of representative trajectories. We propose a C-Tree (Cluster-Tree) to speed up the clustering process and develop methods to incrementally calculating error rates. The rest of the paper is organized as follows: Section 2 reviews related work, Section 3 presents our proposed approach, Section 4 reports experimental results, and Section 5 concludes the paper with lessons learned and future research directions.

2 Related Work

Privacy-preserving location publishing is a relatively young area in which little research has been carried out. In [7,12], the spatial-temporal cloaking technique is applied to generate cloaking regions covering segments of trajectories. In [2], Abul et al. consider a trajectory as a cylindrical volume where the radius represents the location imprecision. Then they perturb and cluster trajectories with overlapping volumes to ensure that each released trajectory volume encloses at least $k - 1$ other trajectories. Unlike the previous work which is based on the similarity of trajectories, Yarovoy et al. [19] group trajectories based on so-called quasi-identifiers which is hard to be selected in practice. None of the approaches considers the impacts of road network constraints and hence, their anonymization results are vulnerable to attack when the malicious party knows the road map or holds some other background knowledge. E.g., if a cloaking region covers only one road, the corresponding trajectory can be easily mapped to the road.

In [17], Terrovitis and Mamoulis assume that the adversaries know partial trajectory information of some individuals. They use it as part of input to their anonymization algorithm. Such usage limits the generality and feasibility of their approach. In [1], Abul et al. used a coarsening strategy which removes one or more spatial points in a trajectory to achieve anonymization. An anonymized trajectory may contain disconnected paths. This is different from our approach which preserves continuous trajectories based on road-network information. Two other related works used time confusion and path confusion respectively. The time confusion approach [9] mixes location samples of different trajectories, and the path confusion approach [8] crosses paths in areas where at least two users meet. The main problem of the two approaches is that traffic flows are no longer preserved.

The most related work is by Pensa et al. [14]. They proposed a prefix-tree based anonymization algorithm which guarantees k-anonymity of the published trajectories in a way that no trajectories with support less than k will be published. They defined the support of a trajectory Trj as the number of trajectories containing Trj, which however causes the inference route problem. Here, we can see that how the concept of k-anonymity is applied will affect the quality of the anonymization result.

3 Problem Statement

In general, raw data collected by location-based applications contains user (object) information as a four-tuple $\langle ID, loc, vel, t\rangle$, where ID is the object ID, loc and vel are object location and velocity at timestamp t respectively. The anonymized dataset contains object information in the form of $\langle aid, rid, dir\rangle$, where aid is an anonymized object ID, rid is a road ID and dir is the object's moving direction. Here, for privacy concerns, we replace specific locations and velocities by road ID and moving direction. Such representation is sufficient to derive trajectories or traffic flow information.

The road network is modeled as a directed graph, where each edge corresponds to a road and each node represents an intersection. Specifically, an edge is represented as $\overline{n_i n_j}$, where n_i and n_j denote nodes. We then proceed to define the frequent road and inference route problem.

Definition 1. *Let W be a time interval, and let k be a threshold. We say a road is a frequent road if the number of moving objects moving along one direction on this road is no less than k within time W. We call the number of moving objects the frequency of the road.*

Definition 2. *Let Υ be an intersection of roads r_1, ..., r_m, and let U_i^+, U_i^- be the sets of objects moving toward and outward Υ on road r_i ($1 \leq i \leq m$) during W, respectively. If $\exists$ U_i^+, U_j^-, $|U_i^+| \geq k$, $|U_j^-| \geq k$, and ($0< |U_i^+ - U_j^-| < k$ or $0< |U_j^- - U_i^+| < k$), then we say Υ has an **inference route** problem.*

To have a better understanding of the above definition, let us revisit the example in Figure 1. Node B is an intersection of three roads. On road $\overline{AB}$, $U_{AB}^+ = \{u_1, u_2, u_3, u_4\}$; on road $\overline{BC}$, $U_{BC}^- = \{u_1, u_2, u_3\}$. Since $U_{AB}^+ - U_{BC}^- = \{u_4\}$, $|U_{AB}^+ - U_{BC}^-| = 1 < k$, node B has an inference route problem.

Next, we present how to evaluate the quality of the anonymized dataset or trajectories. Intuitively, the less difference between the anonymized dataset and the original dataset, the better quality the anonymized dataset is. Therefore, we use two common metrics: average error rate and standard deviation. Suppose there are N roads (or edges in a road-network graph) and r_i represents road i. Let $original_{r_i}$ and $anonymized_{r_i}$ denote r_i's original frequency and frequency after the trajectories have been anonymized. Then in Equation 1, the error function E is defined as the average difference between $original_{r_i}$

and $anonymized_{r_i}$ (i.e., E_i), and σ is the standard deviation of the error rates. A low standard deviation indicates that the anonymization quality of each road is similar and close to the average error rate.

$$E = \frac{1}{N}\sum_{i=1}^{N} E_i = \frac{1}{N}\sum_{i=1}^{N}\frac{|original_{r_i} - anonymized_{r_i}|}{original_{r_i}} \tag{1}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(E_i - E)^2} \tag{2}$$

4 Our Approach

In this section, we present our anonymization algorithm. It consists of two main steps. First, from the raw dataset D, we remove records associated with infrequent roads, i.e., roads with less than k objects within a given time interval. We denote the obtained dataset as D'. In D', we construct partial trajectories for the remaining objects based on moving directions. Note that one user may have several disconnected partial trajectories because he may visit some infrequent roads. Each partial trajectory will be assigned an anonymous ID. For the rest of the paper, the word "trajectory" and "partial trajectory" are interchangeable.

The second step is the core of the anonymization process. We propose a clustering-based anonymization algorithm which guarantees that by achieving strict k-anonymity (defined in Section 4.1) among partial trajectories, our anonymization result is free of the inference route problem. Compared to traditional k-anonymization approaches, our approach not only needs to minimize errors caused by anonymization but also needs to satisfy some unique requirements. Road-network constraints should be enforced during the entire anonymization process, especially when computing the representative trajectories. The first step is relatively straightforward. Therefore, the following discussion focuses on the anonymization step.

4.1 Clustering-Based Anonymization

The essential idea of clustering-based anonymization algorithm is to find clusters of similar trajectories and anonymize them by using a representative trajectory. The details are the following.

First, we need to select a proper way to represent trajectories. Trajectories are initially represented as a sequence of timestamped locations. In our anonymized dataset, we do not disclose exact locations because detailed information increases attackers' chances to link published location to specific individuals. Instead, we report only information about which object passing by which road. There are two options: (i) representing a trajectory by road IDs; or (ii) representing a trajectory by node IDs. As illustrated in Figure 2, trajectories Trj_1, Trj_2 and Trj_3 can be represented as r_4r_2, r_1r_3, and r_1r_5 respectively following the first option. Using the second option, trajectories Trj_1, Trj_2 and Trj_3 can be represented as $n_5n_2n_3$, $n_1n_2n_4$, and $n_1n_2n_6$ respectively. Both

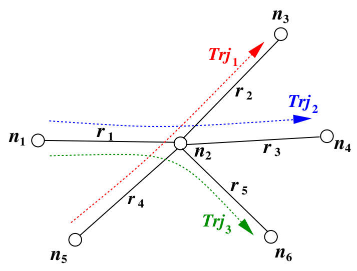


Fig. 2. Trajectory Representation

types of representations well capture the similarity between trajectories Trj_2 and Trj_3 which share one common road. However, the first option treats Trj_1 and Trj_2 as two irrelevant trajectories even though they intersect. To better reflect relationships among trajectories, we adopt the second option and represent a trajectory by a sequence of node IDs.

The second issue is to define the distance between trajectories. We employ the *edit distance* [18]. The edit distance between two trajectories is given by the minimum number of operations needed to transform one trajectory into the other, where an operation is an insertion, deletion, or substitution of a node. For example, the edit distance between $Trj_1(n_5n_2n_3)$ and $Trj_2(n_1n_2n_4)$ is 4, while the distance between Trj_2 and $Trj_3(n_1n_2n_6)$ is 2.

Now we are ready to present our clustering-based anonymization algorithm. An outline is given in Figure 3. First, we group same trajectories and count its *support*. Support is defined as the number of users who have the same trajectories (Definition 3).

Definition 3. *Let u be a user's anonymous ID and Trj_u denote his trajectory in D'. We have the support of trajectory Trj as follows: Support(Trj) = $|\{u|Trj_u = Trj,$ for every $u\}|$.*

Distinct trajectories are arranged in a descending order of their supports. If a trajectory's support is more than the anonymization threshold k, the trajectory itself forms a cluster. For the remaining trajectories, say Trj, we compare it with existing clusters. If there exists a suitable cluster, we insert the new trajectory into that cluster and update the cluster's information. Otherwise, a new cluster will be created for Trj. After all trajectories have been checked, we translate representative trajectories together with their supports into output format, which contains object anonymious IDs, road IDs, and objects' moving directions. For example, we obtain the following intermediate result after anonymizing the trajectories shown in Figure 1: $u_1'(ABC)$, $u_2'(ABC)$, $u_3'(ABC)$ and $u_4'(ABC)$, where $k = 3$. The published dataset will look like this: $(u_1', R_1, \overline{AB})$, $(u_1', R_2, \overline{BC})$, $(u_2', R_1, \overline{AB})$, $(u_2', R_2, \overline{BC})$, ..., $(u_4', R_2, \overline{BC})$. The detailed algorithms for finding candidate clusters, calculation of error rates and selection of representative trajectories will be elaborated in the rest of the section.

Our approach ensures strict k-anonymity (Definition 4) over all trajectories in dataset D'. It is called "strict" because the calculation of trajectory supports is based on an exact match of entire trajectories. In this way, we guarantee that the anonymization result will not contain any inference route. Our proof can be found in [10].

```
Clustering-based Anonymization ($TRJ$, $k$)
Input: $TRJ$ is a set of trajectories to be $k$-anonymized
1.  Group same trajectories and form $TRJ'$
2.  Sort trajectories in $TRJ'$ in a descending order of supports
3.  for each $Trj$ in $TRJ'$ do
4.      if $Trj.support \geq k$ then
5.          create a new cluster for $Trj$
6.      else
7.          check existing clusters
8.          if Find_Cluster($Trj$,$C$) then
9.              insert $Trj$ to cluster $C$
10.             Select_Representative_Trajectory($C$,$Trj_r$)
11.             update $C$'s error rate
12.             update $C - tree$
13.         else
14.             create a new cluster for $Trj$
15. for each cluster C in group of clusters
16.     if $C.Total\_TRJ \geq k/2$ then set $C.Total\_TRJ = k$
17.     else remove C from group of clusters
18. Translate representative trajectories into output format
```

Fig. 3. An Outline of Clustering-based Anonymization Algorithm

Definition 4. *(Strict k-anonymity over trajectories): Let Trj be a trajectory. We say Trj satisfies strict k-anonymity if Support(Trj) is no less than k.*

4.2 Finding Candidate Clusters

In this subsection, we present how to find a candidate cluster for a new trajectory during the clustering-based anonymization. The first step is to check whether a new trajectory can be absorbed by an existing cluster according to the distance metric. As the number of clusters increases, comparing Trj with all clusters becomes very costly. Therefore, we employ an in-memory index structure, C-tree (Cluster-tree), to prune unnecessary comparisons. In particular, each node in the C-tree contains multiple entries and each entry in a node has two fields: a pointer ptr and a set of road IDs (denoted as RID). In leaf nodes, each entry has a pointer to a cluster and the IDs of roads occurring in that cluster. In internal nodes, each entry has a pointer to a child node and the union of roads IDs in its child node. Figure 4 shows an example C-tree.

Given a new trajectory Trj, starting from the root of the C-tree, we calculate the similarity between Trj and every entry's RID in the node by using the following similarity function.

$$Sim_c(Trj, RID) = \frac{|S(Trj) \cap RID|}{|S(Trj)|} \tag{3}$$

Sim_c computes the percentage of common roads included in Trj and RID, where $S(Trj)$ denotes the set of road IDs in trajectory Trj. If Sim_c is above a threshold ρ,

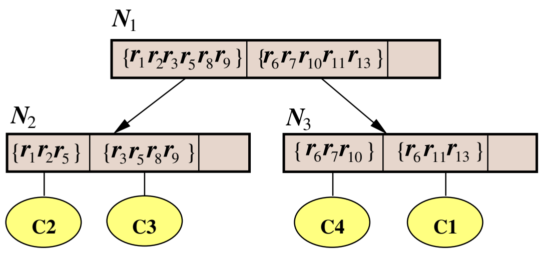


Fig. 4. An Example C-tree

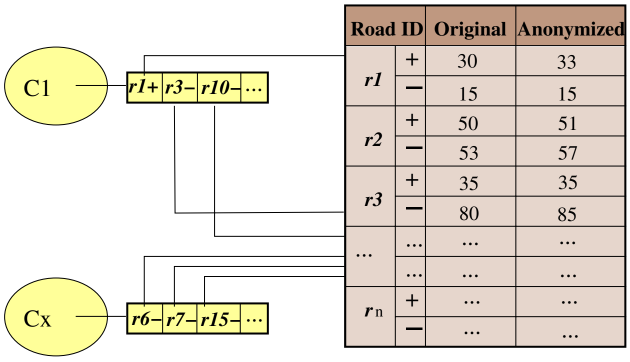
we continue to visit the child node of this entry. This process is repeated until we find all entries in the leaf nodes with Sim_c above the threshold. All the clusters belonging to these entries will be considered as candidate clusters. For example, suppose that a new trajectory contains roads r_2, r_8 and r_9, and the threshold ρ is set to 60%. The similarity Sim_c between the new trajectory and the first and second entries in the root node N_1 are 100% and 0% respectively. The tree below the second entry is pruned and thus we do not need to visit node N_3. We continue to visit the child node N_2 pointed by the first entry. The Sim_c between the trajectory and the first and second entries in N_2 are 33% and 67% respectively. Since the second entry has the similarity score above the threshold, its corresponding cluster $C3$ becomes the candidate cluster for the further consideration.

Among candidate clusters, we further calculate the edit distance between the new trajectory and their representative trajectories. For all clusters which have the shortest edit distance with Trj, we examine the quality of anonymization result (i.e. error rate (E)) by assuming inserting Trj to a cluster. For a cluster C_i, its error rate E_{c_i} is computed based on the roads in this cluster. We select the cluster that satisfies two conditions: (i)

```
Find_Cluster ($Trj$,$C$)
Input: $Trj$ is a trajectory
Output: $C$ is a cluster

1.  $NODE \leftarrow$ {C-tree.root}
2.  while ($NODE$ is not empty) do
3.      for each node $N$ in $NODE$ do
4.          for each entry $en$ in $N$ do
5.              if $Sim_c(Trj, en.RID) > \rho$ then
6.                  if $N$ is not a leaf node then
7.                      add $en$'s child node to $NODE$
8.                  else add $en$'s cluster to candidate list $L_c$
9.  for all clusters in $L_c$ do
10.     find clusters with shortest edit distance with $Trj$
11.     if more than one clusters found then
12.         find the cluster with the smallest error rate
13.     if error rate after adding $Trj$ does not exceed threshold then
14.         return the cluster found
```

Fig. 5. Algorithm of Finding Clusters



Road ID		Original	Anonymized
r1	+	30	33
	–	15	15
r2	+	50	51
	–	53	57
r3	+	35	35
	–	80	85
...	...	...	...
	...	...	...
*r*n	+	...	...
	–	...	...

Fig. 6. Anonymization Table

it yields the smallest error rate after inserting Trj; (ii) its new error rate is below the global threshold Err. Figure 5 summarizes the procedure of finding candidate clusters.

To efficiently and incrementally calculate error rates during clustering, we employ a global data structure, i.e. *anonymization table*. Anonymization table has three fields: *roadID*, *original* and *anonymized*, where “original” is the number of objects before anonymization, and “anonymized” records the latest number of objects on road *roadID* during anonymization. Each cluster only needs to maintain a set of road IDs with pointers referring to the anonymization table. Figure 6 illustrates the data structure.

When actually inserting Trj to C_i, there are three steps: (i) update the representative trajectory; (ii) update the error rate in the anonymization table; and (iii) update the C-tree. The algorithm for selecting the representative trajectory is presented in Section 4.3. Once the representative trajectory is chosen, we recompute the error rate and modify the corresponding field in the anonymization table. Finally, we check whether the node in the C-tree with respect to current cluster needs to be updated. If current cluster contains road IDs which are not included in the road ID list of the corresponding C-tree entry, we will append the new road IDs to the road ID list. This change will be propagated to higher levels of the C-tree until an entry containing all road IDs in current cluster is reached. Consider the C-tree in Figure 4 and suppose that a new trajectory that consists of roads r_2, r_8 and r_9 will be inserted into cluster C_3. We check the road list of C_3's entry in the C-tree, which is $\{r_3 r_5 r_8 r_9\}$ and does not contain r_2. We then add r_2 to the road list. Now the second entry in the C-tree becomes $\{r_2 r_3 r_5 r_8 r_9\}$. Next, we check its parent entry, the first entry in N_1. Since r_2 is included in the first entry in N_1, the tree update operation completes.

If no cluster is similar enough to Trj, we create a new cluster for Trj and follow the three similar steps discussed in the previous paragraph. The main difference is that we need to insert a new entry for this new cluster to the C-tree (the insertion algorithm is in Section 4.4).

4.3 Selecting Representative Trajectory

There are two key requirements when selecting a representative trajectory. First, the error rate should be minimized. Second, the representative trajectory must satisfy the road-network constraint. By keeping these in mind, we design the following algorithm.

In a cluster, we find the trajectory with the highest support and then trim the trajectory from both ends to obtain the final representative trajectory. To illustrate it, we use the example in Figure 7. The cluster contains three types of trajectories: Trj_1, Trj_2 and Trj_3. Each trajectory is associated with a number of support, e.g., $support(Trj_1) = 10$. Numbers on the last line indicates the original numbers of users on each road, e.g., $original(n_1n_2)$=15. Since Trj_1 has the highest support, let us have a further look at it. We compute the error rate E by treating Trj_1 as the representative trajectory. The support of the representative trajectory is the sum of all trajectories in the cluster. The reason behind is to maintain the same amount of trajectories after anonymization. In this example, if we use Trj_1 as the representative trajectory, we will have E = 58%.

Trj_1 (10):	n_1 —	n_2 —	n_4 —	n_7 —	n_8 —	n_9
Trj_2 (5):	n_1 —	n_2 —	n_4 —	n_7		
Trj_3 (6):		n_2 —	n_4 —	n_7 —	n_8	
original:	15	21	21	16	10	

Fig. 7. An Example of Selecting Representative Trajectory

$$E = (E_{n_1n_2} + E_{n_2n_4} + E_{n_4n_7} + E_{n_7n_8} + E_{n_8n_9})/5$$

$$= \frac{\left(\frac{21-15}{15} + \frac{21-21}{21} + \frac{21-21}{21} + \frac{21-16}{16} + \frac{21-10}{10}\right)}{5} = 58\%$$

Observe that $E_{n_8n_9}$ is higher than 100%. If the road n_8n_9 is excluded from the representative trajectory Trj_1, the overall error can be reduced to 34%. Based on this observation, the second step is to trim the trajectory until the overall error rate cannot be further reduced. Due to the road-network constraint, we can not arbitrarily remove nodes from a trajectory. Our strategy is to remove nodes starting from both ends of the selected trajectory if the road r satisfies the following condition: $original_r < support(Trj_1) - original_r$, i.e., its individual error rate larger than 100%. The process continues until we cannot find such a road at either end of the trajectory. The final representative trajectory for the example case is $n_1n_2n_4n_7n_8$. The algorithm is summarized in Figure 8.

4.4 Construction of the C-tree

In Section 4.2, we have discussed the search and update operations in the C-tree. We now proceed to introduce how to insert a new entry into the C-tree, which occurs when a new cluster is created. Recall that each entry in the node of the C-tree has two fields: (i) a set of road IDs and (ii) a pointer. The maximum number of entries in each node is the same. All insertions start at a leaf node which is identified during the process of finding candidate clusters. We insert the new entry into that node (denoted as N) with the following steps:

1. If the node N contains fewer than the maximum legal number of entries, then there is room for the new entry. Insert the new entry in the node.

```
Select_Representative_Trajectory ($C$,$Trj_r$)
Input: $C$ is a cluster
Output: $Trj_r$ is the representative trajectory
1.  support($Trj_r$)← 0
2.  for each $Trj$ in $C$ do
3.       if support($Trj$) $>$support($Trj_r$) then
4.           $Trj_r \leftarrow Trj$
5.           support($Trj_r$) ← support($Trj$)
6.  $i \leftarrow 1$; $j \leftarrow length(Trj_r)$-1
7.  continue ← 1
8.  while ($i < j$ and $continue$) do
9.      continue ← 0
10.     if original($r_i$) $<$support($Trj_r$)-original($r_i$) then
11.         $i \leftarrow i + 1$; continue← 1
12.     if original($r_j$) $<$support($Trj_r$)-original($r_j$) then
13.         $j \leftarrow j - 1$; continue← 1
14. $Trj_r$ ←($r_i$...$r_j$)
15. return $Trj_r$
```

Fig. 8. Algorithm of Selecting Representative Trajectory

2. Otherwise N is full, and we evenly split it into two nodes. In particular, we randomly select an entry as seed. Then we compute Sim_c (Equation 3) between other entries and the seed. The average of all Sim_c serves as a separation value. Entries with Sim_c above the average are put in the node N, and the remaining entries are put in the new right node N'.
3. Next, we update the entry pointing to N. The road ID set in the parent is updated to include all roads occur in N. The update may be propagated to the upper levels of the tree. Moreover, if there is a split in the previous step, we need to insert a new entry which includes road IDs in the new node N' to the parent level. This may cause the tree to be split, and so on. If current node has no parent (i.e., the node is the root), a new root will be created above this one.

5 Experimental Study

All the experiments were run on a PC with 2.6G Pentium IV CPU and 3GB RAM. We use both synthetic and real road networks to generate moving objects. In the synthetic datasets, objects are moving on a randomly generated road map which has about 700 roads. Objects can have different speeds which are controlled by the parameter "average trajectory length". As for the real datasets, we use the generator by Brinkhoff [6]. Objects are moving on real road networks. A road consists of multiple segments and each segment is a straight line. An object is initially placed on a randomly selected road segment and then moves along this segment in a randomly selected direction. When the object reaches the end of the segment, an update is issued and a connected segment is selected. Object speeds are varied within a given speed range.

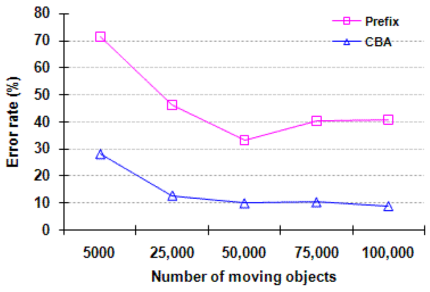
We compare our Clustering-Based Anonymization (CBA) algorithm with the latest work (denoted as Prefix [14]) by Pensa et al. In particular, we examine the existence of the inference route, the error rate, standard deviation and the running time. Unless noted otherwise we use the dataset containing 50,000 moving objects and set k to 30.

5.1 Experimental Results in Synthetic Datasets

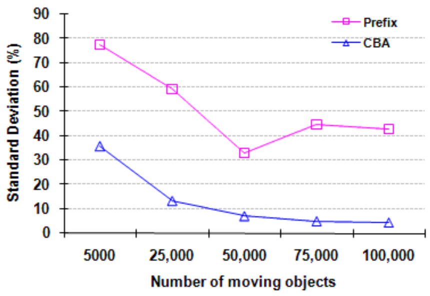
Effect of Data Sizes. In the first set of experiments, we study the effect of data sizes by varying the number of moving objects (i.e. number of trajectories) from 5K to 100K. Figure 9(a) shows the average error rate of the anonymization results obtained from Prefix algorithm and our CBA algorithm. We can observe that the CBA algorithm yields much less error rate than the Prefix algorithm in all cases. When the dataset is small (e.g. 5K), the anonymization results obtained from both algorithms have relatively high error rates. This is because the number of objects on each road is few and even a small change of an object trajectory by the anonymization process will have a big impact on the error rate. With the increase of the data sizes, the error rate caused by the CBA algorithm keeps decreasing and it is more than 5 times less compared to that of the Prefix algorithm for 100K dataset. The reason of such behavior is that CBA effectively groups similar trajectories and carefully selects representative trajectories which minimize the overall error rate. Figure 9(b) shows the standard deviation, where we can see that our standard deviation is much lower than that obtained from the Prefix algorithm. This confirms that our anonymization result on each road has similarly good quality.

Figure 10(a) shows the number of nodes having the inference route problem. It is not surprising to see that the anonymization result produced by our CBA algorithm contains 0 inference route. However, the anonymized result obtained from the Prefix algorithm has many road intersections (denoted as node) with the inference route problem caused by their definition of the trajectory support.

We also compare the running time of both approaches. As shown in Figure 10, our CBA algorithm is up to 5 times faster than the Prefix algorithm. This can be attributed to the C-tree that helps prune the clusters to be compared with each new trajectory and hence avoids many unnecessary calculation. The total time is inclusive of the construction and update cost of the C-tree which is almost neglectable compared to the benefits brought by the C-tree.

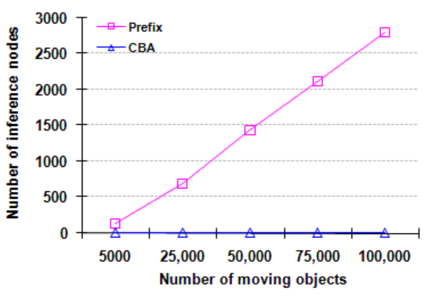


(a) Error rate

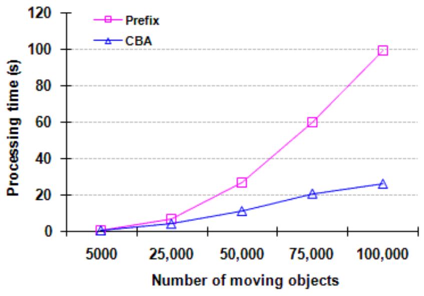


(b) Standard deviation

Fig. 9. Quality of the Anonymized Results



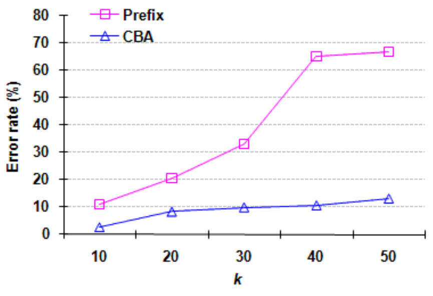
(a) Inference route problem



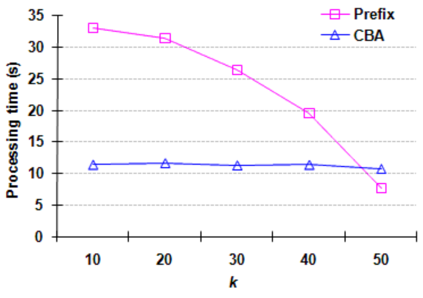
(b) Processing time

Fig. 10. Effect of Data Size

Effect of Parameter *k*. This set of experiments aims to evaluate the performance of both algorithms regarding different values of k. As shown in Figure 11(a), the error rate increases drastically with k by using the Prefix algorithm, while k has only minor effect on our CBA approach. Such behavior can be explained as follows. The Prefix algorithm removes all infrequent trajectories and add their supports to most similar frequent trajectories. When k is large, there are more infrequent trajectories, which thus causes more error. The standard deviation has also demonstrated the similar pattern as the error rate, and the Prefix algorithm again suffers from the inference route problem. Due to the space limit, we do not include the figures here. Regarding processing time (in Figure 11(b)), our CBA approach has a consistent performance while the Prefix approach requires less time for larger k. This is because the Prefix approach needs to deal with less frequent trajectories for a larger k. Note that this results in higher error rates.



(a) Error rate



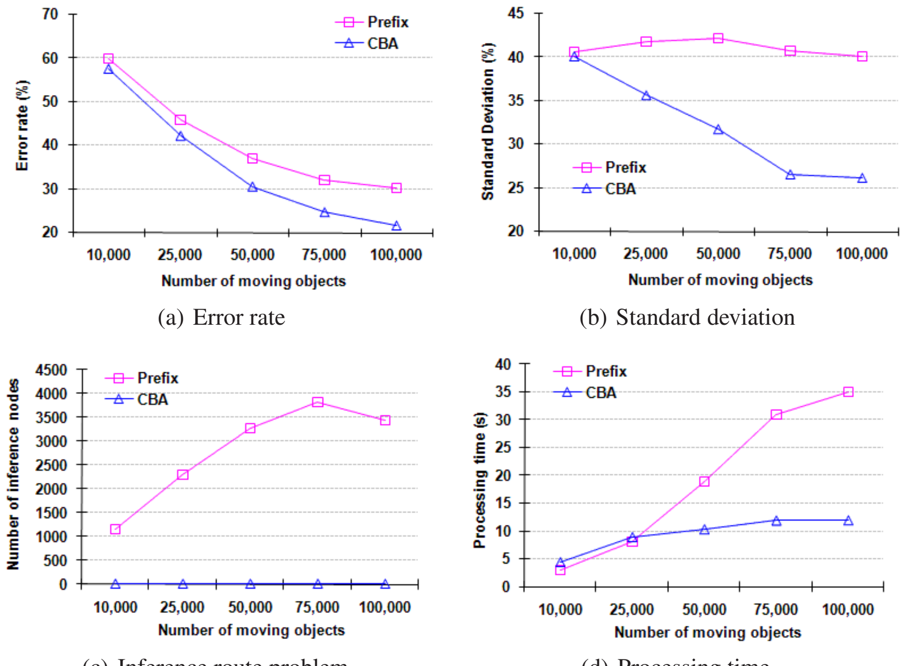
(b) Processing time

Fig. 11. Varying Parameter k

Effect of the Average Trajectory Length. We also investigated the effect of trajectory lengths by testing it up to 50 roads per trajectory. Within the same time interval, a longer trajectory indicates that the object has a faster speed. Our CBA algorithm outperforms the Prefix algorithm in all cases. Please refer to our technical report [**?**] for figures.

5.2 Experimental Results in Real Datasets

In this set of experiments, the datasets are generated based on the road map of Phelps County (Missouri, USA) using the generator [6]. The value of k is 10. From Figure 12, we can observe similar performance patterns as that using synthetic datasets.



(a) Error rate (b) Standard deviation

(c) Inference route problem (d) Processing time

Fig. 12. Performance in Real Road-network

6 Conclusion

Privacy preserving location data publishing has received increasing interest nowadays. In this paper, we address this newly emerging problem by taking into account an important factor, the road network constraint, which has been overlooked by many existing works. We identified and defined a new privacy problem (i.e. the inference route problem), and proposed an efficient and effective clustering-based anonymization algorithm. The clustering-based algorithm guarantees strict k-anonymity of the published dataset and avoids the inference route problem. We compared our approach with the most recent work and the experimental results demonstrate the superiority of our approach.

Acknowledgement

We thank authors of "Pattern-Preserving k-Anonymization of Sequences and its Application to Mobility Data Mining" who share their source code with us. This work was supported in part by the National Science Foundation under the contract IIS-0324835 and UM Research Board under the project "Preserving Location Privacy in Pervasive Environments".

References

1. Abul, O., Atzori, M., Bonchi, F., Giannotti, F.: Hiding sensitive trajectory patterns. In: Proc. of ICDM Workshop, pp. 693–698 (2007)
2. Abul, O., Bonchi, F., Nanni, M.: Never walk alone: Uncertainty for anonymity in moving objects databases. In: Proc. of the International Conference on Data Engineering, pp. 376–385 (2008)
3. Agrawal, R., Srikant, R.: Privacy-preserving data mining. In: Proc. of the ACM SIGMOD International Conference on Management of Data, pp. 439–450 (2000)
4. Atzori, M., Bonchi, F., Giannotti, F., Pedreschi, D.: Anonymity preserving pattern discovery. The VLDB Journal 17(4), 703–727 (2008)
5. Bettini, C., Wang, X.S., Jajodia, S.: Protecting Privacy Against Location-Based Personal Identification. In: Jonker, W., Petković, M. (eds.) SDM 2005. LNCS, vol. 3674, pp. 185–199. Springer, Heidelberg (2005)
6. Brinkhoff, T.: A framework for generating network-based moving objects (2004), `http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator`
7. Gidofalvi, G., Huang, X., Pedersen, T.B.: Privacy-preserving data mining on moving object trajectories. In: Proc. of the International Conference on Data Engineering, pp. 60–68 (2007)
8. Hoh, B., Gruteser, M.: Protecting location privacy through path confusion. In: Proc. of SecureComm., pp. 194–205 (2005)
9. Hoh, B., Gruteser, M., Xiong, H., Alrabady, A.: Preserving privacy in gps traces via uncertainty-aware path cloaking. In: Proc. of the ACM conference on Computer and Communications Security, pp. 161–171 (2007)
10. Lin, D., Gurung, S., Jiang, W., Hurson, A.: Privacy-preserving location publishing under road-network constraints. Technical Report, `http://web.mst.edu/~lindan/others/trajectory.pdf`
11. Mokbel, M.F.: Privacy in location-based services: State-of-the-art and research directions. In: Proc. of the International Conference on Mobile Data Management, p. 228 (2007)
12. Nergiz, M.E., Atzori, M., Saygin, Y., Guc, B.: Towards trajectory anonymization: a generalization-based approach. Transactions on Data Privacy 2(1), 47–75 (2009)
13. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Transactions on Knowledge & Data Engineering 16(1), 1424–1440 (2004)
14. Pensa, R.G., Monreale, A., Pinelli, F., Pedreschi, D.: Pattern-preserving k-anonymization of sequences and its application to mobility data mining. In: Proc. of the International Workshop on Privacy in Location-Based Applications (2008)
15. Sweeney, L.: Achieving k-anonymity privacy protection using generalization and suppression. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems 10(5), 571–588 (2002)
16. Tanner, J.C.: In search of lbs accountability. In: Telecom Asia (2008)
17. Terrovitis, M., Mamoulis, N.: Privacy preservation in the publication of trajectories. In: Proc. of the International Conference on Mobile Data Management, pp. 65–72 (2008)
18. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. Journal of the ACM 21(1), 168–173 (1974)
19. Yarovoy, R., Bonchi, F., Lakshmanan, L.V.S., Wang, W.H.: Anonymizing moving objects: how to hide a mob in a crowd? In: Proc. of the International Conference on Extending Database Technology, pp. 72–83 (2009)
20. Zaki, M.J.: Spade: An efficient algorithm for mining frequent sequences. Machine Learning 42(1/2), 31–60 (2001)

Incremental Clustering for Trajectories*

Zhenhui Li[1], Jae-Gil Lee[2], Xiaolei Li[3], and Jiawei Han[1]

[1] Univ. of Illinois at Urbana-Champaign
{zli28,hanj}@illinois.edu
[2] IBM Almaden Research Center
leegj@us.ibm.com
[3] Microsoft
xiaoleil@microsoft.com

Abstract. Trajectory clustering has played a crucial role in data analysis since it reveals underlying trends of moving objects. Due to their sequential nature, trajectory data are often received *incrementally*, e.g., continuous new points reported by GPS system. However, since existing trajectory clustering algorithms are developed for static datasets, they are not suitable for incremental clustering with the following two requirements. First, clustering should be processed efficiently since it can be frequently requested. Second, huge amounts of trajectory data must be accommodated, as they will accumulate constantly.

An *incremental clustering framework for trajectories* is proposed in this paper. It contains two parts: online micro-cluster maintenance and offline macro-cluster creation. For online part, when a new bunch of trajectories arrives, each trajectory is simplified into a set of directed line segments in order to find clusters of trajectory subparts. Micro-clusters are used to store compact summaries of similar trajectory line segments, which take much smaller space than raw trajectories. When new data are added, micro-clusters are updated incrementally to reflect the changes. For offline part, when a user requests to see current clustering result, macro-clustering is performed on the set of micro-clusters rather than on all trajectories over the whole time span. Since the number of micro-clusters is smaller than that of original trajectories, macro-clusters are generated efficiently to show clustering result of trajectories. Experimental results on both synthetic and real data sets show that our framework achieves high efficiency as well as high clustering quality.

1 Introduction

In recent years, the collection of trajectory data has become increasingly common. GPS chips implanted in animals have enabled scientists to track their study objects as they travel. RFID technology installed in vehicles has enabled traffic officers to track road traffic in real-time. With such data, trajectory clustering is a very useful task. It discovers movement patterns that help analysts see overall trends in the trajectories. For example,

* The work was supported in part by the U.S. National Science Foundation grants IIS-08-42769 and IIS-09-05215, and a grant from the Boeing company. Any opinions, findings, and conclusions expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

analysis of bird feeding and nesting habits is an important task. With the help of GPS, scientists can tag and track birds as they fly around. Such tracking devices report the trajectories of animals on a continual basis (*e.g.*, every minute, every hour). With such data, scientists can study the movement habits (*i.e.*, trajectory clusters) of birds.

One important property with tracking application is the *incremental* nature of the data. The data will grow to be in huge size as time goes by. Consider the following real case of moving vehicle data which is used in experiment evaluation.

Example 1. A taxi tracking system tracks the real-time locations of more than 5,000 taxis in San Francisco. With the sensor installed on each taxi, the system is able to receive information about current location(longitude and latitude) of each taxi with a precise timestamp. The system accumulates the updated data every minute. After a single day, the system will collect totally 7.2 million points with 1,440 points for each taxi. After a week, the number of points will be accumulated to 50.4 million points.

For static data sets, there are many existing trajectory clustering algorithms developed. However, to the best of our knowledge, none of them targeted at solving clustering problem for incremental huge trajectory data as pointed out in Example 1. Facing continuous data, previous methods will take long time to retrieve all the data and re-compute the trajectory cluster over the whole huge data set. If the users want to track real-time clusters every hour, it is almost impossible to finish computation within the time period threshold, especially considering the data size still keeps growing every minute. Therefore, trajectory data must be accommodated incrementally.

An important point to notice is that *new data will only affect local shifts*. It will not have big influence on clusters in the areas which are far away from the local area of new data. So, a more sensible approach to accommodate huge amount of data is to maintain and adjust *micro-clusters* of the trajectory data. Micro-clusters are tight clusters over small local regions. Due to their small sizes, they are more flexible to changes in the data source. Yet they still achieve the desired space savings of clusters by summarizing extremely similar input trajectories. These properties make them suitable for incremental clustering.

This work proposes an *incremental Trajectory Clustering using Micro- and Macro-clustering* framework called TCMM. It makes the following contributions towards an incremental trajectory clustering solution. First, trajectories are simplified by partitioning into line segments to find the clusters of sub-trajectories. Second, micro-clusters of the partitioned trajectories are computed and maintained incrementally. Micro-clusters hold and summarize similar trajectory partitions at very fine granularity levels. They use very little space and can be updated efficiently. And finally, micro-clusters are used to generate the macro-clusters(*i.e.*, final trajectory clusters).

The TCMM framework is truly incremental in the sense that micro-clusters are incrementally maintained as more and more data are received. Because their granularity level is low, they can adjust to all types of change in the input data. The number of micro-clusters is much smaller than that of the original input data. When the user wants to compute the full trajectory clusters, micro-clusters are combined together to form the macro-clusters in higher granularity level.

The rest of this paper is organized as follows. Section 2 formally defines the problem and gives an outline of the TCMM framework. Sections 3.1 and 3.2 discuss the micro-clusters and the macro-clusters, respectively. Experiments are shown in Section 4. Related work is analyzed in Section 5. Finally, the paper concludes in Section 6.

2 General Framework

2.1 Problem Statement

The data to be studied in this work will be in the context of an *incremental data source*. That is, new batches of trajectory data will continuously be fed into the clustering algorithm (*e.g.*, from new data recordings). The goal is to process such data and produce clusters *incrementally* and *not* have to re-compute from scratch every time.

Let the input data be represented by a sequence of time-stamped trajectory data sets: $\langle I_{t_1}, I_{t_2}, \ldots \rangle$ where each I_{t_i} is a set of trajectories being presented at time t_i. Each $I_{t_i} = \{TR_1, TR_2, \ldots, TR_{n_{TR}}\}$ where each TR_j is a trajectory. A single trajectory TR_j is often represented as a polyline, which is a sequence of connected line segments. It can be denoted as $TR_j = p_1 p_2 \ldots p_{len_j}$, where each point p_i is a time-stamped point. TR_j can be further simplified to derive a new polyline with fewer points while its deviation from the original polyline is below some threshold. The simplification techniques have been studied extensively in previous work [11,5] . In this paper, we use the simplification technique in our previous paper [11]. Simplified trajectory is represented as $TR_j^{simplified} = L_1 L_2 \ldots L_n$, where L_i and L_{i+1} are connected directed line segments (i.e., trajectory partitions).

Given such input data, the goal is to produce a set of clusters $O = \{C_1, C_2, \ldots, C_{n_C}\}$. A *cluster* is a set of directed trajectory line segments $C_i = \{L_1, L_2, \ldots, L_{ln}\}$, where L_k is a directed line segment from certain simplified trajectory $TR_j^{simplified}$ at certain time stamp t_i. Because we do clustering on line segments rather than whole trajectories, the clusters we find are actually sub-trajectory clusters, which are the popular paths visited by many moving objects.

2.2 TCMM Framework

Figure 1 shows the general data flow of TCMM. The x-axis represents the progress of time and the y-axis shows the progress of data processing. As the figure illustrates, input data are received continuously.

The first step is micro-clustering. Because there is an infinite data source, it is impossible to store all the preprocessed input data and compute clusters from them on request. To solve this problem, this work introduces the concept of *trajectory micro-clusters*. The term "micro" refers to the extreme tightness of the clusters. The idea is to only cluster at very fine granularity. Hence, the number of micro-clusters is much larger than that of final trajectory clusters. Figure 1 shows the micro-clusters in the second row. Section 3.1 will discuss them in detail.

The second step is macro-clustering, which will be discussed in detail in Section 3.2. Compared to the micro-clustering step, which are updated constantly as new data is received, the macro-clustering step is *only* evoked after receiving the user's request of trajectory clusters. This step will then use the micro-clusters as input.

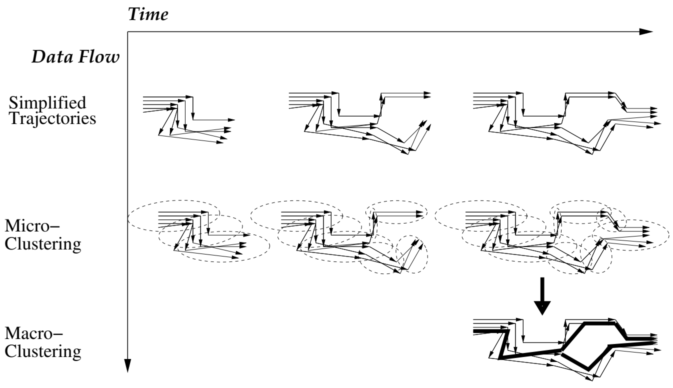


Fig. 1. The Framework

3 Trajectory Clustering Using Micro- and Macro-clustering

3.1 Trajectory Micro-Clustering

As newly arrived trajectories will only affect local clustering result, trajectory micro-clusters (or just micro-clusters) are introduced here to maintain a fine-granularity clustering. Micro-clusters (defined in Section 3.1) are much more restrictive than the final clusters in the sense that each micro-cluster is meant to only hold and summarize the information of local partitioned trajectories. Micro-clustering will enable more efficient computation of final clusters comparing with computation from original line segments.

Algorithm 1. Trajectory Micro-Clustering

```
1: **Input**:New trajectories $I_{t_{current}} = \{TR_1, TR_2, \cdots, TR_{nTR}\}$ and existing micro-clusters
   $MC = \{MC_1, MC_2, \ldots, MC_{n_{MC}}\}$.
2: **Parameter**: $d_{max}$
3: **Output**: Updated $MC$ with new trajectories inserted.
4: **Algorithm**:
5: **for** every $TR_i \in I_{t_{current}}$ **do**
6:    **for** every $L_j \in TR_i$ **do**
7:       Find the closest $MC_k$ to line segment $L_j$ /* Section 3.1 */
8:       **if** *distance*$(L_j, MC_k) \leq d_{max}$ **then**
9:          Add $L_j$ into $MC_k$ and update $MC_k$ accordingly
10:      **else**
11:         Create a new micro-cluster $MC_{new}$ for $L_j$;
12:         **if** size of $MC$ exceeds memory constraint **then**
13:            Merge micro-clusters in $MC$ /* Section 3.1 */
```

Algorithm 1 shows the general work flow of generating and maintaining micro-clusters. It proceeds as follows. After a batch of new trajectories arrive, we compute the closest micro-cluster MC_k for each line segment L_i in every trajectory. If the distance between L_i and MC_k is less than a distance threshold (d_{max}), L_i will be inserted into

MC_k. Otherwise, a new micro-cluster MC_{new} will be created for L_i. If the creation of the new micro-cluster results in the overload of the total number of micro-clusters, some micro-clusters will be merged. The rest of this section discuss these steps in detail.

Micro-Cluster Definitions. Each trajectory micro-cluster will hold and summarize a set of partitioned trajectories, which are essentially line segments.

Definition 1 (Micro-Cluster). *A* trajectory micro-cluster (or micro-cluster) *for a set of directed line segments $L_1, L_2, \cdots, L_N$ is defined as the tuple: (N, LS_{center}, LS_{θ}, LS_{length}, SS_{center}, SS_{θ}, SS_{length}), where N is the number of line segments in the micro-cluster, LS_{center}, LS_{θ}, and LS_{length} are the linear sums of the line segments' center points, angles and lengths respectively, SS_{center}, SS_{θ}, and SS_{length} are the squared sums of the line segments' center points, angles and lengths respectively.*

The definition of trajectory micro-cluster is an extension of the cluster feature vector in *BIRCH* [14]. The linear sum LS represents the basic summarized information of line segments(*i.e.*, center point, angle and length). The square sum SS will be used to calculate the tightness of micro-cluster which will be discussed in Section 3.1. The additive nature of the definition makes it easy to add new line segments into the micro-cluster and merge two micro-clusters. Meanwhile, the definition is designed to be consistent with the distance measure of line segments in Section 3.1.

Also, every trajectory micro-cluster will have a *representative line segment*. As the name suggests, this line segment is the representative line segment of the cluster. It is an "average" of sorts.

Definition 2 (Representative Line Segment). *The* representative line segment *of a micro-cluster is represented by the starting point s and ending point e. s and e can be computed from the micro-cluster features.*

$$s = (center_x - \frac{\cos\theta}{2} len, center_y - \frac{\sin\theta}{2} len)$$
$$e = (center_x + \frac{\cos\theta}{2} len, center_y + \frac{\sin\theta}{2} len)$$

where $center_x = LS_{center_x}/N$, $center_y = LS_{center_y}/N$, $len = LS_{length}/N$, and $\theta = LS_{\theta}/N$.

Figure 2 shows an example. There are four line segments in the micro-cluster, which are drawn in thin lines. The representative line segment of the micro-cluster is drawn in a thick line.

Creating and Updating Micro-Clusters. When a new line segment L_i is received, the first task is to find the closest micro-cluster MC_k that can absorb L_i (*i.e.*, Line 7 in Algorithm 1). If the distance between L_i and MC_k is less than the distance threshold d_{max}, L_i is then added to MC_k and MC_k is updated accordingly; if not, a new micro-cluster is created (*i.e.*, Line 8 to 11 in Algorithm 1). This section will discuss how these steps are performed in detail.

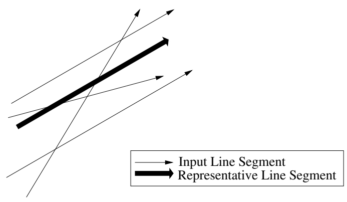


Fig. 2. Representative Line Segment

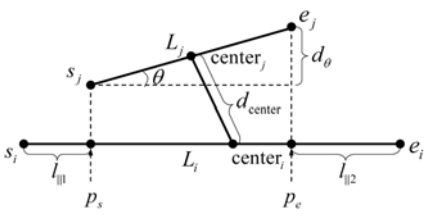


Fig. 3. Line Segments Distance

Before proceeding, the distance between a line segment and a micro-cluster is defined. Since a micro-cluster has its representative line segment, the distance is in fact defined between two line segments, which is composed of three components: the center point distance (d_{center}), the angle distance (d_θ) and the parallel distance ($d_{\|}$) . The distance is adapted from a similarity measure used in the area of pattern recognition [10], which is a modified line segment Hausdorff distance. The similar distance measure is also used in [11]. Different from [11], we use component d_{center} instead of $d_\perp$. The reason to choose d_{center} is because it is a more balanced measure between d_θ and $d_{\|}$ and it is easier to adapt the concept of extent, which will be introduced in Section 3.1.

Let s_i and e_i be the starting and ending points of L_i; similarly for s_j and e_j with L_j. Without loss of generality, the longer line segment is assigned to L_i, and the shorter one to L_j. Figure 3 gives an intuitive illustration of the distance function.

Definition 3. *The distance function is defined as the sum of three components:*

$$dist(L_i, L_j) = d_{center}(L_i, L_j) + d_\theta(L_i, L_j) + d_{\|}(L_i, L_j)$$

The center distance:

$$d_{center}(L_i, L_j) = \| \ center_i - center_j \ \| ,$$

where $\| \ center_i - center_j \ \|$ *is the Euclidean distance between center points of* L_i *and* L_j*.*
The angle distance:

$$d_\theta(L_i, L_j) = \begin{cases} \| \ L_j \ \| \times \sin(\theta), & 0^o \leq \theta < 90^o \\ \| \ L_j \ \|, & 90^o \leq \theta \leq 180^o \end{cases},$$

where $\| \ L_j \ \|$ *denote length of* L_j*,* $\theta(0^o \leq \theta \leq 180^o)$ *denote the smaller intersecting angle between* L_i *and* L_j*. Note that the range of* θ *is not* $[0^o, 360^o)$ *because* θ *is the value of smaller intersecting angle without considering the direction.*
The parallel distance:

$$d_{\|}(L_i, L_j) = \min(l_{\|1}, l_{\|2}),$$

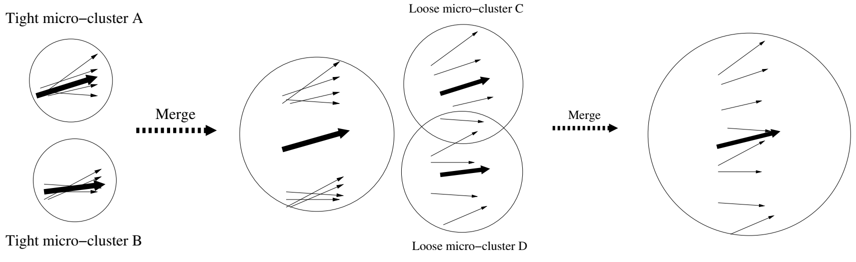
where $l_{\|1}$ *is the Euclidean distances of* p_s *to* s_i *and* $l_{\|2}$ *is that of* p_e *to* e_i*.* p_s *and* p_e *are the projection points of the points* s_j *and* e_j *onto* L_i *respectively.*

After finding the closest micro-cluster MC_k, if the distance from L_i is less than d_{max}, L_i is inserted into it, and the linear and square sums in MC_k are updated accordingly.

Because they are just sums, the additivity property applies and the update is efficient. If the distance between the nearest micro-cluster and L_i is bigger than d_{max}, a new micro-cluster will be created for L_i. The initial measures in the new micro-cluster is simply derived from line segment L_i ($i.e.$, center point, theta, and length).

Merging Micro-Clusters. In real world applications, storage space is always a constraint. The TCMM framework faces this problem with its micro-clusters as shown in Line 12 to 13 of Algorithm 1. If the total space used by micro-clusters exceeds a given space constraint, some micro-clusters have to be merged to satisfy the space constraint. Meanwhile, if the number of micro-clusters keeps increasing, it will affect the efficiency of algorithm because the most time-consuming part is finding the nearest micro-cluster. And what is most important, it may be unnecessary to keep all the micro-clusters since some of the micro-clusters may become closer after several rounds of updates. Therefore, the algorithm demands merging close micro-clusters when necessary to speed up efficiency and save storage. Obviously, pairs of micro-clusters that contain similar line segments are better candidates for merging because the merge results in less information loss.

One way to compute the similarity between two micro-clusters is to calculate the distance between the representative line segments of the micro-clusters. Though intuitive, this method fails to consider the tightness of the micro-clusters. Figure 4 shows an example that how tightness might effect distance between two micro-clusters. Figure 4(a) shows two tight micro-clusters and the micro-cluster after merging them. Figure 4(b) shows the case for two comparatively loose micro-clusters. We can see that micro-cluster A and micro-cluster C have same representative line segments, and so do micro-clusters B and D. Thus the distance between micro-cluster A and B should be the same as that between micro-clusters C and D if we measure the distance only using representative line segments. In this case, the chance to merge micro-clusters A and B is equal to that of merging micro-clusters C and D. However, we actually prefer merging micro-clusters C and D. There are two reasons: on one hand, if both micro-clusters are very tight, they may not be good candidates for merging because it would break that tightness after the merge. On the other hand, if they are both loose, it may not do much harm to merge them even if their representative line segments are somewhat far apart.



(a) Merging tight micro-clusters (b) Merging loose micro-clusters

Fig. 4. Merging micro-clusters

Hence, a better approach would be to consider the *extent* of the micro-clusters and use that information in computing the distance between micro-cluster.

In the following parts, we will first introduce the way to compute micro-cluster extent, then give definitions of the distance between micro-clusters with extent information. Lastly, we will discuss how to merge two micro-clusters.

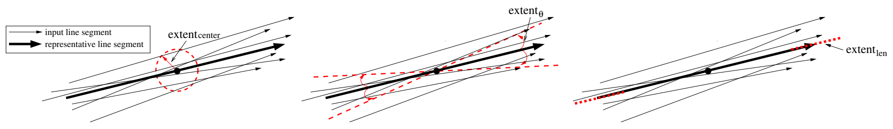
Micro-Cluster Extent. The extent of a micro-cluster is an indication of its tightness. Recall that micro-clusters are represented by tuples of the form: (N, LS_{center}, LS_{θ}, LS_{length}, SS_{center}, SS_{θ}, SS_{length}), which maintain linear and square sums of center, angle and length. The extent of the micro-cluster also includes three part $extent_{center}$, $extent_{\theta}$ and $extent_{length}$ to measure the tightness of three basic facts of a trajectory micro-cluster. The extents are the standard deviation that calculated from its corresponding LS and SS. We have the following lemma from [14].

Lemma 1. *Given a set of distance values, $D = (d_1, d_2, ..., d_n)$. Let $LS = \sum_{i=1..n} d_i$, and $SS = \sum_{i=1..n} (d_i)^2$. The standard deviation of the distances is $\sigma = \sqrt{\frac{n \times SS - (LS)^2}{n^2}}$.*

Using Lemma 1, we give a formal definition for extent of a micro-cluster:

$$extent_{\alpha} = \sqrt{(N \times SS_{\alpha} - LS_{\alpha}^2)/N^2}$$

where symbol α represents $center$, θ, or $length$ and N is the number of line segments in the micro-cluster.

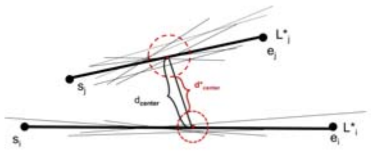


(a) Center extent (b) θ extent (c) Length extent

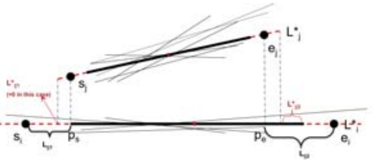
Fig. 5. Micro-Cluster Extent

To give an intuition of extent concept, Figure 5 shows an example of $extent_{center}$, $extent_{\theta}$ and $extent_{length}$. Figure 5(a) states that "most" center points of the line segments stored in this micro-cluster are within the circle of radius $extent_{center}$. Figure 5(b) illustrates that "most" angles vary within a range of $extent_{\theta}$ and Figure 5(c) reflects the uncertainty of length.

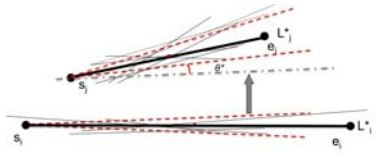
Micro-Cluster Distance with Extent. With the extents properly defined, we can now incorporate them into the distance function. Recall that the intention of extent was to adjust the distance function based on the tightness of micro-clusters. For instance, let $d_{1,2}$ be the distance between micro-clusters MC_1 and MC_2 according to the distance function defined previously. If these two micro-clusters are both "tight" (*i.e.*, having zero or very small extent), then $d_{1,2}$ indeed represents the distance between them. However, if these two micro-clusters are both "loose" (*i.e.*, having large extent), then their "true" inter-cluster distance should actually be *less* than $d_{1,2}$. This is because the line segments at the borders of the two micro-clusters are likely to be much closer than $d_{1,2}$. With respect to merging micro-clusters, this allows loose micro-clusters to be more easily merged and vice-versa. The adjustment of the distance function using extent is relatively simple. Whenever possible, extent is used to reduce the distance between the representative line segments of micro-clusters.



(a) Center distance with extent



(b) Parallel distance with extent



(c) Angle distance with extent

Fig. 6. Line Segments Distance with Extent

To measure the distance between micro-cluster i and micro-cluster j, it is equivalent to measure the distance $d^*(L_i^*, L_j^*)$ between the representative line segments L_i^* with $extent^i$ and L_j^* with $extent^j$. Figure 6 shows an intuitive example of distance measure with extent. For example, in Figure 6(a), the distance between the centers is the distance between representative line segments minus the center extents of two micro-clusters. The formal definition is given as follows based on the modification of distance measure between line segments (*i.e.*, Definition 3). To avoid the redundancy in presentation, the symbols explained in Definition 3 are not repeated in Definition 4.

Definition 4. *The distance between L_i^* and L_j^* contains three parts: center distance d_{center}^*, angle distance d_θ^* and parallel distance $d_\parallel^*$.*

$$dist(L_i^*, L_j^*) = d_{center}(L_i^*, L_j^*) + d_\theta(L_i^*, L_j^*) + d_\parallel(L_i^*, L_j^*)$$

The center distance:

$$d_{center}^*(L_i^*, L_j^*) = \max\left(0, \|center_i - center_j\| - extent_{center}^i - extent_{center}^j\right)$$

The angle distance:

$$\theta^* = \theta - (extent_\theta^i + extent_\theta^j)$$

$$d_\theta^*(L_i^*, L_j^*) = \begin{cases} \| L_j^* \| \times \sin(\theta^*), & 0^o \leq \theta^* < 90^o \\ \| L_j^* \|, & 90^o \leq \theta^* \leq 180^o \end{cases}$$

The parallel distance:

$$d_{\|}^{*}(L_i^*, L_j^*) = \max\Big(0, \min(l_{\|1}, l_{\|2}) - (extent_{length}^{i} + \overline{extent_{length}^{j}})/2\Big),$$

where $\overline{extent_{length}^{j}}$ *is the projection of* $extent_{length}^{j}$ *onto* L_i^*.

Note that the distances defined between two representative line segments with extent are smaller than those defined between two original ones. And the distance may be equal to zero when there is an overlap between representative line segments with extent.

Merging Algorithm. The final algorithm of merging micro-clusters is as follows. Given M micro-clusters, the distance between any two micro-clusters is calculated. They are then sorted from the most similar to the least similar. The most similar pairs are the best candidate for merging since merging them result in the least amount of information loss. They are merged until the number of micro-clusters satisfy the given space constraints.

3.2 Trajectory Macro-clustering

The last step in the TCMM framework produces the overall trajectory clusters. While micro-clustering is processed with a new batch of data comes in, macro-clustering is evoked *only when* it is called upon by the user.

Since the distance between micro-clusters is defined in Definition 4, it is easy to adapt any clustering method on spatial points. We simply need to replace the distancce between spatial points with the distance between micro-clusters. In our framework, we use density-based clustering [7], which is also used in TRACLUS [11]. The clustering technique in macro-clustering step is the same as the clustering algorithm in TRACLUS. The only difference is that macro-clustering in TCMM is performed on the set of micro-clusters rather than the set of trajectory partitions as in TRACLUS. The micro-clusters are clustered through a density-based algorithm which discovers maximally "density-connected" components, each of which forms a macro-cluster.

4 Experiments

This section tests the efficiency and effectiveness of the proposed framework under a variety of conditions with different datasets. The TCMM framework and the TRA-CLUS [11] framework are both implemented using C++ and compiled with gcc. All tests were performed on a Intel 2.4GHz PC with 2GB of RAM.

4.1 Synthetic Data

As a simple way to quickly test the "accuracy" of TCMM, synthetic trajectory data is generated. Objects are generated to move along pre-determined paths with small perturbations ($< 10\%$ relative distance from pre-determined points). 15% trajectories are random noises added to the data. Figure 7 shows the result of incremental micro-clustering at two different snapshots. Figure 7(a) shows raw trajectories in gray; one can clearly see the trajectory clusters. The extracted micro-clusters are drawn with red/bold lines; they match the intuitive clusters. Figure 7(b) shows the trajectories and extraction results for a later snapshot. Again, they match the intuitive clusters.

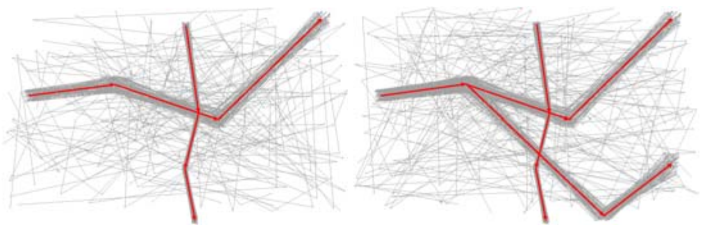

(a) Micro-clusters at snapshot 1 (b) Micro-clusters at snapshot 2

Fig. 7. Micro-clusters from synthetic data

4.2 Real Animal Data in Free Space

Next, clusters are computed from deer movement data [1] in Year 1995. This data set contains 32 trajectories with about $20,000$ points in total. The dataset size of animal is considerably small due to the high expense and technological difficulties to track animals. But it is worth studying animal data because the trajectories are in free space rather than on restricted road network. In Section 4.3, a further evaluation on a much larger vehicle dataset containing over $7,000$ trajectories will be conducted.

To the best of our knowledge, there is no any other incremental trajectory clustering algorithm. So the results of TCMM will be compared with TRACLUS [11], which does trajectory clustering over the whole data set. Since micro-clusters in TCMM summarize original line segments information with some information loss, the clustering result on micro-clusters might not be as real as TRACLUS. So the cluster result from TRACLUS is used as a standard to test the accuracy of TCMM. Meanwhile, it is important to show the efficiency against TRACLUS while both results are similar.

We adapt performance measure, sum of square distance (SSQ), from CluStream [1] to test the quality of clustering results. Assume that there are a total of n line segments at the current timestamp. For each line segment L_i, we find the centroid (*i.e.*, representative line segment) C_{L_i} of its closest macro-cluster, and compute $d(L_i, C_{L_i})$ between L_i and C_{L_i}. The SSQ at timestamp is equal to the sum of $d^2(L_i, C_{L_i})$ and the average SSQ is SSQ/n.

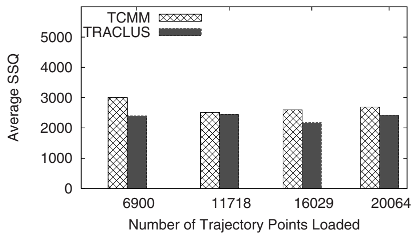


Fig. 8. Effectiveness Comparison (Deer)

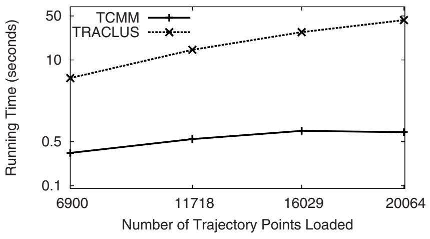


Fig. 9. Efficiency Comparison (Deer)

[1] http://www.fs.fed.us/pnw/starkey/data/tables/

As shown in Algorithm 1, there is only one parameter d_{max} in micro-clustering step and we set it to 10. The parameter sensitivity is analyzed and discussed in Section 4.4. For macro-clustering and TRACLUS, they use the same parameters ε and $MinLns$. Here, ε is set to 50 and $MinLns$ is set to 8.

Figure 8 shows the quality of clustering results. Comparing with TRACLUS, the average SSQ of TCMM is slightly higher. In the worst case, the average SSQ of TCMM is 2% higher than TRACLUS. But the processing time of TCMM is significantly faster than TRACLUS. To process all the $20,000$ points, TCMM only takes 0.7 seconds while TRACLUS takes 43 seconds. The reason is that it is much faster to do clustering over micro-clusters rather than over all the trajectory partitions. With the deer dataset, at last, the number of trajectory partitions (3390) is much more than the number of micro-clusters (324) in total.

4.3 Real Traffic Data in Road Network

Real world GPS recorded data from a taxi company in San Francisco is used to test the performance of TCMM. The data set is huge and keeps growing as time goes by. It contains 7,727 trajectories($100,000$ points) of taxis as they travel around the city picking up and dropping off passengers.

Figure 10 shows the visual clustering result of taxi data. First row and second row show the micro-clusters (d_{max} set to 800) and macro-clusters (ε set to 50 and $MinLns$ set to 8). Last row shows cluster result from TRACLUS. Time 0, 1, and 2 correspond to the timestamps respectively when 52317, 74896, and 98002 trajectory points have been loaded. As we can see from Figure 10, the results from TCMM and TRACLUS are similar except very few differences. The similar clustering performance is further proved in Figure 11, where the average SSQ of TCMM is only slightly higher than that of TRACLUS (2% higher in worst case and 1.4% higher on average).

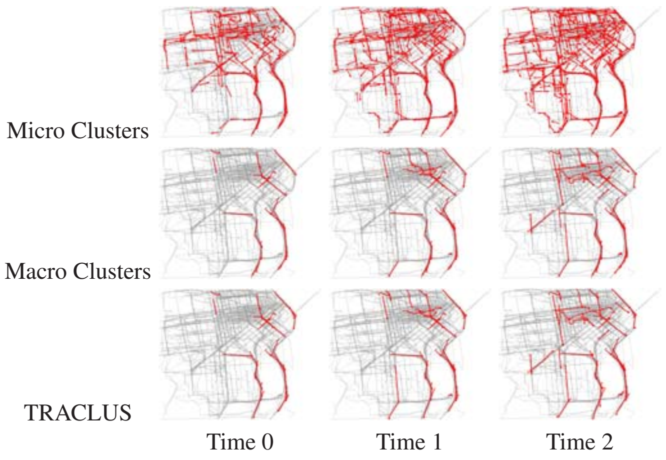


Fig. 10. Taxi Experiment

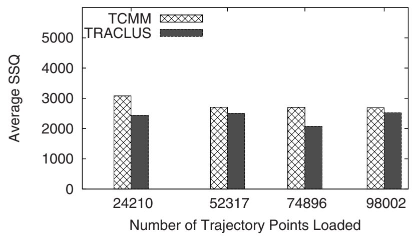


Fig. 11. Effectiveness Comparison(Taxi)

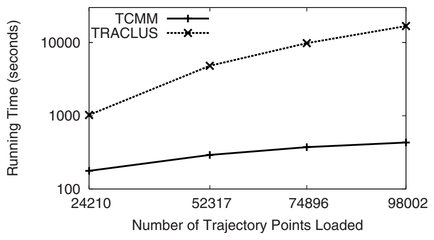


Fig. 12. Efficiency Comparison(Taxi)

Regarding to efficiency issue, Figure 12 shows the time needed to process the data in 4 increments with TCMM and TRACLUS. Compared to previous data sets, TRACLUS is substantially slower this time due to the larger data set size. To process all the data, TRACLUS takes about 4.6 hours while TCMM only takes about 7 minutes to finish. This is because the number of trajectory partitions (52,600) is much larger than the number of micro-clusters (2,013). It means that TCMM is much more efficient than TRACLUS as data set is getting bigger, while at the same time, the effectiveness remains the same as TRACLUS.

4.4 Parameter Sensitivity

The micro-clustering step of TCMM has the nice property that it only requires one parameter: d_{max}. A large d_{max} builds micro-clusters that are large in individual size but small in overall quantity, whereas a small d_{max} has the opposite effect. If we set $d_{max} = 0$, TCMM is actually TRACLUS because each line segment will form a micro-cluster itself. Then the macro-clustering applied on micro-clusters is exactly the one applied on original line segments. Therefore, the smaller the d_{max} is, the better the quality of clustering should be but the longer processing time is needed. At the same time, if we set d_{max} larger, the algorithm runs faster but loses more information in micro-clustering. Hence there is a trade-off between effectiveness and efficiency.

We use taxi datasets to study the parameter sensitivity of our algorithm. Figure 13 and Figure 14 show the performance of TCMM with different d_{max}. We can see that when $d_{max} = 600$, the average SSQ is closer to that of TRACLUS, which shows that it has more similar performance as TRACLUS. But it also takes longer time to do

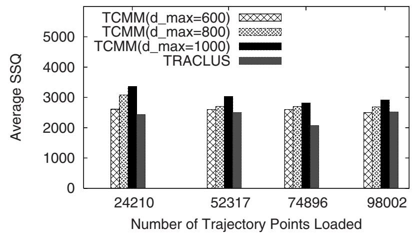


Fig. 13. Effectiveness with d_{max}

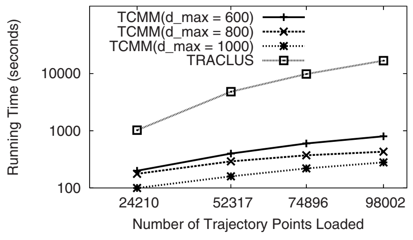


Fig. 14. Efficiency with d_{max}

clustering when $d_{max} = 600$. However, comparing with TRACLUS, the time spent on incremental clustering is still significantly shorter.

5 Related Work

Clustering has been studied extensively in machine learning and data mining. A number of approaches have been proposed to process *point* data in various conditions , such as k-means [12], BIRCH [14,3] and OPTICS [2]. The micro-clustering step in TCMM share the idea of micro-clustering in BIRCH [14]. However, BIRCH [14] cannot handle trajectory clustering. The clustering feature in TCMM has been extended to exactly describe a line-segment cluster by including three kinds of information. The data bubble [3] is an extension of the BIRCH framework and introduces the idea of the extent. TCMM also uses the extent in its micro-cluster, but the definition has been changed to accommodate trajectories.

Trajectory clustering has been studied in various contexts. Gaffney *et al.* [9,4,8] proposes several algorithms for model-based trajectory clustering. TRACLUS [11] is a trajectory clustering algorithm which performs density-based clustering over the entire set of sub-trajectories. However, all of these algorithms cannot efficiently handle *incremental* data. They are not suitable for incremental data since clusters are re-calculated from scratch every time.

CluStream [1] studies clustering dynamic data streams. Our method adapts its micro-/macro-clustering framework for trajectory data. However, our method so far handles only incremental data but not trajectory streams. This is because sub-trajectory micro-clustering has to wait for nontrivial number of new points accumulated to form sub-trajectories, which needs addition buffer space and waiting time. Moreover, the processing of sub-trajectories is more expensive and additional processing power is needed for real time stream processing. Thus, the extension of our framework for trajectory streaming left for future research.

Ester *et al.* [6] proposes the Incremental DBSCAN algorithm, which is an extension of DBSCAN for incremental data. Here, the final clusters are directly updated based on new data. We believe our two-step process is more flexible since any clustering algorithm can be employed for macro-clustering, whereas IncrementalDBSCAN is dedicated to DBSCAN. More recently, Sacharidis *et al.* [13] discusses the problem of online discovering hot motion. The basic idea is to delegate part of the path extraction process to objects, by assigning to them adaptive lightweight filters that dynamically suppress unnecessary location updates. Their problem is different from ours in two ways: first, they are trying to find recent hot paths whereas our clusters target at whole time span; and second, they require the objects in a moving cluster to be close enough to each other at any time instant during a sliding window of W time units but we are more from geometric point of view to measure the distance between trajectories.

6 Conclusions

In this work, we have proposed the TCMM framework for incremental clustering of trajectory data. It uses a two-step process to handle incremental datasets. The first step maintains a flexible set of micro-clusters that is updated continuously with the input data. Micro-clusters compress the infinite data source to a finite manageable size while still recording much of the trajectory information. The second step, which is on-demand, produces the final macro-clusters of the trajectories using the micro-clusters as input. Compared to previous static approaches, the TCMM framework is much more flexible since it does not require all of the input data at once. The micro-clusters provide a summary of the trajectory data that can be updated easily with any new information. This makes it more suitable for many real world application scenarios.

References

1. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: VLDB 2003 (2003)
2. Ankerst, M., Breunig, M., Kriegel, H.-P., Sander, J.: OPTICS: Ordering points to identify the clustering structure. In: SIGMOD 1999 (1999)
3. Breunig, M.M., Kriegel, H.-P., Kröger, P., Sander, J.: Data bubbles: Quality preserving performance boosting for hierarchical clustering. In: SIGMOD 2001 (2001)
4. Cadez, I.V., Gaffney, S., Smyth, P.: A general probabilistic framework for clustering individuals and objects. In: KDD 2000 (2000)
5. Douglas, D., Peucker, T.: Algorithms for the reduction of the number of points required to represent a line or its character. In: The Ameican Cartographer (1973)
6. Ester, M., Kriegel, H.P., Sander, J., Wimmer, M., Xu, X.: Incremental clustering for mining in data warehousing environment. In: VLDB 1998 (1998)
7. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases. In: KDD 1996 (1996)
8. Gaffney, S., Robertson, A., Smyth, P., Camargo, S., Ghil, M.: Probabilistic clustering of extratropical cyclones using regression mixture models. Technical Report UCI-ICS 06-02, University of California, Irvine (January 2006)
9. Gaffney, S., Smyth, P.: Trajectory clustering with mixtures of regression models. In: KDD 1999 (1999)
10. Chen, M.K.L.J., Gao, Y.: Noisy logo recognition using line segment hausdorff distance. Pattern Recognition (2002)
11. Lee, J.-G., Han, J., Whang, K.-Y.: Trajectory clustering: A partition-and-group framework. In: SIGMOD 2007 (2007)
12. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proc. 5th Berkeley Symp. Math. Statist., Prob., vol. 1, pp. 281–297 (1967)
13. Sacharidis, D., Patroumpas, K., Terrovitis, M., Kantere, V., Potamias, M., Mouratidis, K., Sellis, T.: On-line discovery of hot motion paths. In: EDBT 2008 (2008)
14. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: an efficient data clustering method for very large databases. In: SIGMOD 1996 (1996)

NNCluster: An Efficient Clustering Algorithm for Road Network Trajectories*

Gook-Pil Roh and Seung-won Hwang

Department of Computer Science & Engineering,
Pohang University of Science and Technology (POSTECH),
Pohang, Republic of Korea
{noh9pil,swhwang}@postech.ac.kr

Abstract. With the advent of ubiquitous computing, we can easily acquire the locations of moving objects. This paper studies clustering problems for trajectory data that is constrained by the road network. While many trajectory clustering algorithms have been proposed, they do not consider the spatial proximity of objects across the road network. For this kind of data, we propose a new distance measure that reflects the spatial proximity of vehicle trajectories on the road network, and an efficient clustering method that reduces the number of distance computations during the clustering process. Experimental results demonstrate that our proposed method correctly identifies clusters using real-life trajectory data yet reduces the distance computations by up to 80% against the baseline algorithm.

1 Introduction

Due to the evolution of positioning and sensor technologies such as GPS and RFID, we can now easily record the movements or trajectories of moving objects. Some examples of this include vehicle locations, object tracking data, and animal movement data.

Recently, a massive amount of collected trajectory information has been published and shared in websites or in web communities [1,2,3]. While these sites simply visualize the relevant trajectories, advanced data analysis techniques, such as clustering, can be used for recommending interesting travel sequences based on the common trajectories of users [28] or finding users with similar life experiences based on their trajectories [22].

This paper studies how to develop an efficient clustering algorithm for trajectory data. Clustering trajectories can be used to identify distinct groups in which trajectories have more similar moving patterns than those in other groups. Specifically, this paper focuses on clustering the vehicle trajectories of users on

* This work was supported by Microsoft Research Asia and Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology (MEST) / Korea Science and Engineering Foundation (KOSEF), grant number R11-2008-007-03003-0.

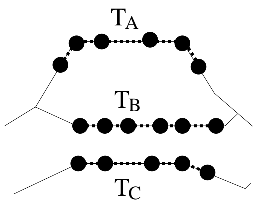


Fig. 1. Spatial proximity on road networks

road networks, which we will call *the network trajectory* from this point on. While there has been prior research work done on clustering trajectories, none of them considered the movement constraints imposed by the underlying road networks– Two locations that are close based on Euclidean distance may not be reachable over road networks, if no road exists between the two locations.

To achieve the goal, we first identify desirable properties that the distance measure should satisfy. For instance, the distance measure should reflect the *spatial proximity* from the viewpoint of the road network (will be referred to *road-network proximity*). To illustrate, Figure 1 presents three trajectories on road networks. When the distance measure used does not consider the underlying road network, *e.g.*, Euclidean distance, the trajectory T_B is more similar to T_C than T_A. However, on the road network, T_B is more similar to T_A than T_C, as no route exists between between T_B and T_C. A desirable distance measure D should thus take into account that $D(T_A, T_B) < D(T_A, T_C)$.

We then need to define a cost model and devise an algorithm to minimize the cost. Specifically, to devise a cost model, we focus on online clustering, as many recent web-based services use trajectories for the post-processing of search results, *e.g.*, clustering search results based on spatial proximity. In such cases, it is not feasible to (1) assume an index structure for the search results determined at a query time or (2) build an index at a query time. We thus consider clustering algorithms without the prior existence of index structures and use the number of distance computations as a cost model, because it dominates the overall cost of clustering trajectories– Computing the distance between two trajectories is expensive, as each trajectory typically consists of potentially thousands of sampling points (see Section 6.1).

To address this problem, we first propose a baseline algorithm which applies agglomerative hierarchical clustering to our clustering problem. We then devise an improved algorithm which reduces the distance computations by up to 80% against the baseline algorithm.

We can summarize the contributions of our paper as follows:

- We studied clustering problems for trajectories constrained by road networks.
- We identified road-network proximity which is satisfied by a clustering algorithm for network-trajectories.
- We proposed a distance measure which satisfies road-network proximity.
- We presented an efficient clustering algorithm.

– We extensively validated the efficiency and the quality of our proposed clustering algorithm using real-life trajectory data.

The rest of the paper is organized as follows. Section 2 presents some prior research work. In Section 3, we present our definition of the problem. Section 4 and Section 5 describe our proposed the distance measure and the clustering algorithm, respectively. Section 6 reports our experimental results. Section 7 concludes the paper.

2 Related Work

In this section, we survey prior research efforts on trajectory clustering algorithms and distance measures for trajectory data.

2.1 Clustering Algorithm

As good examples of clustering methods for trajectories, we can looks at two representative clustering algorithms as proposed in [14] and [20]. Gaffney *et al.* [14] proposed a trajectory clustering algorithm, which mainly focused on grouping similar trajectories as a whole. They approach was to model a set of trajectories using a regression mixture model and then use an EM (Expectation-Maximization) algorithm to determine the cluster memberships. However, due to the slow convergence of the EM algorithm, applying this algorithm to our problem is not appropriate as an instant online clustering response is needed.

TRACLUS [20] was developed to cluster common sub-trajectories. It uses a *partition-and-group* framework which divides trajectories into line segments and then merges similar line segments. Specifically, TRACLUS uses the minimum description length (MDL) principle to represent trajectories in a *partition* phase and a density-based clustering algorithm, *i.e.*, DBSCAN [13], to cluster trajectory partitions in a *group* phase. A known drawback of density-based clustering algorithms is their assumption that trajectories in the same cluster have a rather homogeneous density. However, as we will observe from real-life trajectories in Section, 6.1 densities in the same cluster can vary significantly, which discouraged us from applying TRACLUS for our proposed problem.

2.2 Distance Measure for Trajectory Data

We can categorize the distance measures proposed for trajectories into two groups– order-dependent distance measures and order-independent distance measures.

Order-dependent distance measures [21,7,26,10,9,17] build upon trajectories which are represented as a *sequence* of points in two or three dimensional space. Then they compare how closely the two sequences align with each other. LCSS [26] is rather robust to noise by quantizing the distance between a pair of elements as either 0 or 1. However, it is well known that LCSS could be inaccurate because it neglect to consider the *gap g* between similar subsequences. To address this problem, Lei Chen *et al.* [10] proposed EDR which considers the gap penalty between the two matching subsequences. ERP [9] is a variant of EDR, which does not quantize the distance between elements.

Order-independent distance measures [24,18,15,5,8,20,23] build upon trajectories which are represented as a *set* of points or line segments. In this line of research, the time information of trajectories is ignored and the distance between trajectories is measured with respect to the *shape* of the trajectory. For this purpose, Hausdorff distance has been used to measure the distance between two trajectories in [24,18,15,5,8]. Lin *et al.* proposed the distance measure shown in [23] based on a modified Hausdorff distance [12].

However, no measure discussed above fully reflects road-network proximity and thus cannot be applied to our problem "as is".

3 Problem Definition

We assume that the road network can be represented as a graph where a vertex is the intersection of the road network and the edge is a road segment. We can thus give a formal definition of the road network as follows.

Definition 1 (Road network). *A road network RN is defined as a graph* $G_{RN} = (V, E)$*, where* V *is a set of intersections of the road network, and* E *is a set of road segments* $R_i \in E$ *such that*

$$R_i = (r_{i,s}, r_{i,e}),$$

where $r_{i,s}, r_{i,e} \in V$ *and there exists an road between* $r_{i,s}$ *and* $r_{i,e}$*.*

We assume that a trajectory is given as a *connected* sequence of road segments, which is obtained using a map matching algorithm [6], in such a way that each road segment of a trajectory should be connected to the next road segment in the sequence, *i.e.*, the ending position of each segment is the starting position of the next segment in the sequence.

Definition 2 (Trajectory). *Given a road network* $G_{RN} = (V, E)$ *and a set of trajectory* T*, a trajectory of length* l *is defined as*

$$TR_i = \left[t_1^i, \ldots, t_l^i\right],$$

where $t_j^i \in E$*,* $1 \leq j \leq l$*, and* t_j^i *and* t_{j+1}^i *are connected.*

For a given set of trajectories $T = \{TR_1, \cdots, TR_n\}$ on a road network RN, our clustering algorithm employs a set of clusters $\mathbf{C} = \{C_1, \ldots, C_k\}$, which are defined as follows.

Definition 3 (Cluster). *Given input trajectories* T*, a cluster* $C_i \in \mathbf{C}$ *is a set of trajectories in* T *such that*
1. $C_i \neq \varnothing$*,* $i = 1, \ldots, k$*;*
2. $\bigcup_{i=1}^k = C$*;*
3. $C_i \cap C_i = \varnothing$*,* $i, j = 1, \ldots, k$ *and* $i \neq j$*.*

4 Distance Measure

This section presents a novel distance measure for network trajectories. To generally support both order-dependent and order-independent measures, we focus on devising an order-independent measure as a lower bound for both distance measures. To illustrate the differences, Figure 2 illustrates two trajectories that move along the same route but in opposite directions. Assume that the two trajectories have the same sampling points $(t_0, \cdots, t_5)$. T_A moves from t_5 to t_0 and T_B moves from t_0 to t_5. When the distance measure becomes sensitive to the order of sampling points, *i.e.*, an order-dependent measure, T_A is thought to be totally different from T_B. In contrast, when using order-independent measures, the two trajectories are considered as being identical.

This suggests that ranking pairwise distances by order-independent measure "underestimates" the order-dependent distances. We can thus obtain a ranking of order-dependent distances by using order-independent metrics (if required by an application), by simply adding a post-filtering step to discard the "false positives". We can thus focus on order-independent metrics.

The underlying idea of our distance measure is that we determine the distance between two trajectories as the longest distance that an adversary can force you to travel from one road segment to another.

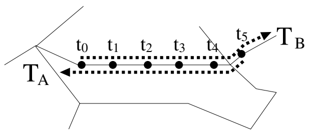


Fig. 2. Effects on the order-dependence of distance measure

4.1 Road Segment Distance

First, we define the distance $d(R_i, R_j)$ between two road segments R_i and R_j and based on this, we will define the distance between the trajectories later on.

The road segment distance between R_i and R_j, is therefore defined as follows:

Definition 4 (Road segment distance)

$$d(R_i, R_j) = \max \left\{ \overrightarrow{d}(R_i, R_j), \overrightarrow{d}(R_j, R_i) \right\}, \tag{1}$$

where $\overrightarrow{d}(R_i, R_j)$ *is a one-way road segment distance from* R_i *to* R_j.

The *one-way* road segment distance $\overrightarrow{d}(R_i, R_j)$ is defined as follows:

Definition 5 (*One-way* road segment distance)

$$\overrightarrow{d}(R_i, R_j) = \max \left\{ \begin{array}{l} \min\{\|r_{i,s}, r_{j,s}\|, \|r_{i,s}, r_{j,e}\|\}, \\ \min\{\|r_{i,e}, r_{j,s}\|, \|r_{i,e}, r_{j,e}\|\} \end{array} \right\}, \tag{2}$$

where $\|a, b\|$ *is the length of the shortest path between* a *and* b.

4.2 Trajectory Distance

Based on the road segment distance defined in the previous section, we now move on to define the distance measure between two trajectories.

Definition 6 (Trajectory distance). *Given two trajectories* $TR_i = \left[t_1^i, \ldots, t_n^i\right]$ *and* $TR_j = \left[t_1^j, \ldots, t_m^j\right]$*, the distance between them is defined as follows:*

$$D(TR_i, TR_j) = \max\left\{\overrightarrow{D}(TR_i, TR_j), \overrightarrow{D}(TR_j, TR_i)\right\}, \tag{3}$$

where $\overrightarrow{D}(TR_i, TR_j)$ *is the one-way trajectory distance from* TR_i *to* TR_j*.*

Definition 7 (*One-way* trajectory distance). *Given two trajectories* TR_i *and* TR_j*, the one-way trajectory distance is defined by*

$$\overrightarrow{D}(TR_i, TR_j) = \max_{t_a^i \in TR_i} \min_{t_b^j \in TR_j} d(t_a^i, t_b^j) \tag{4}$$

The *One-way* trajectory distance finds the road segment t_a^i that is the farthest from road segments of TR_j, and calculates the distance from t_a^i to its nearest road segment in TR_j.

While this notion of Hausdorff distance was used in prior work [24,18,15,5,8], they used Euclidean distance to quantify the distance between point pairs. In a clear contrast, we consider network constraints when quantifying the distance between point pairs, as the longest path from each road segment to its closest road segment on another trajectory. To illustrate this idea, consider two trajectories A and B in Figure 3. The two trajectories share the same movement until the point where A keeps moving straight while B makes a right turn. This suggests the distance between A and B is 0, until the two objects take different paths. Afterwards, the travel distance for A to reach B, or vice versa, is now either the distance of A making a U-turn to B, as represented as a dashed line in the figure, or B making a U-turn to A. As the former distance is longer, we use the length of the dashed line as the distance between the two trajectories.

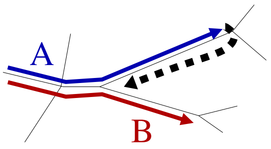


Fig. 3. Meaning of trajectory distance

Theorem 1. *Our proposed trajectory distance is a metric.*

Proof. Due to space limitation, we omit the proof here but it is in the extended version [25].

5 Clustering Algorithm

In this section, we present the clustering algorithm for trajectory data. We first present the baseline algorithm which employs the agglomerative hierarchical clustering algorithm. Then we propose our clustering method which is more efficient than the baseline algorithm.

5.1 Baseline Algorithm

This section presents the baseline algorithm which adopts agglomerative hierarchical clustering (HC) method to our clustering problem. Each trajectory is regarded as a singleton cluster and then the clusters are merged until all the trajectories are eventually assigned to a single cluster.

More specifically, we first compute a distance matrix (DM) that contains the distances between all pairs of clusters. DM_{ij} contains the distance between the two clusters. At each particular step, we merge two clusters that have the minimum distance value in DM. Assume that TR_i and TR_j are merged into a new cluster. Whenever a new cluster is generated, we then update DM by re-calculating the distances of the new cluster against all of the existing clusters (except TR_i and TR_j). For this update, we need to define the distance between two clusters, which we will discuss later in Section 5.2. This algorithm terminates when all of the trajectories are merged into a single cluster.

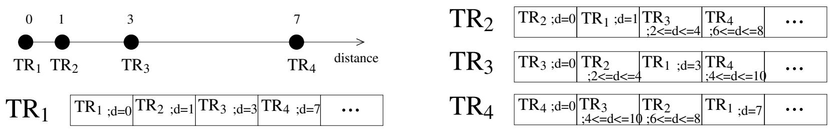
However, this baseline algorithm requires a large number of distance computations which significantly degrades the performance of the clustering algorithm. More specifically, building a distance matrix for any N number of trajectories requires $\frac{N \times (N-1)}{2}$ distance computations. As mentioned in Section 1, computing the distance between two trajectories is expensive. While there are many HC techniques [27] that address the high computational costs of the classical HC algorithm, they generally introduce a data structure for efficiency, which cannot be used for our target problem of online clustering.

We therefore devise a technique to reduce distance computations not requiring an index structure, without much loss in the quality of clustering results.

5.2 NNCluster

This section proposes an efficient method for clustering trajectories. The underlying principle of our clustering method is that trajectories belong to the same cluster if they share common nearest neighbors. This is inspired by the distance relationship in which inter-cluster trajectories have a higher proximity than intra-cluster trajectories.

Conceptually, clustering trajectories into a group that share common nearest neighbors can be viewed in the same way as clustering data points in a Voronoi cell defined by the common nearest neighbor as shared by all points in the cell. Even though applying the Voronoi diagram for clustering has been studied in [19], applying this to our problem requires the actual materialization of the Voronoi diagram on the trajectory data. However, building Voronoi diagrams of



(a) Exact sorted list for TR_1 (b) *Derived* sorted lists

Fig. 4. Example of approximating sorted list (TR_1 is a seed trajectory)

network trajectories would incur a prohibitive computational cost, as it needed exponential cost over the number of trajectories [4].

We will thus develop an algorithm that incrementally and approximately materializes the cells, which consists of the following two steps: construction of initial cluster and expansion of initial clusters.

Construction of initial cluster: To efficiently construct an initial cluster, we compare k-nearest neighbors of each trajectory with those of another trajectory. The two trajectories are then merged into a cluster if the ratio between the common nearest neighbors is greater than a user-defined threshold value.

More specifically, we repeat the following processes until no further merging is possible. To reduce the distance computation, at each iteration, we randomly select a *seed* trajectory TR_{seed} among trajectories that have yet to be assigned to any initial cluster and compute a sorted list for TR_{seed}, which contains the distances between TR_{seed} and all of the other trajectories in an ascending order. Figure 4(a) shows an example of such a sorted list for TR_{seed}, when TR_1 is selected as a *seed* trajectory. From the top trajectory in the list, *e.g.*, TR_2 in Figure 4(a), we proceed to test whether the ratio of common nearest neighbors of TR_2 and TR_{seed} is greater than the user-defined threshold value. To find the nearest neighbors of TR_2, we need to compute a sorted list for TR_2 as TR_{seed}. By building exact sorted lists only for seed trajectories, we can avoid the prohibitive cost of building exact lists for every trajectory.

For the remaining "non-seed" trajectories, instead of building exact sorted lists, we generate approximate lists that are based on the exact lists of seed trajectories. This suggests that once we compute the *exact* sorted list for a *seed* trajectory, we can then estimate the sorted lists of the remaining non-seed ones by approximating the distance based on the *exact* sorted lists on the seed trajectories. Hereafter we call an approximated sorted list a *derived* sorted list. For *derived* sorted lists, each element of the list has a lower bound distance and an upper bound distance. Trajectories are sorted first by the lower bound and the ties in lower bound (if they exist) are broken by upper bounds.

For example, we can illustrate how to approximate the distance between TR_2 and TR_3, when we already have the sorted lists for TR_1 as illustrated in Figure 4(a). Figure 4(b) shows *derived* sorted lists for TR_2, TR_3, and TR_4 after we compute the exact sorted list for the trajectory TR_1.

As our distance measure satisfies the property of triangle inequality, the following two inequalities are also satisfied.

$$D(TR_2, TR_3) \leq D(TR_1, TR_2) + D(TR_1, TR_3) \tag{5}$$

$$D(TR_1, TR_3) \leq D(TR_1, TR_2) + D(TR_2, TR_3) \tag{6}$$

By using the above two inequalities, we can compute the lower and upper bound of $D(TR_2, TR_3)$ as follows:

$$\begin{aligned} & D(TR_1, TR_3) - D(TR_1, TR_2) \\ & \leq D(TR_2, TR_3) \\ & \leq D(TR_1, TR_2) + D(TR_1, TR_3). \end{aligned} \tag{7}$$

After a *derived* sorted list is established for a *non-seed* trajectory TR_i, we can compare k-nearest trajectories of TR_i with those of TR_{seed}. If the ratio of the common trajectories to the union of the two sets of k-nearest neighbors is greater than the user-defined threshold, we can merge TR_i with TR_{seed}.

Expansion of initial cluster: During construction of initial clusters, some clusters can be divided into multiple initial clusters. Therefore, in this step, we need to merge initial clusters, if two clusters are closer than the user-specified threshold.

There are two challenges in this step: *First* we need to define the distance measure between clusters. *Second*, a stop condition is required to determine when the expansion of the initial cluster terminates.

To address the fist challenge, we define the distance between clusters as follows:

Definition 8 (Distance between clusters). *Given two clusters, c_i and c_j, the distance between them is defined by*

$$D_c(c_i, c_j) = D(rt(c_i), rt(c_j)), \tag{8}$$

where $rt(\cdot)$ is the representative trajectory of a cluster and $D(\cdot, \cdot)$ is the trajectory distance (Equation 3).

As a representative trajectory, we choose the trajectory that minimizes the average distance between the trajectories in a cluster.

Definition 9 (Representative trajectory)
Given cluster $c_i = \{TR_1, \ldots, TR_l\}$, a representative trajectory, $rt(c_i)$, is defined by

$$rt(C_i) = \underset{TR_i \in C_i}{\arg\min} \frac{1}{l} \sum_{m=1}^{l} D(TR_i, TR_m). \tag{9}$$

For the second challenge, *i.e.*, a stop condition, we first introduce quality measures which have been used to validate the clustering results. Then we present a method to exploit the quality measure as a stop condition for our clustering method.

Many quality measures (*e.g.*, Dunn Index, Silhouette Index, and Davies-Bouldin (DB) Index) have been introduced previously to address issues of this type[1].

These quality measures were originally developed to validate clustering results *after* a clustering process terminates. Therefore, the cost for computing a quality measure has not been considered in the context of their application. However, to apply these measures in order to invoke a termination condition, we need to consider the computational overhead. We thus choose the DB Index [11] from the range of quality measures available, as this measure can be computed based on the distances computed during clustering and does not require additional computation on the values. The DB index is defined as:

$$DB_m = \frac{1}{m}\sum_{i=1}^{m} R_i,$$

where $R_i = \max_{j=1,\ldots,m j\neq i} \frac{s_i+s_j}{d_{ij}}$, $i = 1,\ldots,m$. d_{ij} is the distance between cluster C_i and cluster C_j, and s_i is the variance of cluster C_i. The DB index approaches zero value if the clusters are more compact and well-separated.

We store the value of the DB index at each step when merging clusters. Then, we report the clustering result which minimizes the DB index. Note that we ignore the last step, because the DB index becomes 0 if all of the trajectories are assigned to a single cluster.

6 Experimental Evaluation

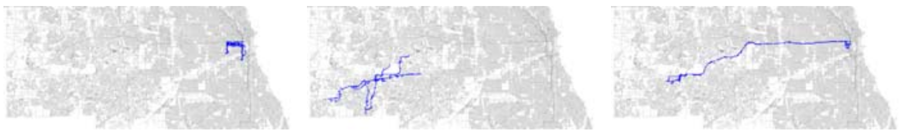
In this section, we will validate the efficiency and the effectiveness of our proposed clustering method with real-life trajectory data. More specifically, we will first describe the details of the real-life trajectory dataset used for our experiments in Section 6.1. We will then report on the efficiency of our evaluation results, using response time as metrics in Section 6.2. Next, We will validate the quality of the clustering results in Section 6.3. Lastly, we dicuss the effect of the parameter k on the performance of the proposed method and the quality of the results.

6.1 Experimental Settings

As our experimental dataset, we used a real-life trajectory dataset[2] that contained 214 trajectories collected as a sequence of positions from GPS receivers in two counties of Illinois, *i.e.*, Cook and DuPage county. The length of the trajectory ranged from 18 to 1486.

[1] The interested readers may find an exhaustive study of clustering validation in [16].

[2] `http://cs.uic.edu/~boxu/mp2p/gps_data.html`


(a) Group I
Avg:184, Min:30, Max:1486

(b) Group II
Avg:174, Min:18, Max:488

(c) Group III
Avg:901, Min:580, Max:1200

Fig. 5. Three groups of the real-life trajectories

To get "ground truth" clustering results, we manually classified the real trajectory data into three groups according to their movement patterns. Figure 5 visualizes the trajectories of each group on the road network and also shows statistics such as average, minimum, and maximum values of the trajectory length for each group. In group I, there were 97 trajectories restricted within a small region of Cook county (see Figure 5(a)) Figure 5(b) illustrates the 23 trajectories of group II, which were located within DuPage county. Group III included the trajectories of those traveling from one county to another, covering 93 trajectories in the dataset.

We can observe from this real-life dataset that trajectories in the same cluster can vary in "density" within the group. To illustrate, the trajectories in group III are identical to each other except for two trajectories that are missing road segments. This suggests that the density of the identical trajectories is high, while that of the remaining two is much lower.

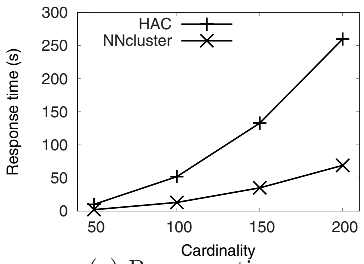
As density-based clustering algorithms are known to fail when clusters have "heterogeneous" densities within the group, applying the algorithms to this set is not desirable. In contrast, this section shows how our proposed algorithm identifies highly accurate clusters for real-life data.

All experiments were conducted on a machine with a Pentium-4 (3.2GHz) CPU and 1 GBytes for its main memory, running a version of the Linux operating system. We implemented our proposed algorithms using the C programming language.

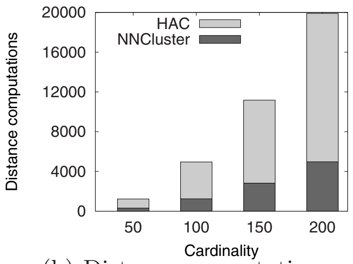
6.2 Efficiency

In this section, we will validate the efficiency of proposed method using a response time. To observe the performance of our algorithm with changing cardinality, we evaluated sample trajectories from the real-life dataset discussed above, of size 50, 100, 150, and 200. To ensure the representativeness of these samples, we made sure that each sample set maintained the same distribution over three categories. In other words, we sampled to make sure that the ratio of categories in each of three categories remained unchanged both in the original dataset and sample sets.

As shown in Figure 6(a), our proposed clustering algorithm outperformed the baseline algorithm in all of the experimental settings in this section. Overall, our proposed algorithm was 4 or 5 times faster than the baseline algorithm.



(a) Response time



(b) Distance computations

Fig. 6. Efficiency test for different cardinalities

Figure 6(b) shows the number of distance computations, performed by our proposed algorithm and the baseline algorithm. It is clear that our proposed algorithm reduces the number of distance computations by four times as much, as similarly observed trend for the response time in Figure 6(a). These results suggest that our proposed algorithm can efficiently cluster trajectories by reducing the number of distance computations, which directly influences the overall performance of clustering algorithms.

6.3 Quality of the Clustering Result

In this section, we will validate the quality of the clustering results, produced by NNCluster, by comparing the clusters of NNCluster with the original class of trajectories. To visualize the clustering results with the dendrograms in this section, we focus on the smallest sample set of 50 trajectories used in Section 6.2. Table 1 presents the sample trajectory data with a list of trajectory IDs extracted from each "ground truth" group.

Table 1. Sample trajectory data

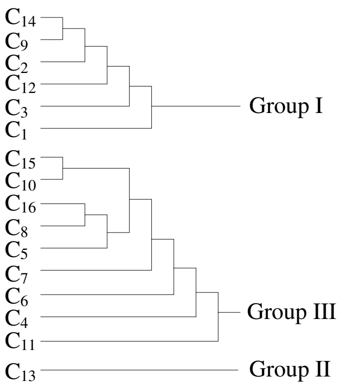
Group	Size	Trajectory IDs
I	22	121 $\sim$ 142
II	6	1 11 100 112 $\sim$ 114
III	22	0 10 13 14 15 17 18 101 $\sim$ 111 117 $\sim$ 120

We now validate the quality, by comparing clusters generated by our algorithm, with the those of the ground-truth clustering (of the three groups of trajectories discussed in Figure 5). Recall that, NNCluster starts with a set of initial clusters and then merges them in a bottom-up fashion, which can be represented as a cluster hierarchy. We summarized the initial clusters in Figure 7(a), and then represented their clustering hierarchy in Figure 7(b) with a dendrogram.

It is clear that, initially, there were 16 clusters identified by NNCluster. Figure 7(a) lists the initial clusters with their representative trajectory marked with

cluster id	member(s)
C_1	128* 139 129 141 140
C_2	127*
C_3	142* 131 132
C_4	102*
C_5	108* 107
C_6	17*
C_7	14*
C_8	119*
C_9	137* 124 130 122 121 136
C_{10}	109* 118 18 120 103 101
C_{11}	0*
C_{12}	125* 126
C_{13}	100* 112 11 1 114 113
C_{14}	133* 123 134 138 135
C_{15}	105* 111 10
C_{16}	110* 104 106 13 15 117

(a) Initial clusters of NNCluster



(b) Dendrogram

Fig. 7. Clustering results of NNCluster

an asterisk. The quality of initial clusters can be validated by the fact that every initial cluster belongs to the same cluster group.

After merging the initial clusters, *i.e.*, at the second step of NNCluster, NNCluster finally produces three clusters as shown in Figure 7(b). It is apparent that the three clusters produced by NNCluster are equivalent to the three ground-truth cluster groups. This observation suggests that NNCluster correctly partitions the sample trajectory data.

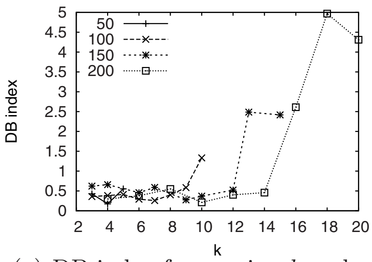
6.4 Effects on Parameter k

In this section, we discuss the effect of the user parameter k, when deciding the number of neighbors to consider, on the performance of NNCluster.

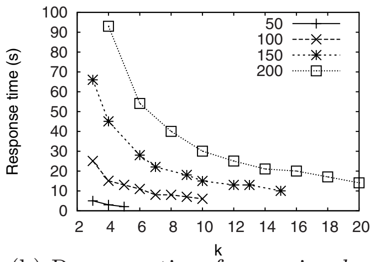
Intuitively, as observed above, k determines the trade-off between the performance and the quality of NNCluster. For a smaller k, NNCluster identifies more initial clusters of smaller sizes, and eventually, when $k = 1$, our algorithm degenerates to the baseline algorithm. For a larger k, NNCluster identifies a smaller number of bigger initial clusters and terminates earlier, but starting with large initial clusters may negatively affect the output cluster quality. In an extreme case when $k = n$, NNCluster will identify a single cluster containing all of the objects.

It is thus important for NNCluster to tune k to the right value. To find the right value of k, we conducted extensive experiments by varying the value of k over the four sample dataset as described in Section 6.2. Figure 8 shows the decreasing quality (in Figure 8(a)) and response time (in Figure 8(b)) for larger k.

It is clear that, as k increases, the quality is not affected up to a certain point and that the performance of NNCluster improves. Based on this observation, we suggest that the value of k is set to the largest possible value that does not degenerate the quality. In all of the sample dataset, NNCluster shows best



(a) DB index for varying k and cardinality



(b) Response time for varying k and cardinality

Fig. 8. Effects on the parameter k

performance and a reasonable quality when k is set to an amount of 7 percent of sample dataset.

7 Conclusion

In this paper, we studied how to cluster network trajectories in an on-line environment. To work towards this goal, we first investigated the proximity for network trajectories, *i.e.*, road-network proximity, and then devised a distance measure, which fulfilled the required properties. With the proposed distance measure, we presented an efficient clustering algorithm NNCluster, which is based on an intuitive notion of common nearest neighbors.

We validated the efficiency and the quality of the clustering results using real-life road network trajectories. To validate the efficiency of NNCluster, we conducted experiments on sample trajectory data varying the cardinality of the trajectories. NNCluster outperformed the baseline approach in all samples. We then validated the quality of the clustering results of NNCluster. NNCluster correctly partitioned the trajectory data, which made it equivalent to the result from ground-truth clustering. Our results demonstrate that NNCluster identifies identical clusters to ground-truth clustering but incurs only the 20% of the baseline computation cost.

References

1. Bikely, http://www.bikely.com/
2. Gps route sharing, http://www.sharemyroutes.com/
3. Gps track route exchange, http://www.gpsxchange.com/
4. Bereg, S., Buchin, K., Buchin, M., Gavrilova, M.L., Zhu, B.: Voronoi diagram of polygonal chains under the discrete fréchet distance. In: Hu, X., Wang, J. (eds.) COCOON 2008. LNCS, vol. 5092, pp. 352–362. Springer, Heidelberg (2008)
5. Biliotti, D., Antonini, G., Thiran, J.P.: Multi-layer hierarchical clustering of pedestrian trajectories for automatic counting of people in video sequences. In: Proc. WACV/MOTION, pp. 50–57 (2005)

6. Brakatsoulas, S., Pfoser, D., Salas, R., Wenk, C.: On map-matching vehicle tracking data. In: Proc. VLDB, pp. 853–864 (2005)
7. Cai, Y., Ng, R.T.: Indexing spatio-temporal trajectories with chebyshev polynomials. In: Proc. SIGMOD, pp. 599–610 (2004)
8. Cao, H., Wolfson, O.: Nonmaterialized motion information in transport networks. In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, pp. 173–188. Springer, Heidelberg (2004)
9. Chen, L., Ng, R.T.: On the marriage of lp-norms and edit distance. In: Proc. VLDB, pp. 792–803 (2004)
10. Chen, L., Özsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: Proc. SIGMOD, pp. 491–502 (2005)
11. Davies, D.L., Bouldin, D.W.: A cluster separation measure. IEEE Transactions on Pattern Analysis and Machine Intelligence 1(2), 224–227 (1979)
12. Dubuisson, M.P., Jain, A.: A modified hausdorff distance for object matching. In: Proc. ICPR, pp. 566–568 (1994)
13. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proc. KDD, pp. 226–231 (1996)
14. Gaffney, S., Smyth, P.: Trajectory clustering with mixtures of regression models. In: Proc. KDD, pp. 63–72 (1999)
15. Gudmundsson, J., van Kreveld, M.J., Speckmann, B.: Efficient detection of motion patterns in spatio-temporal data sets. In: GIS, pp. 250–257 (2004)
16. Halkidi, M., Batistakis, Y., Vazirgiannis, M.: On clustering validation techniques. Int. J. Intell. Inf. Syst. 17(2-3), 107–145 (2001)
17. Jeung, H., Yiu, M.L., Zhou, X., Jensen, C.S., Shen, H.T.: Discovery of convoys in trajectory databases. PVLDB 1(1), 1068–1080 (2008)
18. Junejo, I.N., Javed, O., Shah, M.: Multi feature path modeling for video surveillance. In: Proc. ICPR, pp. 716–719 (2004)
19. Koivistoinen, H., Ruuska, M., Elomaa, T.: A voronoi diagram approach to autonomous clustering. In: Todorovski, L., Lavrač, N., Jantke, K.P. (eds.) DS 2006. LNCS (LNAI), vol. 4265, pp. 149–160. Springer, Heidelberg (2006)
20. Lee, J.G., Han, J., Whang, K.Y.: Trajectory clustering: a partition-and-group framework. In: Proc. SIGMOD, pp. 593–604 (2007)
21. Lee, S.L., Chun, S.J., Kim, D.H., Lee, J.H., Chung, C.W.: Similarity search for multidimensional data sequences. In: Proc. ICDE, pp. 599–608 (2000)
22. Li, Q., Zheng, Y., Xie, X., Chen, Y., Liu, W., Ma, W.Y.: Mining user similarity based on location history. In: Proc. GIS, pp. 1–10 (2008)
23. Lin, B., Su, J.: One way distance: For shape based similarity search of moving object trajectories. Geoinformatica 12(2), 117–142 (2008)
24. Lou, J., Liu, Q., Tan, T., Hu, W.: Semantic interpretation of object activities in a surveillance system. In: Proc. ICPR, pp. 777–780 (2002)
25. Roh, G., Hwang, S.: Nncluster: An efficient clustering algorithm for road network trajectories, `http://ids.postech.ac.kr/nnClusterExtended.pdf`
26. Vlachos, M., Gunopulos, D., Kollios, G.: Discovering similar multidimensional trajectories. In: Proc. ICDE, pp. 673–684 (2002)
27. Xu, R., Wunsch, D.I.: Survey of clustering algorithms. IEEE Transactions on Neural Networks 16(3), 645–678 (2005)
28. Zheng, Y., Zhang, L., Xie, X., Ma, W.Y.: Mining interesting locations and travel sequences from gps trajectories. In: Proc. WWW, pp. 791–800 (2009)

Dynamic Skyline Queries in Large Graphs

Lei Zou[1], Lei Chen[2], M. Tamer Özsu[3], and Dongyan Zhao[1,4,⋆]

[1] Institute of Computer Science and Technology, Peking University, Beijing, China
{zoulei,zdy}@icst.pku.edu.cn
[2] Hong Kong of Science and Technology, Hong Kong, China
leichen@cse.ust.hk
[3] University of Waterloo, Waterloo, Canada
tozsu@cs.uwaterloo.ca
[4] Key Laboratory of Computational Linguistics (PKU), Ministry of Education, China

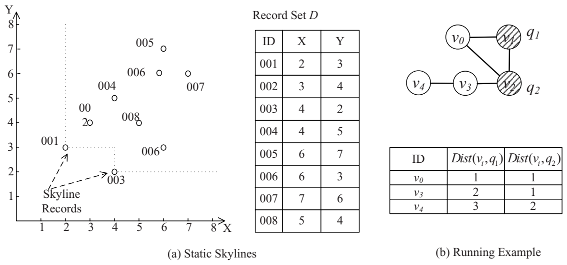
Abstract. Given a set of query points, a dynamic skyline query reports all data points that are not dominated by other data points according to the distances between data points and query points. In this paper, we study *dynamic skyline queries in a large graph* (DSG-query for short). Although dynamic skylines have been studied in Euclidean space [14], road network [5], and metric space [3,6], there is no previous work on dynamic skylines over large graphs. We employ a filter-and-refine framework to speed up the query processing that can answer DSG-query efficiently. We propose a novel pruning rule based on graph properties to derive the candidates for DSG-query, that are guaranteed not to introduce false negatives. In the refinement step, with a carefully-designed index structure, we compute short path distances between vertices in $O(H)$, where H is the number of maximal hops between any two vertices. Extensive experiments demonstrate that our methods outperform existing algorithms by orders of magnitude.

1 Introduction

As a popular multi-criteria decision making and business planning operator, *skyline* has attracted considerable attention. Given a record set $\mathcal{D}$ of n dimensions, a *skyline query* over $\mathcal{D}$ returns a set of records that are not dominated by any other record in $\mathcal{D}$ [1]. A record r is said to dominate another record r', if and only if the value of r is no larger than that of r' in each dimension, and the value of r is smaller than that of r' in at least one dimension. Fig. 1 shows a simple skyline query example. Given a record set D with 8 records, only 001 and 003 are reported as skyline records, since they are not dominated by any other record in D. The others are all dominated by 001 or 003. For example, 002 is dominated by 001, since $2 < 3$ in dimension x and $3 < 4$ in dimension y. Based on the skyline definition, given a record set $\mathcal{D}$, the skylines of $\mathcal{D}$ are fixed, thus, we refer to the skylines following the original definition as *static* skylines [3].

⋆ Lei Zou and Dongyan Zhao were supported by the National High Technology Research and Development Program("863"Program) of China (No. 2009AA01Z408). Lei Chen was supported by NSFC/RGC joint research scheme under project no. N_HKUST612/09. M. Tamer Özsu's research has been supported by Natural Sciences and Engineering Research Council (NSERC) of Canada.

In some cases, the values of records are computed at run time based on the values of query points, even for the same record set D, given different query points, we might obtain different skylines. We refer to these as *dynamic skylines*. These have been studied in various contexts. For example, "spatial" skylines have been proposed in *Euclidean* space [14]. Specifically, given a set of query points $Q = \{q_i\}$ $(i = 1...n)$, for each record $r \in \mathcal{D}$, we compute a new vector r^d of dimension n, where r^d's i-th dimension is computed as *Euclidean Distance* between r and q_i. The spatial skylines refer to all vectors r^d whose values are not dominated by other r'^d in the record set $\mathcal{D}$. A similar query, called multi-source skyline query in road networks, is studied by Deng et al. [5], where the values of records are defined as the shortest path lengths on road networks from data points to query points.



ID	X	Y
001	2	3
002	3	4
003	4	2
004	4	5
005	6	7
006	6	3
007	7	6
008	5	4

ID	$Dist(v_i, q_1)$	$Dist(v_i, q_2)$
v_0	1	1
v_3	2	1
v_4	3	2

Fig. 1. (a)Static Skyline Query and (b) Running Example

Compared to static skylines, dynamic skylines offer users more flexibility in specifying their search criteria. In other words, different users can specify different sets of query points. Meanwhile, the flexibility of dynamic skyline queries brings new challenges for efficient query processing. A naive solution computes all the new vectors according to the query points, and then searches the skylines over the generated vectors. This approach is clearly inefficient, since it requires scanning the whole record set $\mathcal{D}$ to compute the new vectors.

In this paper, we study the problem of dynamic skyline queries over graph data, which is formally defined as follows:

Definition 1. ***Dominate****. Given a large undirected and edge-weighted graph G and a set of query vertices $Q = \{q_i\}$, $i = 1...n$, in graph G, for two data vertices v' and v in G (v and v' are not query vertices), we say that v'* dominates *v, if and only if the following holds: $(\forall i, Dist(v', q_i) \le Dist(v, q_i)) \wedge (\exists j, Dist(v', q_j) < Dist(v, q_j))$, where $Dist(v, q_i)$ is the shortest path distance between v and q_i in graph G.*

Definition 2. ***Problem Definition****. Given a large undirected and edge-weighted graph G and a set Q of query vertices $Q = \{q_i\}$, $i = 1...n$, in graph G, a dynamic skyline query in graph (DSG-query) reports all data vertices v in graph G ($v \neq q_i$) where each v is not dominated by any other vertex v' in G. All skyline vertices of query Q in graph G are denoted as $Skyline(G, Q)$.*

Example 1 (Running Example). Consider a graph G and two query vertices q_1 and q_2 (denoted as shaded vertices) in Fig. 1b. The number in the vertex is the vertex ID. For simplicity, we assume that all edges have the same weight 1. Obviously, in this example, there is only one skyline vertex, that is v_0.

Similar to the cases in the Euclidean space, dynamic skylines over graph data are quite useful. For example, given a social network modeled as a large graph, where each vertex corresponds to an individual, and each edge denotes the friendship between two corresponding individuals, we can use shortest path distance to define the relationship score between two individuals in a social network [16]. Assuming that there are two important latent customers (two query vertices), a company may look for a salesman who has "closer" relationship to these customers than any other salesmen. In fact, the company is looking for the skyline salesmen with respect to the two given potential customers c_1 and c_2. A salesman r is a skyline if and only if there exists no other salesman r', such that $Dist(r', c_1) < Dist(r, c_1)$ and $Dist(r', c_2) < Dist(r, c_2)$, where $Dist(r, c_i)$ denotes the shortest path distance between r and c_i. Finally, consider a Peer-to-Peer (P2P) network with a number of peers that are interested in some movies. In order to reduce the communication cost, we can put replicas of the movies on a node that is near these peers. Obviously, dynamic skylines with respect to these peers in the topology map (a graph) of this P2P network can provide some candidate nodes for storing the replicas.

Although efficient solutions have been proposed for dynamic skylines over Euclidean space [14], these cannot be applied to graphs. In a graph, the shortest path distance is often used as a measure between two vertices, rather than Euclidean distance. Thus, it is impossible to utilize existing pruning rules, such as MBR and Voronoi Diagrams that have been used in Euclidean space [14].

The most related work is multi-source skyline query processing in road networks [5], which also uses shortest path distance as the measure. Three different algorithms have been proposed to find dynamic skylines in road networks. Two of the algorithms (EDC and LBC) [5] utilize Euclidean distance as the lower bound of shortest path distance in a road network to perform pruning. However, for a general graph G, we cannot define Euclidean distances to bound the shortest path distances between any two vertices in G, since there is no coordinate associated with each vertex. Therefore, EDC and LBC algorithms cannot be applied to DSG-Query. The third algorithm (CE) is not efficient. It expands each query point towards all directions, which may generate too many candidate objects and cause unnecessary shortest path distance computation, as confirmed by our experiments (Section 4).

$Dist(v, q_i)$ in DSG-query is a metric distance. Thus, the approaches that address skyline queries in metric space (e.g. [3]) are of interest. However, these solutions are only designed for the general metric space, and the methods are not optimized for large graphs. Experiments in Section 4 show that our methods outperform these general approaches by orders of magnitude. For a DSG-query, there exist two challenges: 1) *Huge Search Space*: each vertex in graph G (except for query vertices) is a candidate for DSG-query; and 2) *Expensive Shortest Path Computation*: In order to find final skyline vertices, we need to compute $Dist(v, q_i)$ (see Definitions 1 and 2). However, the expansion process in shortest path algorithms (e.g. Dijkstra's Algorithm [4]) is very time-consuming, especially in very large graphs.

In order to address the above challenges, we adopt the "filter-and-refine" framework. During the filtering process, we prune most false positives (the vertices that cannot be skyline vertices) to generate a set of candidate vertices. We compute $Dist(v, q_i)$ for each candidate vertex during refinement using an index structure. Furthermore, we can compute $Dist(v, q_i)$ in $O(H)$ where H is the number of maximal hops between any two vertices in the graph, without expensive expansions that are employed in previous solutions. In summary, we make the following contributions:

1. We propose shared-shortest-paths (SSP) pruning for DSG-query, that considers graph properties, and that filters out most false positive vertices. We also give a theoretical analysis of the pruning power of SSP-pruning.
2. During offline processing of DSG-query, we build carefully-designed index structures that support both filtering and verification processes in online DSG-query.
3. Based on novel pruning rules and index structures, we propose the SSP query algorithm (see Algorithm 3) for DSG-query.
4. We show by extensive experiments that our methods have good pruning power and fast query response time.

The remainder of this paper is organized as follows. Some background knowledge is discussed in Section 2. A novel pruning rule is proposed in Section 3. The index structures and SSP-query algorithm are also presented in Section 3. We evaluate the efficiency of our methods with extensive experiments in Section 4. We discuss related work in Section 5 in detail. Finally, we conclude the paper in Section 6.

2 Preliminaries

Definition 3. ***Shortest Path Tree.*** *(SP-Tree for short) Given a large graph G and a vertex v, we perform a single-source shortest path algorithm (such as Dijkstra algorithm [4]) from vertex v to get a tree $SP(v)$. The root of $SP(v)$ is v, and all paths from v to another node v' in $SP(v)$ is the shortest path from v to v' in graph G.*

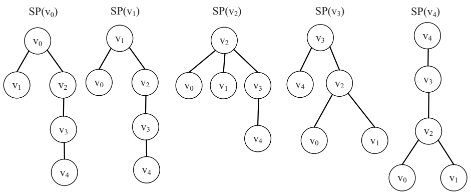


Fig. 2. Shortest Path Trees

Fig. 2 shows all SP-Trees for graph G of Example 1. We can perform Dijkstra's algorithm [4] offline to obtain the SP-Tree. Note that, for a vertex v in a large graph G, there may exist more than one SP-Tree rooted at v. Actually, for a vertex v in G, we can select any SP-Tree rooted at v without affecting the correctness of our methods. We will prove this claim in Section 3 (see Lemma 4). We utilize SP-Tree in our SSP query algorithm (see Section 3).

Definition 4. ***Minimum Common Ancestor***. *Given a SP-Tree $SP(v)$ in a large graph G and a set of nodes $\{v_1, ..., v_n\}$ in $SP(v)$, a node v' is the minimum common ancestor of $\{v_1, ..., v_n\}$ (denoted as $MCA(v_1...v_n, SP(v))$) if and only if*
1) v' is the ancestor of all nodes $v_1, ..., v_n$; and
2) there exists no other node v'', where v'' is the ancestor of all nodes $v_1, ..., v_n$, and v' is the ancestor of v''.

Take $SP(v_3)$ in Fig. 2, for example, $MCA(v_0, v_1, SP(v_3)) = v_2$.

3 Shared Shortest Path Algorithm

3.1 SSP Pruning Algorithm

We first note an interesting property of graphs: many pairwise shortest paths in a graph have shared parts. We propose a pruning strategy (SSP Pruning) that exploits these shared paths. We also give a theoretical analysis of pruning power of SSP pruning. Theoretical analysis and experiments confirm the effectiveness of SSP pruning. Furthermore, the indexing structures proposed for SSP can be used to support the efficient computation of $Dist(v, q_i)$ (discussed in Section 3.2).

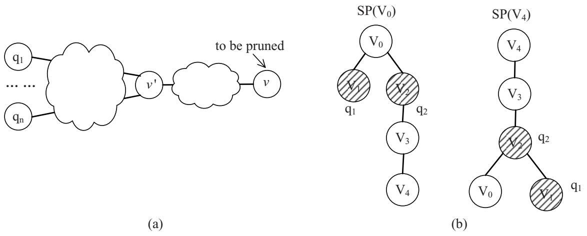


Fig. 3. (a)Shared Shortest Path Pruning and (b) Lemma 2

Pruning Rule 1: Shared Shortest Path (SSP) Pruning. Given a large graph G and a set of query vertices $Q = \{q_i\}$, $i = 1...n$, for a data vertex v, if there exists at least one joint (common) vertex v' among all shortest paths between v and q_i (denoted as $\overline{vq_i}$)($v \neq v'$, $i = 1...n$), v can be pruned safely for DSG-query.

Lemma 1. *SSP pruning will not lead to false negatives.*

Proof. Since vertex v' is in the shortest path $\overline{vq_i}$ (as shown in Fig. 3a), it is straightforward to see that $Dist(v', q_i) < Dist(v, q_i)$, $i = 1...n$. According to Definition 1, vertex v' dominates v. Therefore, v cannot be a skyline vertex, without causing false negatives.

Definition 5. ***Strictly Dominate, Candidate Skyline Vertex***. *Given a large graph G and a set Q of query vertices $Q = \{q_i\}$, $i = 1...n$, for a vertex v, if there exists a joint vertex v' ($v \neq v'$) in all shortest paths $\overline{vq_i}$, vertex v'* ***strictly dominates*** *vertex v.*

For a vertex v in graph G, if there exists no other vertex v' that ***strictly dominates*** *v, the vertex v is a* ***candidate skyline vertex***.

Theorem 1. *Given a large graph G and a set Q of query vertices $Q = \{q_i\}$, $i = 1...n$, the following formula holds:$Skyline(G,Q) \subseteq Can\ Skyline(G,Q)$, where $Skyline$ (G) (or $CanSkyline(G)$) is the set of* skyline vertices *(or* candidate skyline vertices*) in graph G.*

Proof. SSP-pruning will not lead to false negatives, therefore, $Skyline(G,Q) \subseteq Can$ $Skyline(G,Q)$

Obviously, $CanSkyline(G,Q)$ can be regarded as a candidate set for $Skyline(G,Q)$. In Example 1, we enumerate all SP-Trees in graph G, as shown in Fig. 2.

Lemma 2. *Given a large graph G, a set Q of query vertices $Q = \{q_i\}$, $i = 1...n$, and a SP-Tree $SP(v)$ and $v' = MCA(\ q_1...q_n, SP(v))$, v is a candidate skyline vertex if and only if $v = v'$.*

Proof. Proof is based on Definition 5. Due to space limit, we omit the details.

We show two shortest path trees $SP(v_0)$ and $SP(v_4)$ in Fig. 3b. In these trees, $MCA(q_1,$ q_2, $SP(v_4)) = v_3 \neq v_4$, therefore,$v_4 \notin CanSkyline(G,Q)$. However, $MCA(q_1,q_2,$ $SP(v_0)) = v_0$, thus, $v_0 \in CanSkyline(G,Q)$.

Using Lemma 2, we propose a conceptually simple framework for SSP-pruning. During offline processing, we enumerate all SP-Trees in graph G. Given a set Q of query vertices $Q = \{q_i\}$, $i = 1...n$, in each $SP(v)$, if $v' = MCA(q_1 ... \ q_n, SP(v))$ and $v' = v$, v will be inserted into candidate set CL. Otherwise, v can be pruned safely. We can find $CanSkyline(G,Q)$ by sequentially scanning all SP-Trees.

However, due to large space cost, it is impractical to store all SP-Trees of a large graph G. The space cost of each SP-Tree is $O(|V(G)|)$, where $|V(G)|$ is the number of vertices in G. Therefore, we would need $O(|V(G)|^2)$ space to store all SP-Trees in graph G. For example, if G has 10K vertices, the total space cost is $O(10^8)$. To alleviate this space cost, in this work, we only store 1-hop SP-Trees.

Definition 6. $1-$***Hop Shortest Path Tree*** *Given a large graph G and a vertex v in G,* 1-*Hop Shortest Path Tree from v (denoted as $SP(v,1)$) is obtained by extracting vertex v and vertices that are directly reachable from v in $SP(v)$. Each leaf node f in $SP(v,1)$ also has a set des of vertices (denoted as $f.des$), that corresponds to all descendants of the leaf node f in $SP(v)$. We call $f.des$ the* node area *of f.*

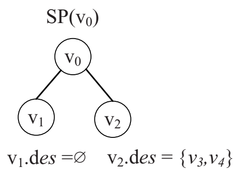


Fig. 4. $1-$Hop Shortest-Distance-Path Tree

Fig. 4 shows $SP(v_0,1)$ in Example 1, that is obtained by extracting vertices v_1 and v_2 that are directly reachable from v_0 in $SP(v_0)$ to form $SP(v_0,1)$. Since vertices v_3 and v_4 are descendants of v_2 in $SP(v_0)$, $v_2.des = \{v_3, v_4\}$.

Theorem 2. *Given a set of query vertices $Q = \{q_i\}, i = 1...n$, and 1-hop SP-Tree $SP(v,1)$, if there exists a leaf node f in $SP(v,1)$ and $f.des$ contains all query vertices, vertex v cannot be a skyline vertex.*

Proof. Proof is based on Definition 5. Due to space limit, we omit the details.

Based on Theorem 2, we propose Algorithm 1. For a vertex v, we only need to check whether all query vertices are in the same leaf node area $f.des$. If so, v can be pruned. Furthermore, Lemma 3 shows that using 1-hop shortest path tree does not affect the pruning power of Pruning Rule 1.

Lemma 3. *Given a large graph G and a set of query vertices $Q = \{q_i\}$, $i = 1...n$, for a data vertex v in G, if v is strictly dominated by another vertex v', then there must exist a leaf node f in $SP(v,1)$ and $f.des$ contains all query vertices.*

Proof. Since v' strictly dominates v, all query vertices are descendent of v' in $SP(v)$. According to Definition 6, it is straightforward to know all query vertices are in one leaf node area $f.des$.

Algorithm 1. Prune False Positives by SSP pruning

```
SSP-pruning(G,Q,S)
Require: Input: A large graph $G$ and a set $Q$ of query vertices $Q = \{q_i\}$, all 1-hop $SP$ Trees,
    and a set $S$ of vertices
    Output: Candidate Set $CL$
1: for each vertex $v$ in $S$ do
2:     if all query vertices $q_i$ are in only one leaf node $n.des$ of $SP(v,1)$ then
3:         continue (Pruned by Theorem 2 )
4:     else
5:         insert $v$ into candidate set $CL$.
6: report $CL$
```

Lemma 4. *Given a vertex v that has more than one SP-Tree rooted at v ($SP_1(v), ...,$ $SP_b(v)$), choosing any SP-Tree rooted at v for SSP pruning does not lead to false negatives.*

Proof. No matter which $SP_j(v)$ is selected, if all query vertices are contained in $f.des$ of $SP_j(v)$, f strictly dominates v. Therefore, Lemma 4 holds.

However, the space cost of the set $f.des$ in each leaf node f is still a problem. The number of vertices in $f.des$ is always large. In order to reduce the space cost, we build hierarchical clusters on vertices in G. If $f.des$ contains all vertices in a cluster P, we can use $P's$ ID instead of the vertices in P in $f.des$. For example, Fig. 5a shows a hierarchical cluster of Example 1. Cluster P_2 has two vertices: v_3 and v_4. Fig. 5b shows that v_3 and v_4 are both in $v_2.des$ of $SP(v_0,1)$. Therefore, we only need to insert P_2 instead of v_3 and v_4 in $v_2.des$ of $SP(v_0,1)$, as shown in Fig. 5c. Intuitively, if some vertices often occur together in $f.des$, they should be grouped together. Based on this intuition, we propose distance definitions (Definitions 7 and 8) that account for clustered vertices. In Example 1, vertices v_3 and v_4 should be grouped into one cluster, since they occur together in three 1-hop SP-Trees (i.e. $SP(v_0, 1)$, $SP(v_1, 1)$ and $SP(v_2, 1)$). Similarly, v_1 and v_2 should be clustered together. Essentially, the coherence of vertices in 1-hop SP-Tree is derived from their shared shortest paths of SP-Trees.

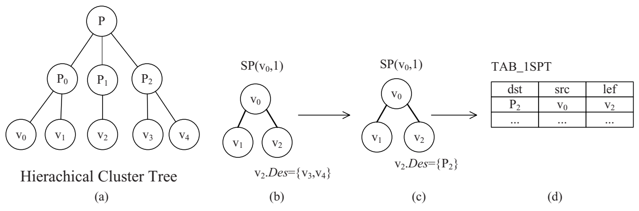


Fig. 5. Hierarchical Cluster Tree and 1-Hop SP-Tree

In order to build a hierarchical cluster on vertices, we propose the following distance functions, which are used to measure the probability that two vertices appear together.

Definition 7. ***Vertex Distance****. Given three vertices v, v_1 and v_2, if there exists a leaf node f in $SP(v, 1)$ and $f.des$ contains both v_1 and v_2, we say that v_1 and v_2 occur together in $SP(v, 1)$. Let the number of vertices v in which v_1 and v_2 occur together in $SP(v, 1)$ be T. Then, vertex distance between v_1 and v_2 (denoted as $VexDis(v_1, v_2)$) is defined as:*

$$VexDis(v_1, v_2) = \frac{T}{|V(G)|}$$

Definition 8. ***Cluster Distance****. Given two clusters P_1 and P_2 and a vertex v, if there exists a leaf node f in $SP(v, 1)$ and $f.des$ contains all vertices in P_1 and P_2, we say that P_1 and P_2 occur together in $SP(v, 1)$. Let the number of vertices v in which P_1 and P_2 occur together in $SP(v, 1)$ be D. Then, cluster distance between P_1 and P_2 (denoted as $CluDis(P_1, P_2)$) is defined as:*

$$CluDis(P_1, P_2) = \frac{D}{|V(G)|}$$

We employ bottom-up clustering to build the hierarchical clusters. First, based on vertex distances (Definition 7), we utilize clustering algorithms to find clusters on vertices. Then, based on cluster distance (Definition 8), small clusters are grouped into larger ones. We can recursively build a hierarchical cluster tree HT on vertices in graph G, as shown in Fig. 5a.

In practice, we store 1-hop SP-Tree in tables using a commercial RDBMS. The table format is shown in Fig. 5d, where 'dst' denotes a destination vertex (or a destination cluster),'src' denotes a source vertex, and 'lef' denotes that the leaf node f whose node area $f.des$ contains the destination dst in $SP(src, 1)$. We illustrate the methods using Fig. 5. Since the cluster P_2 is in $v_2.des$ of $SP(v_0, 1)$, there is a row 'P_2, v_0, v_2' in table TAB_1SPT, which means that $v_2.dst$ contains P_2 in $SP(v_0, 1)$.

Pruning Power of SSP Pruning. We discuss the pruning power of SSP pruning. To facilitate analysis, we assume that all query vertices are selected independently. First, in Lemma 5, we discuss the probability that one vertex v can be pruned in DSG-query with n query vertices.

Lemma 5. *Given a vertex v in graph G, there are C leaf nodes in $SP(v,1)$. The number of vertices in each leaf node area $f_c.des$ is denoted as $|f_c.des|$, $c = 1...C$. Given a query set of $Q = \{q_i\}$, $i = 1...n$, the probability $Pr(v)$ that v is pruned can be evaluated by the following formula.*

$$Pr(v) = \frac{1}{|V(G)|^n} \sum_{i=1}^{C} |f_i.des|^n \tag{1}$$

Proof. If all query vertices are in one leaf node area $f_c.des$ of $SP(v,1)$, f_c strictly dominates v (Definition 5). Therefore, according to Algorithm 1 and SSP pruning, v can be filtered out safely. The probability that one query vertex q_i is in node area $f_c.des$ is $\frac{|f_c.des|}{|V(G)|}$. Since all query vertices are selected independently, the probability that all query vertices are in the same node area $f_c.des$ is $(\frac{|f_c.des|}{|V(G)|})^n$. Since there are C leaf nodes in $SP(v,1)$, we obtain:

$$Pr(v) = \frac{1}{|V(G)|^n} \sum_{i=1}^{C} |f_i.des|^n$$

Theorem 3. *Given a large graph G with $|V(G)|$ vertices, and a query set of $Q = \{q_i\}$, $i = 1...n$, the expected number of pruned vertices can be evaluated by the following formula.*

$$\sum_{i=1}^{|V(G)|} Pr(v_i) \tag{2}$$

where $Pr(v_i)$ is evaluated by Equation 1.

In the above analysis, we assume that all query vertices are selected independently. We evaluate Equation 2 in experiments (see Section 4) and show that the simple model can provide a good approximation of SSP's pruning power.

3.2 Computing Shortest Path Distance

As stated in Section 1, given a DSG query, we adopt filter-and-refine framework to find the answers. In the refinement phase, in order to avoid the expensive expansion in shortest-path algorithms, we can compute $Dist(v_j, q_i)$ directly based on 1-hop SP-Trees. The recursive algorithm (DisQ algorithm) shows the computation of $Dist(v,q)$. $DisQ$ is a recursive function. If the destination vertex q is in $SP(v,1)$, we can report $Dist(v,q)$ directly (Lines 2-3). Otherwise, if the leaf node area $f.des$ in $SP(v,1)$ contains q, we recursively call $DisQ(f,q)$ to compute $Dist(f,q)$ (Line 5). Finally, we report $Dist(v,q) = Dist(v,f) + Dist(f,q)$ (Line 6).

Algorithm 2. Shortest-Distance Query

```
$DisQ(v, q)$
Require: Input: $v$ and $q$: two vertices in graph $G$; $SP(v)$: all 1-Hop SP-Trees
    Output: $Dist(v, q)$: the shortest distance between the two vertices $v$ and $q$.
1: if vertex $q$ is a leaf node in $SP(v, 1)$ then
2:     Return $Dist(v, q)$
3: else
4:     There is leaf node $f$ in $SP(v, 1)$, and $f.des$ contains $q$.
5:     Call $DisQ(f, q)$ to compute $Dist(f, q)$
6:     Return $Dist(v, f)$+$Dist(f, q)$.
```

Theorem 4. *The time complexity of $DisQ(v, q)$ Algorithm (see Algorithm 2) in the worst case is $O(H)$, where H is the number of maximal hops between any two vertices in graph G.*

Proof. Algorithm 2 is a recursive algorithm. Since the number of maximal hops between any two vertices is H, we recursively call $DisQ(v, q)$ at most H times. In each iteration, the time complexity is $O(1)$. Therefore, the time complexity of Algorithm 2 is $O(H)$.

3.3 Putting It All Together: SSP Query Algorithm

We propose SSP-query algorithm in Algorithm 3, which calls SSP pruning algorithm (Algorithm 1) to obtain candidates in CL (Line 1). After that, Algorithm 2 is executed to compute $Dist(v, q)$ (Line 4). Skyline vertices are obtained by BNL algorithm [1] (Line 5). Finally, we report the final results RS (Line 6).

Algorithm 3. SSP Query Algorithm

```
Require: Input: $G$: a large graph; $Q$: a set of query vertices $Q = \{q_i\}$ Output: $RS$: the final
    skyline vertices.
1: call $SSP\_Prune(G, Q)$ (Algorithm 1) to obtain candidate set $CL$.
2: for each candidate vertex $v$ in $CL$ do
3:     for each query vertex $q_i$ do
4:         call $DisQ(v, q)$ (Algorithm 2) to compute $Dist(v, q)$.
5: perform BNL algorithm to find skyline vertices, and insert them into answer set $RS$.
6: Report $RS$.
```

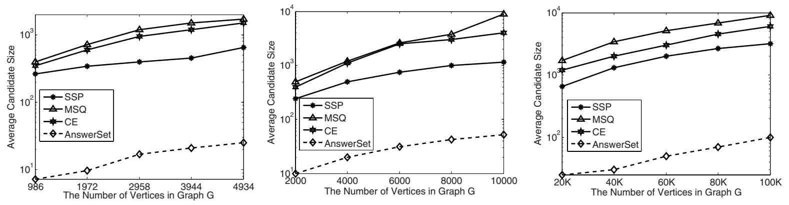
4 Experiments

In this section, we evaluate our methods over both real data sets and synthetic data sets. Although several efficient dynamic algorithms have been proposed, such as B^{2}S^2 and VS^2 [14] in spatial data and EDC and LBC algorithm [5] in road network, they cannot be applied to general graph data as discussed in Section 1. The two algorithms (EDC and

LBC algorithms) proposed in [5] are not applicable to a DSG-query, since they employ Euclidean distances as lower bounds of shortest path distances in a road network. There is no coordinate associated with each vertex, thus, it is impossible to employ Euclidean distances as lower bounds of shortest path distances in general graphs. Therefore, we exclude them from comparisons. We compare our methods with MSQ algorithm [3], since MSQ can work on any metric space and the distance function $Dist(v, q)$ in a graph is also a metric function. We also compare our algorithm with CE algorithm [14]. Although CE algorithm is proposed for skyline queries in road networks, it does not utilize special properties of road networks, suggesting that CE can handle dynamic skyline queries in general graphs. CE runs Dijkstra's algorithm from each query vertex in parallel until that at least one vertex is done by all query vertices. All un-visited vertices can be pruned safely. Furthermore, in the experiments, we also run linear scan for a DSG query as the straightforward approach, denoted as LS algorithm. Specifically, we first perform Dijkstra's algorithm [4] from each query vertex q_i to obtain $Dist(v, q_i)$ for each vertex v in graph G. After that, we perform BNL algorithm [1] to find results from all vertices in graph G. All experiments are implemented using standard C++ and conducted on a P4 1.7GHz machine with 1G RAM running Windows XP.

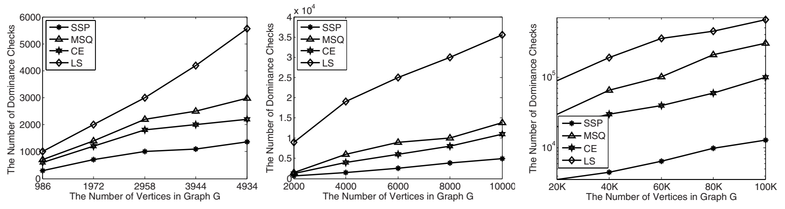
Data Sets: a) **S.cerevisiae Dataset**: This dataset (http://dip.doe-mbi.ucla.edu) is an undirected graph G in which each vertex represents a protein and each edge represents interactions between two proteins. There are 4934 vertices and 17346 edges in G. In experiments, we set all edge weights to be "1".
b) **DBLP Dataset**: This dataset is the well-known publication data from DBLP (dblp.uni-trier.de/xml/). We construct a co-author network G: every author is denoted as a vertex in G; and an edge is introduced when the corresponding two authors have at least one co-authored paper. We consider 100 important conferences in different areas to construct G. On the whole, there are about 100K vertices and about 400K edges in G. We also set all edge weights to be "1".
c) **Synthetic Dataset**: We use the graph generator $gengraph\_win$ (www.cs.sunysb.edu/∼algorith/implement/ viger/distrib/). In experiments, we generate a large graph G with 10K vertices satisfying power-law distribution. The edge weights in G satisfy random distribution between $[1, 1000]$. We denote the synthetic data set as $Powerlaw10K$.

We generate query sets by similar methods to those employed in previous studies [14,3]. We first randomly choose a vertex o in graph G, then retrieve $Max\{\lambda \times |V(G)|, n\}$ vertices in G that are the closest to o, and finally randomly select n vertices



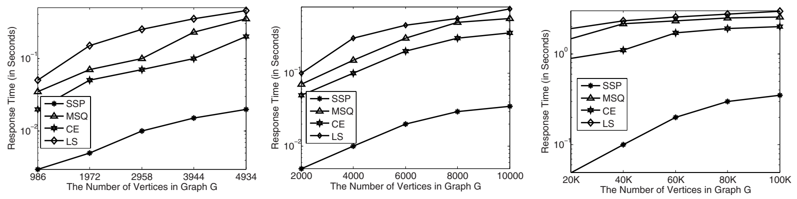
(a) S.cerevisiae (b) $Powerlaw10K$ (c) DBLP

Fig. 6. Candidate Size vs. Data Size



(a) S.cerevisiae (b) $Powerlaw10K$ (c) DBLP

Fig. 7. Number of Dominance Checks vs. Data Size



(a) S.cerevisiae (b) $Powerlaw10K$ (c) DBLP

Fig. 8. Response Time vs. Data Size

from them as query vertices. λ is a parameter within (0,1), $|V(G)|$ is the number of vertices in G, and n is the number of query vertices. Intuitively, large λ leads to a large diameter of query vertices in G. We evaluate query performance under different λ in Section 4.

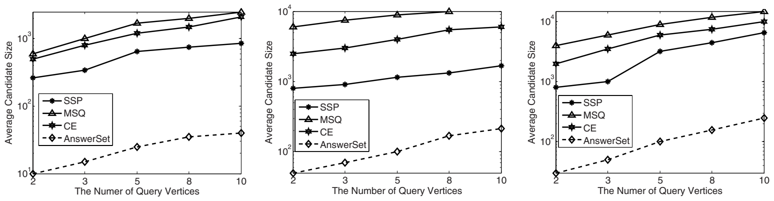
Query Performance vs. Data Sizes. In this subsection, we test our algorithm (denoted as SSP) under different data sizes, and compare it with MSQ and LS over both real datasets and synthetic datasets. In this set of experiments, we set $n = 5$, and $\lambda = 0.003$.

Fig. 6 shows the pruning power of different methods. SSP has the highest pruning power. Furthermore, the pruning power of SSP is stable in all datasets, and scales well with increasing data sizes. MSQ does not work well, especially in large graphs. This is because the pivots in graph G cannot provide the tight bound for the shortest-path distance $Dist(v, q)$. The candidate size in CE algorithm is larger than that in SSP-algorithm. In CE, we need to expand each query point in ascending order of their shortest path distance to this query point (in parallel). The expansion process stops when there exists at least one vertex that is visited by all query points. Therefore, as mentioned in [5], CE may result in many candidates and cause unnecessary shortest-path distance computation.

Fig. 7 illustrates the number of dominance checks in different methods during DSG-query. With increasing data size, the number of dominance checks also increases in all algorithms. SSP requires fewer dominance checks than other algorithms by orders of magnitude, which again confirms the superior efficiency of SSP.

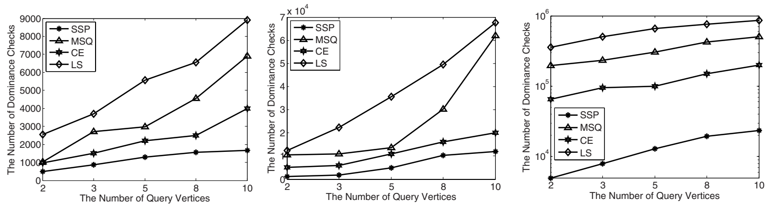
Fig. 8 shows the total response time in different methods. MSQ, CE and LS all need to perform Dijkstra's algorithm [4] from each query vertex q_i. The time complexity of Dijkstra's algorithm is $O(|V(G)|^2)$. In CE algorithm, we also need to expand each query vertex by Dijkstra's algorithm. The cost of running Dijkstra's algorithm constitutes the major portion of the response time. Figure 6 shows that $|CL|$ is about $\frac{1}{10}$ of $|V(G)|$. Thus, SSP's total response time outperforms MSQ, CE and LS by orders of magnitude, as observed in Figure 8.

Query Performance vs. Query Size. In this set of experiments, we evaluate SSP under different query sizes. Specifically, we set the number of query vertices $n = 2, 3, 5, 8, 10$, and $\lambda = 0.003$. As in traditional skylines, with increasing dimensions (i.e. the number of query vertices), the number of skyline vertices as well as the number of candidates grow. Fig. 9 shows that SSP still has the highest pruning power under different query sizes, consequently, the number of dominance checks is smaller than in other methods (Fig. 10). Figure 11 further shows that the total response time of SSP is the smallest among all methods under various query sizes.



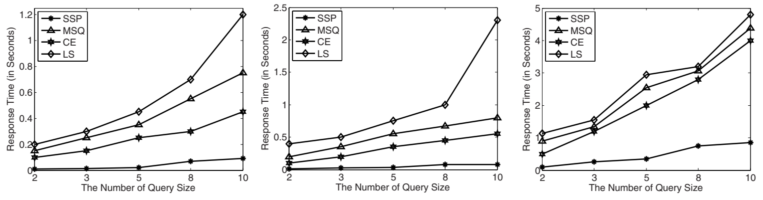
(a) S.cerevisiae (b) $Powerlaw10K$ (c) DBLP

Fig. 9. Candidate Size vs. Query Size



(a) S.cerevisiae (b) $Powerlaw10K$ (c) DBLP

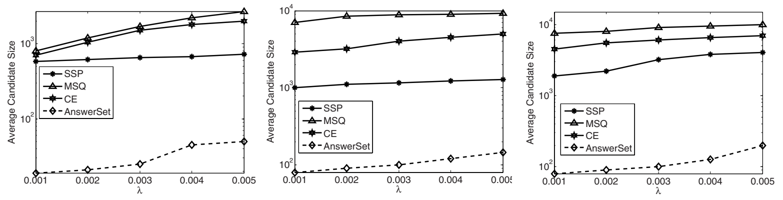
Fig. 10. Number of Dominance Checks vs. Query Size



(a) S.cerevisiae (b) $Powerlaw10K$ (c) DBLP

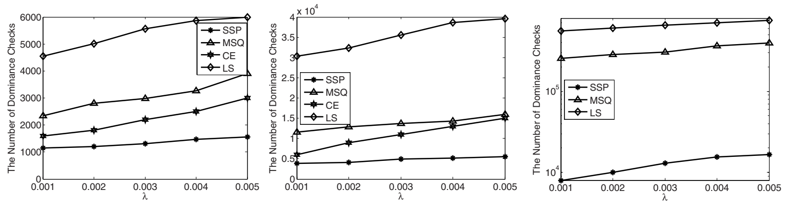
Fig. 11. Total Response Time vs. Query Size

Query Performance vs. Query Distribution. In this subsection, we test SSP under different query vertex distributions. Obviously, large λ means a large diameter of query vertices in G. In this set of experiments, we set $\lambda = 0.001, 0.002, 0.003, 0.004, 0.005$, and the number of query vertices $n = 5$. Fig. 12 shows that, with increasing query diameter, the cardinality of candidates in SSP also increases. This means that the pruning power of SSP decreases. The reason can be explained as follows: When query diameter is large, the probability that all query vertices are in one node area $f.des$ is small. Actually, MSQ's and CE's pruning powers also decrease when query diameter is large. However, the decrease in SSP's pruning power is smaller than MSQ's and CE's, as shown in Fig. 12. Fig. 13 and 14 show that the number of dominance checks and total response times increase with increasing of query diameter in both SSP and MSQ. This can be explained by the decreasing pruning power in Fig. 12.



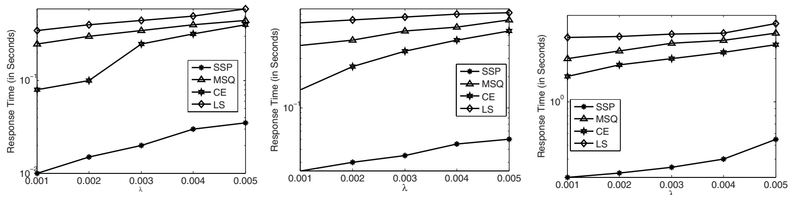
(a) S.cerevisiae (b) $Powerlaw10K$ (c) DBLP

Fig. 12. Candidate Size vs. λ



(a) S.cerevisiae (b) $Powerlaw10K$ (c) DBLP

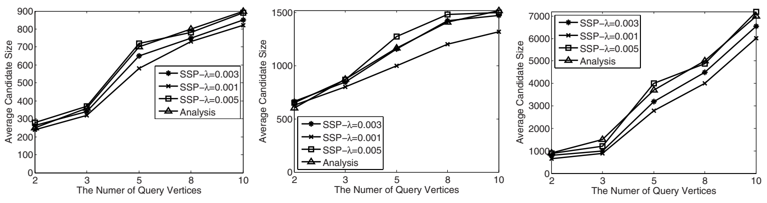
Fig. 13. Number of Dominance Checks vs. λ



(a) S.cerevisiae (b) $Powerlaw10K$ (c) DBLP

Fig. 14. Total Response Time vs. λ

Evaluating Pruning Power Analysis. We evaluate the pruning power analysis in Theorem 3 in Fig. 15. We use Equation 2 to compute the cardinality of pruned space ($Pruned$) under different query sizes. The theoretical candidate size is $|V(G)| - Pruned$. Fig. 15 shows that the theoretical candidate set size is a good approximation of the real candidate set size in SSP pruning, which confirms the effectiveness of our analysis about SSP pruning power in Theorem 3.



(a) S.cerevisiae (b) $Powerlaw10K$ (c) DBLP

Fig. 15. Evaluating Pruning Power Analysis in Equation 2

5 Related Work

Borzsonyi et al. have introduced the *skyline* operator [1], and have proposed block nested loops (BNL) and divide-and-conquer (D&C) algorithms [1] to execute them. Tan et al. [15] propose Bitmap and Index skyline processing algorithms. Kossmann et al. propose Nearest Neighbor (NN) method to process skyline queries progressively [9]. Papadias et al. introduce another efficient progressive algorithm named Branch-and-Bound Skyline (BBS) [11]. Recently, Lee et al. [10] propose a new method to answer skyline queries. The most related works to ours are dynamic skyline [11], spatial skyline [14], multi-source skyline on the road networks [5], and dynamic skyline queries in metric space [3]. Papadias et al. [11] first introduce dynamic skyline problems. Recently, Chen and Xiang have proposed MSQ algorithms for dynamic skyline problems [3], where the dimension functions can be any metric function. Although the shortest path distances in graphs that we use in our work is also a metric function, their methods are not optimized for large graphs. As the experimental results show, our methods outperform MSQ algorithm by orders of magnitude, in terms of both the number of dominance checks and online response time. Sharifzadeh and Shahabi [14] propose spatial skylines, where only Euclidean distances are considered as dimension functions. Thus, their pruning strategies cannot be applied into graph problems. Deng et al. [5] consider dynamic skylines in road networks, where $Dist(o, q_i)$ is defined as the shortest-path distance between o and q_i in the road network. However, their problem definition is different than ours, since each vertex in the graphs that they consider has a coordinate. Euclidean distance is used as lower bounds of shortest-path distances in a road network. Obviously, it is impossible to utilize the bounds in general graphs. Therefore, their pruning strategies cannot be applied to DSG-query.

Dijkstra's algorithm [4] is a classical single-source shortest-path algorithm, which extends graph G from source q until all vertices are reached, if graph G is connected. Given two vertices q and d, in order to answer shortest-path query (SP query for short), we can perform Dijkstra's algorithm from vertex q until vertex d is visited. However, it is inefficient to employ Dijkstra's algorithm to answer SP query in a large graph, since we have to visit a large number of vertices before we reach the desired destination vertex d. Therefore, materialization techniques should be applied to speed up online query. Lim and Chan [2] propose $DiskSP$ algorithm to answer SP queries. Based on graph partitions, they propose super-graph. Jing et al in [7] propose Hierarchical Encoding Path View (HEPV) for SP query. Another hierarchical graph model called HiTi is proposed by Jung and Pramanik [8]. Actually, any efficient SP-query algorithm can be utilized in the refinement process of DSG-query, which is orthogonal to our pruning strategies. There are a lot of work on spatial networks [13,12]. Generally speaking, these methods always utilize some spatial properties for processing. For example, Samet et al. [13] propose a best-first algorithm to find the k nearest neighbors in a spatial network. Data objects are indexed by quadtrees, which is a spatial indexing structure. For general graph problems, it is impossible to employ these spatial properties, such as spatial indexing, spatial coherence, Voronoi Diagrams and Euclidean distances, since vertices in general graphs have no coordinate. The main contributions of our work are that we only employ graph properties to develop pruning rules and process DSG-query.

6 Conclusions

In this paper, we propose dynamic skyline queries in graphs (DSG-query for short). For DSG-query, we propose a novel pruning strategy, that is shared shortest path (SSP) pruning. Based on SSP Pruning, we build careful-designed indexing structures. Extensive experiments confirm the effectiveness of our methods.

References

1. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE (2001)
2. Chan, E.P.F., Lim, H.: Optimization and evaluation of shortest path queries. VLDB J. 16(3) (2007)
3. Chen, L., Lian, X.: Dynamic skyline queries in metric spaces. In: EDBT (2008)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT Press, Cambridge (2001)
5. Deng, K., Zhou, X., Shen, H.T.: Multi-source skyline query processing in road networks. In: ICDE (2007)
6. Fuhry, D., Jin, R., Zhang, D.: Efficient skyline computation in metric space. TR-KSU-CS-2008-02, Department of Computer Science Kent State University (2008)
7. Jing, N., Huang, Y.-W., Rundensteiner, E.A.: Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. IEEE Trans. Knowl. Data Eng. 10(3) (1998)
8. Jung, S., Pramanik, S.: An efficient path computation model for hierarchically structured topographical road maps. IEEE Trans. Knowl. Data Eng. 14(5) (2002)
9. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: An online algorithm for skyline queries. In: VLDB (2002)
10. Lee, K.C.K., Zheng, B., Li, H., Lee, W.-C.: Approaching the skyline in z order. In: VLDB (2007)

11. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: SIGMOD (2003)
12. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: VLDB (2003)
13. Samet, H., Alborzi, J.S.H.: Scalable network distance browsing in spatial databases. In: SIGMOD (2008)
14. Sharifzadeh, M., Shahabi, C.: The spatial skyline queries. In: VLDB (2006)
15. Tan, K.-L., Eng, P.-K., Ooi, B.C.: Efficient progressive skyline computation. In: VLDB (2001)
16. Tong, H., Faloutsos, C.: Center-piece subgraphs: Problem definition and fast solutions. In: SIGKDD (2006)

Top-k Combinatorial Skyline Queries*

I-Fang Su[1], Yu-Chi Chung[2], and Chiang Lee[1]

[1] Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan, R.O.C.
emily@dblab.csie.ncku.edu.tw, leec@mail.ncku.edu.tw
[2] Department of Computer Science and Information Engineering, Chang Jung Christian University, Tainan, Taiwan, R.O.C.
justim@mail.cjcu.edu.tw

Abstract. The problem of top-k skyline computation has attracted considerable research attention in the past few years. Given a dataset, a top-k skyline returns k "most interesting" skyline tuples based on some kind of preference specified by the user. We extend the concept of top-k skyline to a so-called top-k combinatorial skyline query (k-CSQ). In contrast to the existing top-k skyline query (which is mainly to find the interesting skyline tuples), a k-CSQ is to find the interesting skyline tuples from various kinds of combinations of the given tuples. The k-CSQ is an important tool for areas such as decision making, market analysis, business planning, and quantitative economics research. In this paper, we will formally define this new problem, propose an intelligent method to resolve this problem, and also conduct a set of experiments to show the effectiveness and efficiency of the proposed algorithm.

1 Introduction

In this paper, we propose a novel variant of a skyline query called the *top-k combinatorial skyline query*. This type of query deals with data that are combinations of raw data. An example is in the field of stock investment. Figure 1 gives some stocks with the risk of investing in these stocks and their return. It is very likely that an investor will not invest all his money in just one stock, but in a *combination* of stocks which allows the investor to obtain a higher return and/or a lower risk. For example, investment in t_1 (i.e., Legg Mason) will render the highest return. However, this investment is also very risky. Are there any other stocks or combinations of stocks which allow us to have a higher return and/or a lower risk? These answers are often referred to as the *investment portfolio* [1] How to efficiently find such an investment portfolio is the issue studied in this paper.

Let us consider the combinations of two stocks only. We first define the new *risk* and new *return* of a combination of two stocks. One of the most oftenly used function in the estimation of profit gain is the *average function* [1, 2]. For

* This work is supported by National Science Council of Taiwan (ROC) under Grants NSC98-2221-E-006-148 and NSC98-2221-E-006-150.

Investment	Stocks	Risk(%)	Return (%)
t_1	Legg mason (LM)	18.02	9.59
t_2	Bank of New York	6.71	6.2
t_3	Alleghany	42.08	7.71
t_4	McDonalds	3.30	4.80
t_5	Deutsche Bank	31.01	6.8

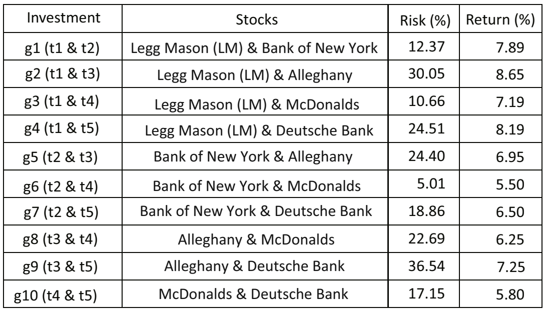
Fig. 1. Details of Stocks

example, Figure 2 gives all the combinations of two stocks in Figure 1. The *risk* and *return* of g_1 (which is combined from t_1 and t_2 as shown in the figure) are obtained from the average of those of t_1 and t_2, i.e., $\frac{18.02+6.71}{2}$=12.37 and $\frac{9.59+6.2}{2}$=7.89, respectively. If c stocks are used to form a combination (instead of two stocks), then *risk* and *return* can be generalized to $risk(g_i) = (risk(t_1)+risk(t_2)+\cdots+risk(t_c))/c$ and $return(g_i) = (return(t_1)+return(t_2)+\cdots+return(t_c))/c$, where g_i is a combination of stocks t_1, t_2, $\cdots$, t_c. A major property of such a function that is proper for defining the new *risk* and *return* is that the function should be monotone for proportion reasons [3, 4, 5]. Sum and average, for instances, are monotone functions and therefore oftenly used in these applications.

For convenience when there is no worry of confusion, we refer to a combination of stocks (such as each row in Figure 2) as a tuple too, just as referring to a single stock in Figure 1 as a tuple. We say that tuple A *dominates* tuple B if the values of the concerned attributes (such as *risk* and *return*) of A are not worse than those of B and at least one of them is better [3, 6, 7, 8, 9]. The problem of finding the portfolio is quite complicated because we need to search all single stocks, all combinations of two stocks, all combinations of three stocks, up to all combinations of, say, c stocks for the dominating tuples so as to form the portfolio, where c is given by the user. We call such a query the *Combinatorial Skyline Query* (*CSQ*). If only the top-k dominating tuples are returned to the user where the sorting is based on a *preference order* $\mathcal{P}$ specified by the user, then the query is a *Top-K Combinatorial Skyline Query* ($k-CSQ$).

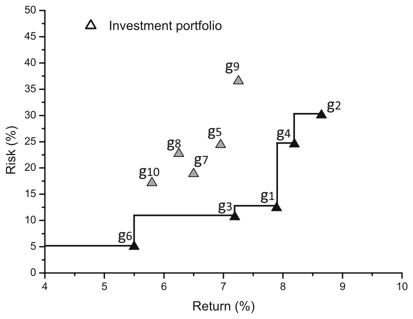
For a table of n tuples, the number of possible combinations where each has a cardinality of c (i.e., each combination is composed of c stocks) would be $\sum_{i=1}^{c}\binom{n}{i}$. If the number of stocks n is 1000 and the user wants to find the portfolio with a cardinality not greater than 5, the search space of a combinatorial skyline would consist $\sum_{i=1}^{5}\binom{1000}{i} \approx 10^{12}$ combinations. This number increases exponentially with the number of stocks and the cardinality. An extremely high computation cost is incurred unavoidably in resolving such a query.

In this paper, we propose an algorithm named the *Restricted Construction Algorithm* (RCA) to resolve the top-k combinatorial skyline query problem. RCA progressively composes the combinations and prunes the combinations that are impossible to be the combinatorial skyline result, and then uses a filtering



Investment	Stocks	Risk (%)	Return (%)
g1 (t1 & t2)	Legg Mason (LM) & Bank of New York	12.37	7.89
g2 (t1 & t3)	Legg Mason (LM) & Alleghany	30.05	8.65
g3 (t1 & t4)	Legg Mason (LM) & McDonalds	10.66	7.19
g4 (t1 & t5)	Legg Mason (LM) & Deutsche Bank	24.51	8.19
g5 (t2 & t3)	Bank of New York & Alleghany	24.40	6.95
g6 (t2 & t4)	Bank of New York & McDonalds	5.01	5.50
g7 (t2 & t5)	Bank of New York & Deutsche Bank	18.86	6.50
g8 (t3 & t4)	Alleghany & McDonalds	22.69	6.25
g9 (t3 & t5)	Alleghany & Deutsche Bank	36.54	7.25
g10 (t4 & t5)	McDonalds & Deutsche Bank	17.15	5.80

(a) Details of investment portfolios.



(b) Skyline of investment portfolios.

Fig. 2. An example of an investment portfolio

mechanism to retrieve the top-k combinatorial skyline tuples without enumerating all combinations. Major contributions of this work include (1) a new problem, the combinatorial skyline query, is identified and investigated, (2) an efficient algorithm is proposed for processing a top-k combinatorial skyline query, (3) the correctness of the answer is guaranteed, (4) experimental result shows that the proposed method is very promising.

The rest of the paper is organized as follows. First, the problem is formulated in Section 2. The core techniques comprising our solution are presented in Section 3. Section 4 presents our experimental evaluation of the proposed solution. Finally, we conclude the article and describe directions for future work in Section 5.

2 Problem Formulation

Let D be a d-dimensional dataset and t_i be the i-th tuple of D. We use $t_i.s_k$ to denote the k-th dimensional value of t_i such that t_i can be represented as $t_i =< t_i.s_1, t_i.s_2, \cdots, t_i.s_d >$. Without loss of generality, we consider the MAX function [9, 10] as the skyline operator throughout this paper.

Definition 1. t_i ***dominates*** t_j
A tuple t_i *is said to dominate another tuple* t_j *if and only if* $t_i.s_k \geq t_j.s_k$ *for* $1 \leq k \leq d$ *and there exists at least one dimension* ℓ $(1 \leq \ell \leq d)$ *such that* $t_i.s_\ell > t_j.s_\ell$*. We use* $t_i \succ t_j$ *to represent that* t_i *dominates* t_j*.*

Definition 2. ***Skyline***
A tuple t_i *is a skyline of* D *if and only if there does not exist a tuple* t_j $(t_j \in D)$ *such that* $t_j \succ t_i$*.*

In the following, we first introduce the concept of a *combination*, and then use it to define a combinatorial skyline.

Definition 3. ***Combination*** g_p
Given a tuple set $p = \{t_{p1}, t_{p2}, \cdots, t_{pk}\}$ *and combinatorial functions* $\mathcal{F} = \{f_1, f_2, \cdots, f_d\}$*, where* t_{pi} *is the* i*-th element of the tuple set* p *and* f_i *is a combinatorial function in* $\mathcal{F}$ $(1 \leq i \leq d)$*.* f_i *maps multiple parameters* $t_{p1}.s_i, t_{p2}.s_i, \cdots, t_{pk}.s_i$ *to a scaler. A combination, denoted as* g_p*, is expressed as* $g_p = [t_{p1}, t_{p2}, \cdots, t_{pk}] = \ll f_1(t_{p1}.s_1, t_{p2}.s_1, \cdots, t_{pk}.s_1), f_2(t_{p1}.s_2, t_{p2}.s_2, \cdots, t_{pk}.s_2), \cdots, f_d(t_{p1}.s_d, t_{p2}.s_d, \cdots, t_{pk}.s_d) \gg$ *where* $[t_{p1}, t_{p2}, \cdots, t_{pk}]$ *means that the combination is obtained from tuples* $t_{p1}, t_{p2}, \cdots, t_{pk}$*.* $f_i(t_{p1}.s_i, t_{p2}.s_i, \cdots, t_{pk}.s_i)$ *is a scalar which is the* i*-th attribute value of the combination* g_p*.* $|g_p|$ *is the cardinality of* g_p*.*

Notice that conceptually g_p is itself a d-dimensional tuple, although it is obtained from a transformation of several other d-dimensional tuples. In order to distinguish a combination from a lower level tuple, we use the notation $\ll\gg$ to represent a combination and $<>$ to represent a tuple. For convenience, we also call $f_i(t_{p1}.s_i, t_{p2}.s_i, \cdots, t_{pk}.s_i)$ the i-th *attribute* of combination g_p and refer to $f_i(t_{p1}.s_i, t_{p2}.s_i, \cdots, t_{pk}.s_i)$ as $g_p.f_i(s_i)$ for better readability. Thus, can be rewritten as $g_p = \ll f_1(t_{p1}.s_1, t_{p2}.s_1, \cdots, t_{pk}.s_1), f_2(t_{p1}.s_2, t_{p2}.s_2, \cdots, t_{pk}.s_2), \cdots, f_d(t_{p1}.s_d, t_{p2}.s_d, \cdots, t_{pk}.s_d) \gg = \ll g_p.f_1(s_1), g_p.f_2(s_2), \cdots, g_p.f_d(s_d) \gg$.

Following our example in Figure 2(a), let $g_{10} = [t_4, t_5]$ be a combination, and $\mathcal{F} = \{AVG, AVG\}$. Note that, $t_4 =< 3.30, 4.80 >$ and $t_5 =< 31.01, 6.8 >$ respectively. Then, we have $g_{10} = [t_4, t_5] = \ll f_1(t_4.s_1, t_5.s_1), f_2(t_4.s_2, t_5.s_2) \gg = \ll AVG(3.30, 31.01), AVG(4.80, 6.8) \gg = \ll 17.15, 5.80 \gg$.

As a combination is itself a d-dimensional tuple, the notion of dominance can be unaffectedly applied to it.

Definition 4. ***Combinatorial skyline***
A combination g_p *is a combinatorial skyline tuple if and only if there does not exist another combination* g_q *such that* $g_q \succ g_p$*.*

Definition 5. ***Rank of a Combinatorial Skyline***
Let $S = \{s_1, s_2, \cdots, s_d\}$ *be the set of dimensions such that* s_i *is more preferred by the user than* s_j *if* $i < j$*. Also let* g_p *and* g_q *be two combinatorial skyline tuples.* g_p *has a higher rank than* g_q *(denoted as* $g_p \twoheadrightarrow g_q$*), if (1)*$g_p.f_1(s_1) > g_q.f_1(s_1)$ *or (2)* $g_p.f_i(s_i) = g_q.f_i(s_i)$*, for* $i = 1, 2, \cdots, l$ $(l < d)$ *and* $g_p.f_{l+1}(s_{l+1}) > g_q.f_{l+1}(s_{l+1})$*.*

Note that if $g_p \twoheadrightarrow g_q$, then g_q cannot dominate g_p. The reason is that we are sure $\exists i$ such that $g_p.f_i(s_i) > g_q.f_i(s_i)$.

Definition 6. ***top-k Combinatorial Skyline Query*** *k-CSQ$(D, \mathcal{F}, c, \mathcal{P})$*
Let k-CSQ$(D, \mathcal{F}, c, \mathcal{P})$ be a top-k combinatorial skyline query, where D is a d-dimensional database, $\mathcal{F}$ is a set of combinatorial functions, $c \in Z^+$, k is given by the user which specifies the first k combinatorial skyline tuples that satisfy the query, and $\mathcal{P}$ is the preference order of attributes that is also given by the user. The result of k-CSQ$(D, \mathcal{F}, c, \mathcal{P})$ is a set $G = \{g_1, g_2, \cdots, g_k\}$ of combinatorial skyline tuples such that $g_i \twoheadrightarrow g_j$ if $i < j$ (where $1 < i, j < k$).

In the above example, suppose that t_1, t_2, t_4, g_1, g_3, and g_6 are combinatorial skyline tuples, and the user preference order is $\mathcal{P} = \{s_2, s_1\}$. Then, $t_1 \twoheadrightarrow g_1 \twoheadrightarrow g_3 \twoheadrightarrow t_2 \twoheadrightarrow g_6 \twoheadrightarrow t_4$. Therefore, t_1 is the top-1 combinatorial skyline.

3 Top-k Combinatorial Skyline Query Processing

$k = 1$ is a special case of this type of queries. Its answer is easy to find. For example in Figure 3, assume that the query is to find the top-1 combinatorial skyline. Let $c = 3$, $\mathcal{F} = \{SUM, SUM\}$, and $\mathcal{P} = \{s_1, s_2\}$. We sort tuples in Figure 3 according to $\mathcal{P}$ and retrieve the top three tuples (i.e., t_1, t_2, t_3) to construct a combination (i.e., $[t_1, t_2, t_3]$). $[t_1, t_2, t_3]$ is definitely the top-1 combinatorial skyline tuple, because the values of the user preferred attribute of t_1, t_2, and t_3 are the top three among all tuples. Hence, the answer of a top-1 CSQ is obvious. However, if the number of k is greater than 1, the search of the answer will be nontrivial. We also use Figure 3 to explain the complication of the problem.

ID	s_1	s_2
t_1	10	6
t_2	9	11
t_3	8	9
t_4	7	7
t_5	4	18

Fig. 3. The data set of a running example

Assume that the query is simply to find the top-2 combinatorial skyline tuples and other conditions remain the same. $[t_1, t_2, t_3]$ is still ranked first in all combinatorial skyline tuples. One might think that $[t_1, t_2, t_4]$ should be the second combinatorial skyline tuple since t_4 is next to t_3. But in fact $[t_1, t_2, t_4] = \ll 26, 24 \gg$ is dominated by $[t_1, t_2, t_3] = \ll 27, 26 \gg$. Hence, $[t_1, t_2, t_4]$ cannot even be a combinatorial skyline tuple, let alone to become a top-2 combinatorial skyline tuple. The next to consider is $[t_1, t_2, t_5]$. $[t_1, t_2, t_5] = \ll 23, 35 \gg$ is not dominated by $[t_1, t_2, t_3] = \ll 27, 26 \gg$. However, it is too early to say that it is the second combinatorial skyline tuple, because there may be other combinations better than it, e.g., $[t_2, t_3, t_4]$. In this example, $[t_2, t_3, t_4] = \ll 24, 27 \gg$ is more preferred than $[t_1, t_2, t_5] = \ll 23, 35 \gg$ as the order in s_1 is more important than in s_2. Hence, $[t_2, t_3, t_4]$ will be ranked the second place in the combinatorial skyline tuples. This example indicates the complication factors of the problem. In fact, the number of combinations to compare increases exponentially with k. Hence, numerous computation will be required.

3.1 The Brute-Force Method

The brute-force method is to process a k-CSQ ($k \geq 2$) in a most straightforward manner. First, we compose all possible combinations from tuples in D. Then, we retrieve a set of combinatorial skyline tuples from all combinations and sort the combinatorial skyline tuples according to $\mathcal{P}$. Finally, the top-k combinatorial skyline tuples with the highest user preference are returned to the user. As the idea is simple and clear, we omit the details.

3.2 Restricted Construction Algorithm (RCA)

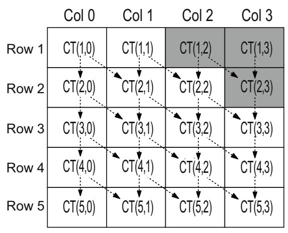
We first take a look in concept how to generate all possible combinations. This can be achieved by utilizing the concept of *binomial coefficients* $\binom{n}{c}$ [11], where $\binom{n}{c}$ is the number of ways that c tuples can be chosen from among n objects regardless of the order and $1 \leq c \leq n$. This property is often described recursively as $\binom{n}{c} = \binom{n-1}{c} + \binom{n-1}{c-1}$. We refer to *c-combination* as a combination of a cardinality of c, and $CT(n, c)$ as the set that contains all *c-combinations* of a dataset with n tuples $\{t_1, t_2, \cdots, t_n\}$. For example, $\{t_1, t_2, t_3, t_4\}$ is a dataset of four tuples, and $CT(4, 2) = \{[t_1, t_2], [t_1, t_3], [t_1, t_4], [t_2, t_3], [t_2, t_4]$, and $[t_3, t_4]\}$. Also we define $\oplus$ as an adding operator such that $S \oplus t_i$ means to add t_i to each element of the set S. For example, $\{[t_1, t_2], [t_1, t_3]\} \oplus t_5 = \{[t_1, t_2, t_5], [t_1, t_3, t_5]\}$. For another example, $CT(4, 1) \oplus t_5 = \{[t_1, t_5], [t_2, t_5], [t_3, t_5], [t_4, t_5]\}$.

By applying this to the binomial coefficients mentioned above, we have

$$CT(n, c) = CT(n-1, c) \cup (CT(n-1, c-1) \oplus t_n), \tag{1}$$

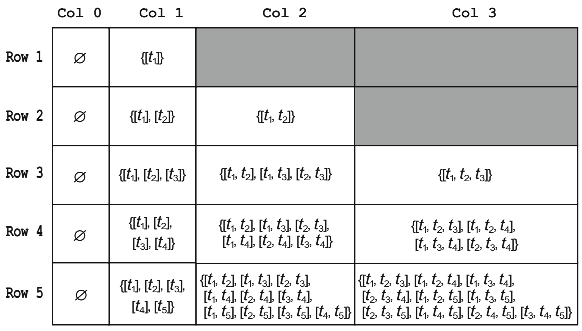
where $1 \leq c \leq n$ and $\cup$ is the set union operator. $CT(n, c)$ is an empty set if $c = 0$ or $c > n$. Equation 1 gives all c-combinations of dataset $D = \{t_1, t_2, \cdots, t_n\}$, where each of the combinations is either a combination that contains t_n (i.e., $CT(n-1, c-1) \oplus t_n$) or one that does not (i.e., $CT(n-1, c)$).

Equation 1 tells us that if $CT(n-1, c)$ and $CT(n-1, c-1)$ are derived, then we can utilize them to construct $CT(n, c)$. This can, in turn, be applied to obtaining $CT(n-1, c-1)$ if $CT(n-2, c-2)$ and $CT(n-2, c-1)$ are known. Following this, the whole problem can be divided into a number of subproblems and the result of each subproblem can be applied to resolving the subproblem of the next stage. We use a table $\mathcal{T}$, called the *solution table*, to store the solution of each subproblem. Figure 4 shows the solution table while processing $CT(5, 3)$. $\mathcal{T}$ can be thought of as a data structure of a two-dimensional array, consisting of n rows and $(c+1)$ columns if there are n tuples in D and the cardinality is c. In Figure 4, $\mathcal{T}$ has 5 rows and 4 columns. An arrow pointing from $CT(h, k)$ to $CT(i, j)$ means that the solution of $CT(h, k)$ is used to construct the solution of $CT(i, j)$. For instance, $CT(3, 2) = CT(2, 2) \cup (CT(2, 1) \oplus t_3)$. Hence, there are arrows pointing from $CT(2, 2)$ and $CT(2, 1)$ to $CT(3, 2)$. Note that Col 0 is also listed in Figure 4. Although all the entries in this column (i.e., $CT(i, 0)$) are empty sets, they can be combined with $CT(i, 1)$ to generate meaningful combinations (i.e., $CT(i+1, 1)$). Hence, they cannot be excluded from the table. Also note that there are three shadowed cells in the solution table. Each of them is also an empty set because $c > n$ makes $CT(n, c)$ meaningless. The complete solution table of this particular example is shown in Figure 5.



	Col 0	Col 1	Col 2	Col 3
Row 1	CT(1,0)	CT(1,1)	CT(1,2)	CT(1,3)
Row 2	CT(2,0)	CT(2,1)	CT(2,2)	CT(2,3)
Row 3	CT(3,0)	CT(3,1)	CT(3,2)	CT(3,3)
Row 4	CT(4,0)	CT(4,1)	CT(4,2)	CT(4,3)
Row 5	CT(5,0)	CT(5,1)	CT(5,2)	CT(5,3)

Fig. 4. The solution table of $CT(5,3)$



	Col 0	Col 1	Col 2	Col 3
Row 1	$\varnothing$	$\{[t_1]\}$	▨	▨
Row 2	$\varnothing$	$\{[t_1], [t_2]\}$	$\{[t_1, t_2]\}$	▨
Row 3	$\varnothing$	$\{[t_1], [t_2], [t_3]\}$	$\{[t_1, t_2], [t_1, t_3], [t_2, t_3]\}$	$\{[t_1, t_2, t_3]\}$
Row 4	$\varnothing$	$\{[t_1], [t_2], [t_3], [t_4]\}$	$\{[t_1, t_2], [t_1, t_3], [t_2, t_3], [t_1, t_4], [t_2, t_4], [t_3, t_4]\}$	$\{[t_1, t_2, t_3], [t_1, t_2, t_4], [t_1, t_3, t_4], [t_2, t_3, t_4]\}$
Row 5	$\varnothing$	$\{[t_1], [t_2], [t_3], [t_4], [t_5]\}$	$\{[t_1, t_2], [t_1, t_3], [t_2, t_3], [t_1, t_4], [t_2, t_4], [t_3, t_4], [t_1, t_5], [t_2, t_5], [t_3, t_5], [t_4, t_5]\}$	$\{[t_1, t_2, t_3], [t_1, t_2, t_4], [t_1, t_3, t_4], [t_2, t_3, t_4], [t_1, t_2, t_5], [t_1, t_3, t_5], [t_2, t_3, t_5], [t_1, t_4, t_5], [t_2, t_4, t_5], [t_3, t_4, t_5]\}$

Fig. 5. The final solution table $\mathcal{T}$

The combinations shown in a solution table include all possible combinations of $CT(n, c)$. In the following, we propose a mechanism to find the top-k combinatorial skyline without enumerating all these combinations. We will use a concept, called *descendant*, in the process of pruning the unnecessary comparisons on the combinations. Given a combination g'_p, we say that g'_p is a *descendant* of g_p if g'_p is (1) g_p itself or (2) obtained by combining g_p with other tuple(s). Assume that g_q is a combination and $g_q \twoheadrightarrow g'_p$, which means g_q is more preferred than g'_p. This indicates that $\exists i$ such that $g'_p.f_i(s_i) < g_q.f_i(s_i)$ (i.e., the i-th attribute value of g'_p is less than that of g_p), and therefore g'_p cannot "dominate" g_q.

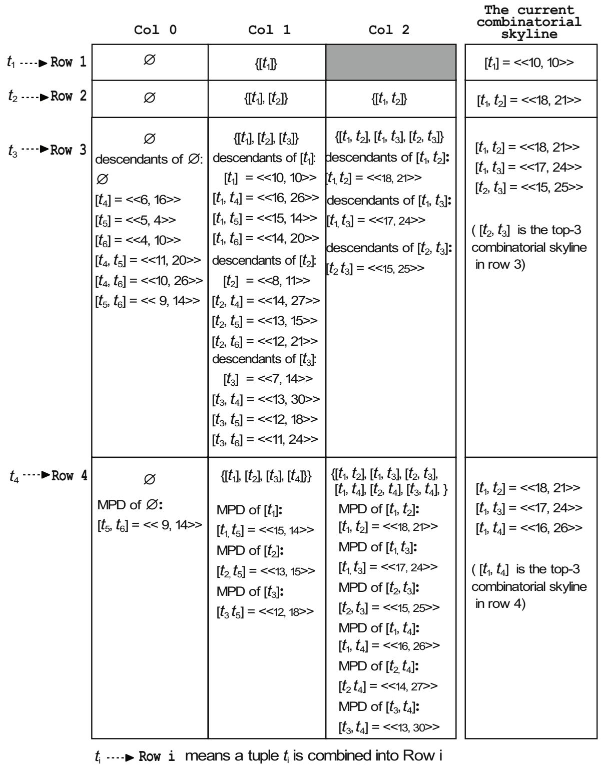
Now, we use an example to illustrate the process of our method. Assume that there are six tuples in D (referring to Figure 6). A user needs to find the top-3 combinatorial skyline tuples with $c = 2$ and user preference $\mathcal{P} = \{s_1, s_2\}$. Our method is to first sort the tuples in D according to $\mathcal{P}$. Thus, the order is t_1, t_2, t_3, t_4, t_5, and t_6, as shown in Figure 6. In Figure 7, the first step (Row 1) is to pick t_1 to construct the solutions in row 1. Thus, we have two combinations, ϕ and $[t_1]$. Next, we do the dominance test to find the current combinatorial skyline in this row. We find that $[t_1]$ is the only combinatorial skyline tuple in this row. Since the user is looking for top-3 combinatorial skyline tuples, the number of currently found top combinatorial skyline tuples is less than three. We proceed to the next row.

In Row 2, t_2 is added to construct the solutions in this row, meaning that four combinations ϕ, $[t_1]$, $[t_2]$, and $[t_1, t_2]$ should be considered. We do the dominance test to find the combinatorial skyline in this row. $[t_1, t_2]$ is the only combinatorial skyline tuple found in this row. Since the number of combinatorial skyline tuples found up to now is still less than required, we proceed to the next row.

In Row 3, t_3 is added to construct the solutions. This time, six combinations, ϕ, $[t_1]$, $[t_2]$, $[t_3]$, $[t_1, t_2]$, $[t_1, t_3]$, and $[t_2, t_3]$ are generated. We also do the dominance test and find that $[t_1, t_2]$, $[t_1, t_3]$, and $[t_2, t_3]$ are the top-3 combinatorial skyline tuples, in which $[t_2, t_3]$ is in the third place of the top-3 combinatorial skyline tuples. The next thing is to check whether the current top-3 combinatorial skyline tuples are the final results. In other words, we need to be sure that the current top-3 combinatorial skyline tuples will not be dominated by or less

ID	s_1	s_2
t_1	10	10
t_2	8	11
t_3	7	14
t_4	6	16
t_5	5	4
t_6	4	10

Fig. 6. The data set of a running example



	Col 0	Col 1	Col 2	The current combinatorial skyline
t_1 ----▸ Row 1	∅	{[t_1]}	▨	[t_1] = <<10, 10>>
t_2 ----▸ Row 2	∅	{[t_1], [t_2]}	{[t_1, t_2]}	[t_1, t_2] = <<18, 21>>
t_3 ----▸ Row 3	∅ descendants of ∅: ∅ [t_4] = <<6, 16>> [t_5] = <<5, 4>> [t_6] = <<4, 10>> [t_4, t_5] = <<11, 20>> [t_4, t_6] = <<10, 26>> [t_5, t_6] = << 9, 14>>	{[t_1], [t_2], [t_3]} descendants of [t_1]: [t_1] = <<10, 10>> [t_1, t_4] = <<16, 26>> [t_1, t_5] = <<15, 14>> [t_1, t_6] = <<14, 20>> descendants of [t_2]: [t_2] = <<8, 11>> [t_2, t_4] = <<14, 27>> [t_2, t_5] = <<13, 15>> [t_2, t_6] = <<12, 21>> descendants of [t_3]: [t_3] = <<7, 14>> [t_3, t_4] = <<13, 30>> [t_3, t_5] = <<12, 18>> [t_3, t_6] = <<11, 24>>	{[t_1, t_2], [t_1, t_3], [t_2, t_3]} descendants of [t_1, t_2]: [t_1, t_2] = <<18, 21>> descendants of [t_1, t_3]: [t_1, t_3] = <<17, 24>> descendants of [t_2, t_3]: [t_2 t_3] = <<15, 25>>	[t_1, t_2] = <<18, 21>> [t_1, t_3] = <<17, 24>> [t_2, t_3] = <<15, 25>> ([t_2, t_3] is the top-3 combinatorial skyline in row 3)
t_4 ----▸ Row 4	∅ MPD of ∅: [t_5, t_6] = << 9, 14>>	{[t_1], [t_2], [t_3], [t_4]} MPD of [t_1]: [t_1 t_5] = <<15, 14>> MPD of [t_2]: [t_2 t_5] = <<13, 15>> MPD of [t_3]: [t_3 t_5] = <<12, 18>>	{[t_1, t_2], [t_1, t_3], [t_2, t_3], [t_1, t_4], [t_2, t_4], [t_3, t_4], } MPD of [t_1, t_2]: [t_1, t_2] = <<18, 21>> MPD of [t_1 t_3]: [t_1 t_3] = <<17, 24>> MPD of [t_2, t_3]: [t_2 t_3] = <<15, 25>> MPD of [t_1, t_4]: [t_1, t_4] = <<16, 26>> MPD of [t_2 t_4]: [t_2 t_4] = <<14, 27>> MPD of [t_3, t_4]: [t_3, t_4] = <<13, 30>>	[t_1, t_2] = <<18, 21>> [t_1, t_3] = <<17, 24>> [t_1, t_4] = <<16, 26>> ([t_1, t_4] is the top-3 combinatorial skyline in row 4)

t_i ----▸ Row i means a tuple t_i is combined into Row i

Fig. 7. The example of how we use MPD in the method

preferred than any other combination that is formed in future steps. Now our method is to construct the descendants of each combination in this row. For example in Col 1 Row 3, the descendants of $[t_1]$ are (1) $[t_1]$ itself, (2) $[t_1]$ combined with one of the remaining tuples in D which means $[t_1, t_4], [t_1, t_5]$, and $[t_1, t_6]$. Note that the remaining tuples here are tuples left in D for further processing in the following rows. For example in terms of Row 1, the remaining tuples in D are t_2, t_3, t_4, t_5, and t_6. And in terms of Row 2, the remaining tuples in D are t_3, t_4, t_5, and t_6. The descendants of ϕ in row 3 are (1) ϕ itself, (2) ϕ combined with one of the remaining tuples in D which gives $[t_4]$, $[t_5]$, and $[t_6]$, and (3) ϕ combined with two of the remaining tuples in D which are $[t_4, t_5]$, $[t_4, t_6]$, and $[t_5, t_6]$. Then, we compare the preference of the k-th (i.e., third in this example) combinatorial skyline, $[t_2, t_3]$, with that of each of these descendants. We find that $[t_1, t_4] \twoheadrightarrow [t_2, t_3]$ ($\ll 16, 26 \gg \twoheadrightarrow \ll 15, 25 \gg$). This means that one combination which is to be constructed in a future row will dominate $[t_2, t_3]$. Hence, the current top-3 combinatorial skyline tuples cannot be the final result. Therefore, the process should continue.

Now, the problem is "can this process stop as early as possible so that we do not need figure out all descendants of each combination?" We find an elegant and efficient way to accelerate the processing of finding the top-k combinatorial skyline tuples. Instead of comparing the k-th combinatorial skyline to each descendant, we compare the k-th combinatorial skyline to the *most preferred descendant* (MPD) of each combination. The MPD of a combination g_p (denoted as $MPD(g_p)$) is a descendant of g_p and MPD(g_p) has the highest user preference among all g_p's descendants. In other words, the user preference of any g_p's descendant is always equal to or lower than that of the MPD(g_p). Let us go back to Row 3 of the example in Figure 7. Since $[t_1, t_4]$ has the highest user preference among all descendants of $[t_1]$, MPD($[t_1]$) is $[t_1, t_4]$. Following this idea, we can find the MPD of each combination (i.e., MPD(ϕ) = $[t_4, t_5]$, MPD($[t_1]$) = $[t_1, t_4]$, MPD($[t_2]$) = $[t_2, t_4]$, MPD($[t_3]$) = $[t_3, t_4]$,, MPD($[t_1, t_2]$) = $[t_1, t_2]$, MPD($[t_1, t_3]$) = $[t_1, t_3]$, and MPD($[t_2, t_3]$) = $[t_2, t_3]$). Then, we compare the preference of the k-th combinatorial skyline ($[t_2, t_3]$) with the preference of these MPDs. In this example, since the preference of MPD($[t_1]$) is higher than that of $[t_2, t_3]$ (i.e., $\ll 16, 26 \gg \twoheadrightarrow \ll 15, 25 \gg$, we know that the current top-k combinatorial skyline tuples are not the final results. The algorithm will proceed to the next row.

As the number of MPDs is much less than the number of all descendants, comparing the k-th combinatorial skyline tuple with these MPDs is therefore much faster and more efficient than comparing with the descendants. However, locating the MPD of g_p requires to construct all descendants of g_p, which is still a very costly process. In the following, we propose a method to find the MPD of a combination without having to enumerate all its descendants. Given a combination g_p, where $|g_p| = u \leq c$, and a set of remaining tuples $D' = \{t_j, t_{j+1}, \cdots, t_n\}$. D' is sorted according to the user preference $\mathcal{P}$ which means that $t_j \twoheadrightarrow t_{j+1}$, $j = 1, 2, \cdots, |D'| - 1$. Let g_p^{u+i} be a descendant of g_p that is obtained by combining g_p with the first i tuples in D', where i =0, 1, $\cdots$, $(c-u)$. g_p^{u+i} has the following interesting properties: (1) g_p^{u+i} is a descendant of g_p, (2) the cardinality of g_p^{u+i} is $u + i$, and (3) g_p^{u+i} is the most preferred combination among the descendants of g_p whose cardinality is $u + i$ (the third property is to be proved in Lemma 1). The following equation can be used to determine $MPD(g_p)$.

$$MPD(g_p) = MAX(g_p^{u+0}, g_p^{u+1}, \cdots, g_p^{u+(c-u)}). \tag{2}$$

For example in Row 3 of Figure 7, the remaining tuples which are sorted according to $\mathcal{P}$ are t_4, t_5, and t_6, respectively. The MPD($[t_1]$) is MAX($[t_1]^{1+0}$,$[t_1]^{1+(2-1)}$). Then we can have MPD($[t_1]$) = MAX($[t_1]$,$[t_1, t_4]$). Since $[t_1, t_4]$=$\ll 16, 26 \gg$ is greater than $[t_1]$=$\ll 10, 10 \gg$, MPD($[t_1]$) = $[t_1, t_4]$. Note that Equation 2 is also applicable to empty set ϕ to find its MPD. For example in Col 0 and in Row 3 of Figure 7, $g_p = \phi$. We have g_p^{u+0}, g_p^{u+1}, and g_p^{u+2} and they are ϕ, $[t_4]$, and $[t_4, t_5]$, respectively. By Equation 2, the MPD(ϕ) = MAX(ϕ, $[t_4]$, $[t_4, t_5]$) = $[t_4, t_5]$.

Notice that using Equation 2, the MPD of any combination can be determined by using only one computation. Constructing all descendants of this combination is no longer required. This dramatically reduces the computation time of finding the top-k combinatorial skyline tuples.

Lemma 1. *Let t_i and t_j be two tuples in D. $g_p = [t_{p1}, t_{p2}, \cdots, t_{pc}]$ is a combination that do not contain t_i and t_j (i.e., t_i, $t_j \neq t_{px}$ for $x = 1, 2, \cdots, c$). If $t_i \twoheadrightarrow t_j$, then the user preference of g_p combined with t_i is higher than that of g_p combined with t_j. That is, $t_i \twoheadrightarrow t_j \Rightarrow [t_{p1}, t_{p2}, \cdots, t_{pc}, t_i] \twoheadrightarrow [t_{p1}, t_{p2}, \cdots, t_{pc}, t_j]$.*

Theorem 1. *If g'_p is a combination that is derived from Equation 2, then g'_p is the MPD of combination g_p (i.e., $MPD(g_p) = g'_p$). That is, there does not exist a descendant g''_p of g_p such that $g''_p \twoheadrightarrow g'_p$.*

Based on Lemma 1 and Theorem 1, we learn that none of the descendants of combination g_p has a higher preference than that of MPD(g_p).

Theorem 2. *Let g_p and g_q be two combinations. If $g_q \twoheadrightarrow MPD(g_p)$, none of the descendants of g_p can dominate g_q.*

To save some space, we omit the proofs of these theorems. Readers may refer to [12] for the details.

Algorithm 1. The pseudo code of the RCA algorithm (for k-CSQ).

```
   GIVEN: A top-$k$ combinatorial skyline query $k$-CSQ$(D, \mathcal{F}, c, \mathcal{P})$.
   FIND  : Top-$k$ combinatorial skyline tuples.
   /* $\mathcal{T}$ is a 2 $\times c$ array. Each entry of $\mathcal{T}$ is the solution of a subproblem.   */
   /* $t_0$ is the first element of $D$                                                                  */
 1 $\mathcal{T}[0][0] \leftarrow t_0$ ;
 2 for $col \leftarrow 1$ to $c - 1$ do /* Initial the first row of $\mathcal{T}$.                        */
 3     $\mathcal{T}[0][col] \leftarrow \phi$ ;
 4 for $row \leftarrow 1$ to $|D| - 1$ do
 5     for $col \leftarrow 0$ to $c - 1$ do
 6         $CT(row - 1, col - 1) \leftarrow \mathcal{T}[(row - 1)\%2][col - 1] \oplus t_{row}$ ;
 7         $CT(row - 1, col) \leftarrow \mathcal{T}[(row - 1)\%2][col]$ ;
 8         $\mathcal{T}[row\%2][col] \leftarrow CT(row - 1, col) \cup CT(row - 1, col - 1)$;
 9     $Results \leftarrow$ extract all combinatorial skyline tuples in $\mathcal{T}$;
10     if there are at least $k$ combinatorial skyline tuples in Results then
11         Sorting Result according to $\mathcal{P}$ ;
12         $g^{kth} \leftarrow$ extract the $k$-th combinatorial skyline tuple from Result;
13         Finding MPD for each combination in $\mathcal{T}$;
14         if the preference of $g^{kth}$ is higher than that of the MPD of each combination
           then
15             return the first $k$ combinatorial skyline tuples in Result;
   /* The following codes deals with the case that the number of combinatorial
      skyline tuples is less than $k$. All combinatorial skyline tuples will be returned.
      */
16 $Results \leftarrow$ extract_all_combinatorial_skyline($\mathcal{T}$, $row$, $c$);
17 Sorting Result according to $\mathcal{P}$ ;
18 return Results;
```

Recall the previous example in Figure 7. We will see what would happen if we apply the theorems regarding MPD to the process. We have found that in Row 3 $[t_2, t_3]$ is invalidated to be the third combinatorial skyline tuple. Hence, the algorithm proceeds to the next row. In Row 4, t_4 is added to construct the solutions in this row. The constructed combinations are ϕ, $[t_1]$, $[t_2]$, $[t_3]$, $[t_4]$, $[t_1, t_2]$, $[t_1, t_3]$, $[t_2, t_3]$, $[t_1, t_4]$, $[t_2, t_4]$, and $[t_3, t_4]$. We do the dominance test to find the combinatorial skyline in this row and find that $[t_1, t_2]$, $[t_1, t_3]$, and $[t_1, t_4]$ are the top-3 combinatorial skyline tuples in Row 4, and $[t_1, t_4]$ is the third among the top combinatorial skyline tuples. Next, we construct the MPD of each combination by applying Equation 2 to these combinations. Then, we compare $[t_1, t_4]$ with these MPDs, i.e., MPD(ϕ), MPD($[t_1]$), MPD($[t_2]$), MPD($[t_3]$), MPD($[t_4]$), MPD($[t_1, t_2]$), MPD($[t_1, t_3]$), MPD($[t_2, t_3]$), MPD($[t_1, t_4]$), MPD($[t_2, t_4]$), and MPD($[t_3, t_4]$). As the preference of $[t_1, t_4]$ is higher than that of the MPD of each combination, we know from the above theorems that none of the descendants of each combination can dominate $[t_1, t_4]$. Hence, the algorithm terminates at Row 4 and $[t_1, t_2]$, $[t_1, t_3]$, and $[t_1, t_4]$ are the final results to the user. Note that by applying Equation 2 and the theorems, numerous computations and comparisons can be saved and the process can be completed much faster than without having these theorems.

We call our algorithm the *Restricted Construction Algorithm* (*RCA*). The pseudo code of the RCA algorithm is shown in Algorithm 1.

4 Performance Evaluation

We conducted a set of experiments to evaluate the effectiveness of the proposed algorithm. The uniform data distribution and the anti-correlated data distribution are used to generate the datasets [3, 6, 8, 9]. These two datasets are commonly used in skyline research. We normalize all dimension values into $[0, 100]$.The generation of the datasets of these two different distributions is based on the paper in the literature [6]. We divide our experiments into two categories: the performance evaluation over (1) a small and (2) a large scale datasets. In the first category, a set of small-scale datasets is used so as to restrict the running time of the Brute-Force method to a measurable period of time. In the second category, a set of large-scale datasets are adopted to show the scalability of RCA.

The parameters and their varying ranges are summarized in Figure 8, in which default values are marked in boldface. All experiments are run on a PC with Intel Core2 CPU 2.4GHz and 2 GB main memory. The programs are compiled with Java v6 in Windows Vista system.

(A) Effect of Dataset Size: The performance results are shown in Figure 9. We use the dataset with the dimensionality 6 and vary the size of the dataset from 50 to 200 tuples. The result shows that RCA significantly outperforms the Brute-Force method. The query time of RCA is very small, only a fraction of a millisecond. However, the performance of Brute-Force degrades rapidly as the size of the dataset increases, especially when the data distribution is anti-correlated (note that only a small portion of the Brute-Force method can be

Parameter	Range or value
Data size for small-scale datasets	50, **100**, 150, 200
Data size for large-scale datasets	200, 1K, 20K, 40K, **60K**, 80K, 100K
Dimensionality (d)	4, **6**, 8, 10
Top-k size (k)	10, 20, **30**, 40, 50
Cardinality (c)	2, 3, **4**, 5, 6
Combinatorial Function ($\mathcal{F}$)	SUM

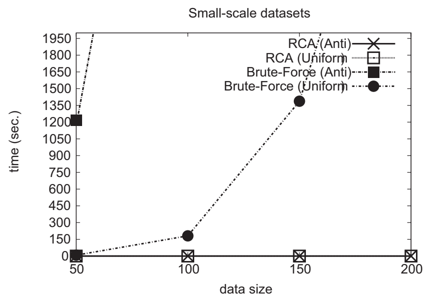
Fig. 8. Configuration Parameters

shown in the figure because the query time is too high). This implies that RCA can successfully reduce the computation cost by using the concept of the MPD, which eliminates unnecessary comparisons on combinations.

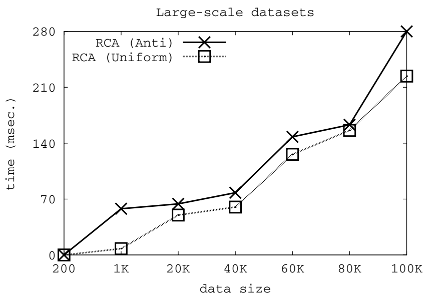
In the next experiment, we adopt large size datasets which vary from 200 to 100K. The query time of RCA is shown in Figure 9(b) and note that this time is in millisecond. The result of Brute-Force is not shown because its performance is over the scale of this figure. The result shows that the size of the dataset has a small impact on RCA. RCA achieves a satisfactory running time (i.e., within 1 sec.) even when the dataset size is so large. The result also shows that even when data is anti-correlated, its impact on RCA is limited. Its performance is only slightly worse than the performance of uniformly distributed data. The increase in computation cost for anti-correlated datasets is mainly due to the fact that the number of combinatorial skyline tuples in an anti-correlated dataset is greater than that in a uniformly distributed dataset.

(B) Effect of Dimensionality (d): The datasets with dimensionality d varying from 4 to 10 and data size = 100 are used in the experiment. Figure 10(a) shows the query times of both algorithms. It is clear that the performance of the Brute-Force method is much worse than that of RCA under all datasets. Its computation cost increases very rapidly with dimension.

In another experiment, we study the performance of RCA under various d over large-scale datasets. The result is shown in Figure 10(b). The result exhibits an interesting phenomenon that the query time of RCA decreases as the dimensionality rises. The reason is as follows. In the beginning, RCA iteratively constructs the solution table, and checks each combination in the current row to find the top-k combinatorial skyline tuples. In this sense, the performance of RCA is dependent on how fast the temporary set of top-k combinatorial skyline tuples is found. Since the number of combinatorial skyline tuples increases with the increasing dimensionality, RCA can acquire the temporary top-k combinatorial skyline tuples much faster in a high dimensional dataset, which allows RCA to find MPDS of the combinations earlier and therefore finish the whole process faster. Note that this feature of RCA is very unique in skyline query processing methods, as most methods performs worse when the dimensionality of data is high. RCA, however, provides an excellent idea for processing large and high-dimensional datasets.

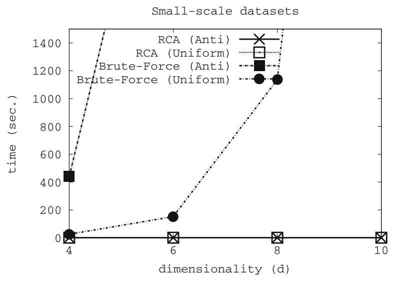


(a) Query time under various data sizes over small-scale datasets.

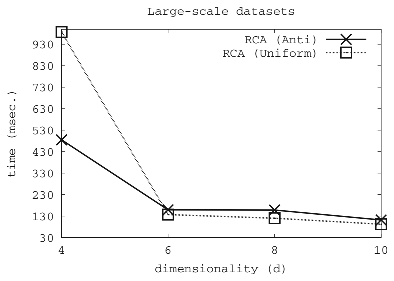


(b) Query time under various data sizes over large-scale datasets.

Fig. 9. Effect of data set size

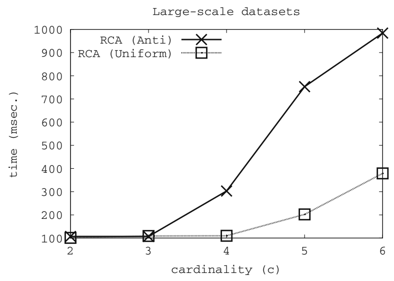


(a) Query time under various d over small-scale datasets.

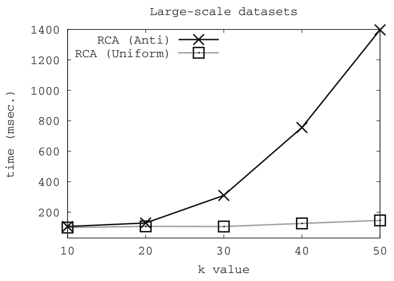


(b) Query time under various d over large-scale datasets.

Fig. 10. Effect of dimensionality



(a) Query time under various c over large-scale datasets.



(b) Query time under various k values over large-scale datasets.

Fig. 11. Effect of c and k

(C) Effect of Cardinality (c) and k: As the performance of Brute-Force is very poor, it is excluded from the discussion. Figure 11(a) studies the effect of c, the cardinality of a kCSQ. We fix the data size at 100 tuples and vary the cardinality from two to six. RCA scales gracefully with c, despite the exponential increase in the total number of combinations. Also, RCA performs better under uniformly distributed datasets. The reason is the same as that in effect of dataset size.

The impact of k is shown in Figure 11(b). Datasize, c and d at 100, 4, and 6, respectively, and increase k from 10 to 50. The performance of RCA indicates that the use of MPD helps RCA to finish the search process quite early so that the execution time even when $k = 50$ is still less than 1.5 sec.

5 Conclusion

Different from the previous work, our study introduces a new and useful type of skyline query, the combinatorial skyline query, and discusses the difficulties in retrieving the top-k combinatorial skylines. The existing techniques cannot be applied to resolve a top-k combinational skyline query. We proposed an algorithm to resolve the problem. Extensive experiments confirm that RCA is useful and is highly efficient for handling top-k combinatorial skyline queries. In our future work, we plan to further improve the performance of RCA by removing the dominated combinations from the solution table and propose other variations of this method.

References

[1] Reilly, F.K., Brown, K.C.: Investment Analysis & Portfolio Management, 7e. Thomson (2003)
[2] Markowitz, H.: Portfolio selection. Journal of Finance 7(1), 77–91 (1952)
[3] Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Transactions on Database Systems 30(1), 41–82 (2005)
[4] Chomicki, J., Godfrey, R., Gryz, J., Liang, D.: Skyline with presorting. In: Proceedings of the 19th International Conference on Data Engineering, ICDE 2003, pp. 717–719. IEEE Computer Society, Los Alamitos (2003)
[5] Gautam, D., Dimitrios, G., Nick, K., Dimitris, T.: Answering top-k queries using views. In: VLDB 2006: Proceedings of the 32nd international conference on Very large data bases, September 2006, pp. 451–462 (2006)
[6] Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th International Conference on Data Engineering, pp. 421–430. IEEE Computer Society, Los Alamitos (2001)
[7] Godfrey, P., Shipley, R., Gryz, J.: Maximal vector computation in large data sets. In: Proceedings of the 31st International Conference on Very Large Data Bases, August 2005, pp. 229–240 (2005)
[8] Li, C., Ooi, B.C., Tung, A.K.H., Wang, S.: Dada: a data cube for dominant relationship analysis. In: SIGMOD Conference, June 2006, pp. 659–670 (2006)

[9] Tan, K.L., Eng, P.K., Ooi, B.C.: Efficient progressive skyline computation. In: Proceedings of the 27th International Conference on Very Large Data Bases, September 2001, pp. 301–310 (2001)
[10] Michael, M., Patel, J.M., Jagadish, H.V.: Efficient skyline computation over low-cardinality domains. In: Proceedings of the 33rd international conference on Very large data bases, September 2007, pp. 267–278 (2007)
[11] Anderson, I.: A First Course in Discrete Mathematics. Springer, Heidelberg (2000)
[12] Su, I.F., Chung, Y.C., Lee, C.: Top-k combinatorial skyline queries. Technical report, Department of Computer Science and Information Engineering, National Cheng Kung University (2009), http://dblab.csie.ncku.edu.tw/~emily/techreportgroupskyline2009.pdf

Extract Interesting Skyline Points in High Dimension

Gabriel Pui Cheong Fung[1], Wei Lu[2,3], Jing Yang[3], Xiaoyong Du[2,3], and Xiaofang Zhou[1]

[1] School of ITEE, The University of Queensland, Australia
{g.fung,zxf}@uq.edu.au
[2] Key Labs of Data Engineering and Knowledge Engineering, Ministry of Education, China
[3] School of Information, Renmin University of China, China
{uqwlu,jyang,duyong}@ruc.edu.cn*

Abstract. When the dimensionality of dataset increases slightly, the number of skyline points increases dramatically as it is usually unlikely for a point to perform equally good in all dimensions. When the dimensionality is very high, almost all points are skyline points. Extract interesting skyline points in high dimensional space automatically is therefore necessary. From our experiences, in order to decide whether a point is an interesting one or not, we seldom base our decision on only comparing two points pairwisely (as in the situation of skyline identification) but further study how good a point can perform in each dimension. For example, in scholarship assignment problem, the students who are selected for scholarships should never be those who simply perform better than the weakest subjects of some other students (as in the situation of skyline). We should select students whose performance on some subjects are better than a reasonable number of students. In the extreme case, even though a student performs outstanding in just one subject, we may still give her scholarship if she can demonstrate she is extraordinary in that area. In this paper, we formalize this idea and propose a novel concept called k-dominate p-core skyline (C_p^k). C_p^k is a subset of skyline. In order to identify C_p^k efficiently, we propose an effective tree structure called Linked Multiple B'-tree (LMB). With LMB, we can identify C_p^k within a few seconds from a dataset containing 100,000 points and 15 dimensions.

1 Introduction

Given a set of points X in an N dimensional space, a point $x_i \in X$ dominates another point $x_j \in X$ if x_i performs better than x_j in at least one dimension and performs not worse than x_j in all other dimensions [1]. A point which cannot be dominated by any other point is called a skyline point. A collection of skyline points formulates a skyline. By definition, we cannot compare skyline points with each other without an appropriate preference function [2,3] because each skyline point must have at least one dimension performs better than any other skyline point. If the dimensionality is very high, almost all points are skyline points [4], extracting interesting skyline points in high dimensional space is therefore necessary [5,6].

* Wei Lu's and Xiaoyong Du's research is partially supported by National Natural Science Foundation of China under Grant 60873017.

Table 1. Academic results of some students

Names	**Subjects**				
	Language	Music	Math	Sport	Art
Amy	A	A	A	A	C
Billy	A-	A-	A-	C	C+
Cathy	A	A	B	C	A
Dorothy	B	B	B	B	C
George	F	F	F	C+	C+

For the sake of discussion, Table 1 shows a simply toy example about the academic results of five students. The first three students perform good but the last student performs very poor. Unfortunately, under the definition of skyline, the skyline points are: Amy, Billy, Cathy and George (Dorothy is dominated by Amy). This means *these four students are indistinguishable in terms of their academic achievements*. This does not make too much sense – we will never consider George's performance is good. George is a skyline point because *he performs slightly better than the weakest subject of any given student*, despite of *he performs significantly poorer than all students in most subjects*.[1]

1.1 Preliminaries

[5] extract interesting skyline points as follows: a point x_i is said to k-dominate another point x_j if x_i performs better than x_j in at least one dimension and not worse than x_j in k dimensions. A point that cannot be k-dominated by any other point is a k-dominant skyline point. Quite obvious, k-dominant skyline is a subset of skyline but its size is much smaller especially when k is small. In Table 1, Amy and Cathy both k-dominate George if $k \leq 3$. Hence, George, a non-interesting point, will not be included in the k-dominant skyline. k-dominant skyline is proven to be very effective in extracting interesting skyline points. Yet, the k-dominance relationship is not transitive. Furthermore, they do not consider vertical relationship of each dimension (as we will discuss it later). [7] proposes a concept called skyline frequency to extract interesting skyline points. It ranks all points according to the number of subspaces that they will be regarded as skyline points and returns the points at the top ranking. This approach is smart, but have two major issues: (1) Some points that are intuitively more superior may sometimes have the same ranking (or lower ranking) as the inferior points. (2) There are $2^N - 1$ subspaces in an N dimensional space, which is too expensive to compute in practice. For the first issue, let us consider Table 1 again. Billy obviously outperforms George. Unfortunately, they have the same ranking (they are skyline point in 8 out of 31 subspaces). For the second issue, [7] proposes an approximation algorithm to solve it but it cannot answer dynamic queries [8]. [9] proposes an idea called δ-subspace to identify a set of interesting skyline points. δ-subspace is a subspace of the whole dataspace such that its skyline contains less than δ points. The union of the skyline points in all the δ-subspace are

[1] *Note: One may argue that the problem in Table 1 is just a matter of scaling. However, we would like to stress that Table 1 is a simply example to demonstrate the potential problem of skyline in high dimensional space. Readers can refer to other examples from some existing work [5,6] if they are confused about the potential problem of skyline in high dimensional space.*

called strong skyline points. This algorithm depends on δ heavily. If δ too large, most of the skyline points will be removed, and vice versa. Setting δ is difficult for a normal user or without analyzing the dataset throughly. In addition, the physical meaning of this approach is not intuitive, which further discourage it to be used in practice. [10] proposes a skyline operator called top-K representative skyline query. In this operator, K skyline points will be selected such that the number of points which can be dominated by them will be maximized. Our problem is different from them. Firstly, they do not have vertical comparison among dimension. Secondly, superior points may frequently dominate less number of points than those inferior points. Thirdly, their aim is to rank the skyline points. Another related concept to our problem is K-skyband [8,11,12,13], which is a set of points that are dominated by at most K points. Yet, K-skyband does not aim at minimizing the number of skyline points which is different from our motivation. We do not attempt the elaborate them further due to the limited space.

1.2 Motivation and Objective

Let us take a simple example to illustrate our motivation. In scholarship assignment problem, scholarships are usually given to students based on two steps: (1) Identify some potential candidates from all students; (2) Assign scholarships to some of these potential candidates after some discussions. Ideally, potential candidates can be regarded as skyline because no student should perform better than any of them. In practice, if we identify the potential candidates by traditional skyline, the number of potential candidates will be too many when the number of subjects is many (i.e. dimensionality is high). We need to have a mechanism to reduce the size of skyline. For example, George in Table 1 should not be considered as a potential candidate. Logically, a potential candidate should not be a student who simply performs better than the weakest subject of any other student (like George). A potential candidate should has some subjects that she can perform better than a reasonable number of students in those subjects. In the extreme case, even though a student performs outstanding in one subject, she may still award scholarship if she can demonstrate she is extraordinary in that area. Based on this idea, we claim that when we extract interesting points from a large dataset, we should not only make *horizontal pairwise comparisons* like the concept of skyline. We should further study how many points that a given point can dominate according to a given dimension (i.e. *vertical pairwise comparison*). For example, suppose there are n students perform better than a student called Smith in a subject. If Smith wants to claim he is one of the potential candidates for awarding scholarship, then it is reasonable to ask him to identify another subject that he can perform better than *all* these n students. In Table 1, George cannot fulfill this criterion.

In this paper, we formalize this motivation and propose a novel concept called *core skyline*. A core skyline contains a set of interesting points extracted from a skyline. This work is different from the existing works as we further consider the vertical relationships among points. Up to our knowledge, we are the first mover in this area.

Given a point x_i and a dimension d_m let $M(x_i, d_m)$ be a set of points performs better than x_i in d_m. x_i is said to *dominate-back* $M(x_i, d_m)$ if x_i performs better than all points in $M(x_i, d_m)$ in some other dimensions. x_i is a core skyline point if it can dominate-back $M(x_i, d_m)$ for all different d_m in a dataset. Based on this concept, we further propose

Table 2. Some sample datasets

ID	d_1	d_2	d_3	d_4
x_1	1	7	3	5
x_2	3	2	7	7
x_3	5	4	2	3
x_4	4	5	6	6
x_5	7	1	1	1
x_6	6	3	4	2
x_7	2	6	5	4

A. Dataset 1 (Order by ID)

Rank	d_1	d_2	d_3	d_4
1	x_1	x_5	x_5	x_5
2	x_7	x_2	x_3	x_6
3	x_2	x_6	x_1	x_3
4	x_4	x_3	x_6	x_7
5	x_3	x_4	x_7	x_1
6	x_6	x_7	x_4	x_4
7	x_5	x_1	x_2	x_2

B. Dataset 1 (Order by ranking)

Insertion order	**Points**	**Dimensions**		
		d_1	d_2	d_3
1	x_1	7	10	8
2	x_2	7	10	11
3	x_3	16	10	17
4	x_4	18	15	1
5	x_5	12	10	20
6	x_6	7	10	8

C. Dataset 2

a more robust concept called *k-dominant p-core skyline* (C_p^k). k denotes the number of dimensions that x_i has to dominate-back and p denotes the fraction of points in a dimension that x_i has to dominate-back. Details of this formulation will be described in Section 2. We propose an indexing algorithm in Section 3 called Linked Multiple B'-tree (LMB)[2] to help us identify C_p^k dynamically and progressively.

2 Problem Definition

Let $D = \{d_1, d_2, \ldots, d_N\}$ be an N-dimensional space and $X = \{x_1, x_2, \ldots\}$ be a set of points in D. We use $x_i.d_m$ to denote the value of x_i in d_m. There is a total order relationship in each dimension such that a smaller value is more preferable. Let us review two existing definitions [14]:

Definition 1. (Dominate, $\prec$) *x_i is said to dominate x_j ($x_i \prec x_j$) iff these conditions hold: (1) $\forall d_m \in D, x_i.d_m \le x_j.d_m$; and (2) $\exists d_m \in D, x_i.d_m < x_j.d_m$.*

Definition 2. (Skyline, S) *Let $S \subseteq X$. x_i is a skyline point ($x_i \in S$) if and only if $\forall x_j \in X, x_j$ cannot dominate x_i.*

We now formally define core skyline and use Table 2A and 2B as a running example. Table 2A contains a dataset with seven points and four dimensions. Table 2B ranks these points according to their values in each dimension. For example, x_1 is ranked highest in d_1 because it has the smallest value in d_1. Note that all points in Table 2A and 2B are skyline points. Before we can define core skyline, we need to define the following:

Definition 3. (Dominant set, $M(x_i, d_m)$) *Let $M(x_i, d_m) \subset X$. $M(x_i, d_m)$ contains points that perform better than or equal to x_i in d_m, i.e. $x_j \in M(x_i, d_m)$ iff $x_j.d_m \le x_i.d_m$.*

Definition 4. (Dominate-back, $\prec_b$) *Let $M \subset X$. A point x_i is said to dominate-back M ($x_i \prec_b M$) iff $M = \emptyset$ or $\exists d_n$ such that $\forall x_j \in M, x_i.d_n \le x_j.d_n$.*

In Table 2A, $M(x_1, d_1) = \emptyset$ because no point has a value less than $x_1.d_1$. Similarly, $M(x_1, d_2) = \{x_2, x_3, x_4, x_5, x_6, x_7\}$, $M(x_1, d_3) = \{x_3, x_5\}$ and $M(x_1, d_4) = \{x_3, x_5, x_6, x_7\}$. Given x_i, if $x_i \in S$ and $x_i \prec_b M(x_i, d_m), \forall d_m \in D$, then x_i is a core skyline point:

[2] "B'-tree" is pronounced as "B-pie-tree".

Definition 5. (Core skyline, C) *Let $C \subseteq S$. A point x_i is a core skyline point ($x_i \in C$) iff $x_i \in S$ and $\forall d_m \in D, x_i \prec_b M(x_i, d_m)$.*

In Table 2A, $C = \{x_1, x_5\}$. All other points do not belong to core skyline. For example, let us consider x_4. In order for x_4 to be a core skyline point, x_4 must be able to dominate-back $M(x_4, d_1)$, $M(x_4, d_2)$, $M(x_4, d_3)$ and $M(x_4, d_4)$. Now, let us consider $M(x_4, d_1) = \{x_1, x_2, x_7\}$. In order for $x_4 \prec_b M(x_4, d_1)$, there must exists a d_n such that $x_4.d_n \leq x_1.d_n$, $x_4.d_n \leq x_2.d_n$ and $x_4.d_n \leq x_7.d_n$. Unfortunately, $x_4.d_2 > x_2.d_2$ (i.e. d_2 fails), $x_4.d_3 > x_1.d_3$ (i.e. d_3 fails) and $x_4.d_4 > x_1.d_4$ (i.e. d_4 fails). Since $x_4 \not\prec_b M(x_4, d_1)$, $x_4 \notin C$. When the dimensionality, N, is very high, it will be very difficult for a point to become a core skyline point because it is difficult for a point to dominate-back all N dominant-sets. Hence, k-dominant core skyline is proposed. k denotes the number of dominant-sets that a point has to dominate-back. Formally:

Definition 6. (k-dominant core skyline, C^k) *Let $C^k \subseteq S$. A point x_i is a k-dominant core skyline point ($x_i \in C^k$) iff $x_i \in S$ and $\exists D' \subseteq D$, $|D'| = k$, $\forall d_m \in D'$, $x_i \prec_b M(x_i, d_m)$.*

Based on Definition 6, $C^4 = \{x_1, x_5\}$, $C^3 = \{x_1, x_3, x_5, x_7\}$, $C^2 = \{x_1, x_2, x_3, x_5, x_6, x_7\}$ and $C^1 = \{x_1, x_2, x_3, x_5, x_6, x_7\}$. Note that x_4 is a skyline point but is not a k-dominant core skyline point. Also, $C^k \subseteq C^{k'}$ for $k < k'$. When there are many points in a dataset, it will be very difficult for a point x_i to dominate-back *all* points in $M(x_i, d_m)$. As such, one may consider x_i is important if it can dominate-back a reasonable number of points in $M(x_i, d_m)$. As a result, a relation called p-dominate-back is defined:

Definition 7. (p-dominate-back, $\prec_{pb}$) *Let $M \subset X$ and $0 \leq p \leq 1$. A point x_i is said to p-dominate-back M ($x_i \prec_{pb} M$) iff $\exists M' \subseteq M, |M'| \geq p \times |M|$ such that $M' = \emptyset$ or $\exists d_n, \forall x_j \in M', x_i.d_n \leq x_j.d_n$.*

Yet, we cannot have the concept of p-dominate-back in k-dominant core skyline by simply replacing $\prec_b$ to $\prec_{pb}$ because it is possible that some non-interesting points might be able to p-dominate-back a large number of its dominant-sets while some interesting points cannot (we obmit this proof due to limited space). So we have a new definition:

Definition 8. (k-dominant p-core skyline, C^k_p) *Let $C^k_p \subseteq S$. A point x_i is a k-dominant p-core skyline point ($x_i \in C^k_p$) if and only if $x_i \in S$ and $\exists D' \subseteq D, |D'| = k, \forall d_m \in D'$ both these two conditions hold: (1) $x_i \prec_{pb} M(x_i, d_m)$; (2) $\forall x_j \in M(x_i, d_m), x_i \prec_{pb} M(x_j, d_m)$.*

In Definition 8, Condition (1) is trivial but will lead to the aforementioned pitfall. So Condition (2) is imposed. x_i can be a k-dominant p-core skyline point only if it can p-dominate-back $M(x_j, d_m)$ for all $x_j \in M(x_i, d_m)$. Note that $C^k_0 = C^0_p = C^0_0 = S$. Since core skyline ($C$) and k-dominant core skyline (C^k) are special cases of C^k_p ($C = C^N_1$ and $C^k = C^k_1$), we will focus on studying C^k_p. The problems that we want to solve are:

1. Given k and p, extract C^k_p dynamically and progressively.
2. Given δ, identify the smallest k' such that $|C^{k'}_p| \leq \delta$ and $0 \leq k' \leq N$. If no k' satisfies this condition, then $k' = N$.

Problem 1 is trivial. For Problem 2, from a user's point of view, it is convenience because a user only needs to specify the maximum number of points that she wants to obtain but does not need to understand the distribution of data.

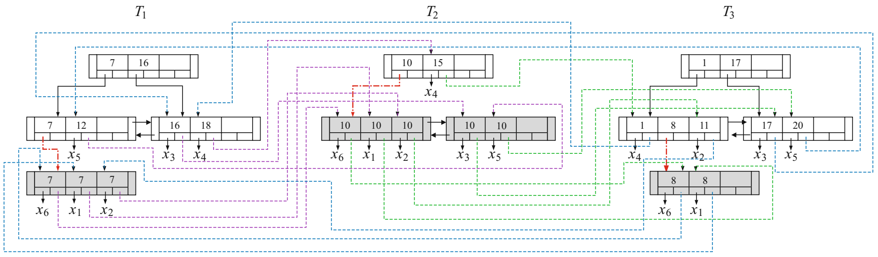


Fig. 1. General structure of the Linked Multiple B'-tree (LMB)

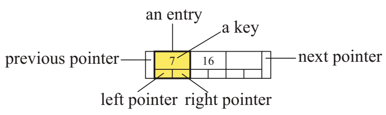


Fig. 2. A node in a B'-tree

3 Proposed Work: LMB

In this section, we describe how to extract C_p^k efficiently by using a novel tree structure called LMB. Note that we do not need to identify skyline before we extract C_p^k. This is important as most skyline points may not belong to C_p^k when p and k are of reasonable values. We can then minimize the computational cost.

Figure 1 shows the structure of LMB by using the dataset in Table 2C . There are three B'-tree, T_1, T_2 and T_3, linked together. T_1, T_2 and T_3 are responsible for indexing the data in d_1, d_2 and d_3, respectively. Each node in a B'-tree contains some entries, a previous pointer and a next pointer (Figure 2). The next pointer points to a succeeding sibling node (if any) and the previous pointer points to a preceding sibling node (if any). Each entry in a node contains a key and two pointers. The left pointer either points to a record or points to a node. The right pointer either points to an entry in another B'-tree that refers to the same point or points to null. In real implementation, each node is a page in disk so a node should have more entries rather than three. The nodes that are shaded are called *overflow nodes*. Overflow nodes are *not* regarded as leave nodes. So the heights of T_1, T_2 and T_3 are respectively two, one and two. In a B'-tree, records are stored at leave nodes and overflow nodes. Each overflow node contains points sharing the same key. For example, T_1 contains one overflow node (key 7) with three entries. This indicates there are three points having the value 7 in d_1.

Points in an overflow node are ordered according to their values in *some other dimensions*. For example, given two points x_i and x_j, where $x_i.d_m = x_j.d_m$, there are two possibilities for their values in the other dimensions: (1) $\forall d_n \in D, x_i.d_n = x_j.d_n$ (i.e. x_i and x_j are identical), and (2) $\exists d_n \in D, x_i.d_n \neq x_j.d_n$. For Case (1), x_i and x_j will be ordered in an overflow node according to their reverse order of insertion. E.g. x_1 and x_6 in Table 2C are identical, so their orders in T_1, T_2 and T_3 are all $x_6 \rightarrow x_1$ (x_6 is on the left of x_1) as x_6 is inserted after x_1. For Case (2), B'-tree will continue to compare

Algorithm 1. `insert`(x_i, T_m)

```
input : A point $x_i$ and a B'-tree $T_m$
$\nu \leftarrow x_i.d_m$;
if $\exists \nu$ in $T_m$ then
    if overflow node for the key $\nu$ does not exist then
        node $\leftarrow$ a new overflow node;
        identify entry where entry.key = $\nu$ and entry is in leave node;
        insert $x_j$ into node where $x_j$ = entry.left;
        entry.left $\leftarrow$ node;    entry.right $\leftarrow$ null;
    else
        node $\leftarrow$ first overflow node of $\nu$;
    insert($x_i, d_m$, node);
else
    insert $x_i$ into $T_m$ just like a traditional B+tree;
```

the values of x_i and x_j in the immediate next dimension, until their values are different. E.g. in d_2, $x_1.d_2 = x_2.d_2 = x_3.d_2 = x_5.d_2 = x_6.d_2 = 10$. To order these points in T_2, we compare their values in the immediate next dimension of d_2, which is d_3. In d_3, $x_1.d_3 = 8$, $x_2.d_3 = 11$, $x_3.d_3 = 17$, $x_5.d_3 = 20$, $x_6.d_3 = 8$. So their ordering in T_2 is $(x_1, x_6) \rightarrow x_2 \rightarrow x_3 \rightarrow x_5$. To order x_1 and x_6 in T_2, we compare their values in the immediately next dimension, which is d_1. Since x_1 and x_6 are identical, we order them according to the reverse order of their insertion, which is $x_6 \rightarrow x_1$. Eventually, the ordering is $x_6 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_5$.

3.1 Implementation

Algorithm 1 outlines the insertion process. Suppose we now insert a new point, x_i, into T_m. First, we check whether there is any point has the same value as x_i in d_m (line 2). If not, the insertion process will be the same as a B+tree [15] (line 13). Otherwise, we check whether an overflow node exists (line 3). If not, we will create a new overflow node (line 4), identify the entry at leave node where its key is $x_i.d_m$ (line 5), insert x_j ($x_j.d_m = x_i.d_m$) into the overflow node (line 6) and attach the overflow node to the correct position in T_m (line 7). x_i will be inserted into the overflow node by calling the function `insert` (line 11).

Algorithm 2 outlines the process of inserting x_i into an overflow node. We iterate each point x_j in an overflow node to see which position should we insert x_i. Let $d_{m'}$ be the immediate next dimension of d_m (line 4). If $x_i.d_{m'} < x_j.d_{m'}$ (line 5), then x_i will be inserted before x_j (line 6). We can do this because all existing points in an overflow node must already be ordered properly based on $d_{m'}$ when they are inserted by calling this function previously. If $x_j.d_{m'} = x_i.d_{m'}$ (line 6), then we will compare the values of x_i and x_j in their immediate next dimension continuously until they are different (line $7-11$). If $\forall d_{m'}, x_i.d_{m'} = x_j.d_{m'}$ (i.e. x_i and x_j are identical), then x_i will be inserted before x_j (line 12) because we have to order the identical points in their reverse order of insertion. Line $14-18$ is used to iterate all points in an overflow node. Finally, if

Algorithm 2. insert($x_i, d_m, node$)

```
input : A point, $x_i$, a dimension, $d_m$, an overflow node, $node$
1  $i \leftarrow 1$;
2  repeat
3      $x_j \leftarrow$ the point at entry $i$ in $node$;
4      if $m \neq N$ then $m' \leftarrow m+1$; else $m' \leftarrow 1$;
5      if $x_i.d_{m'} < x_j.d_{m'}$ then insert $x_i$ before $x_j$; return;
6      else if $x_j.d_{m'} = x_i.d_{m'}$ then
7          for $i \leftarrow 1$ to $N-1$ do
8              if $m' \neq N$ then $m' \leftarrow m'+1$; else $m' \leftarrow 1$;
9              if $x_i.d_{m'} < x_j.d_{m'}$ then insert $x_i$ before $x_j$; return;
10             else if $x_i.d_{m'} > x_j.d_{m'}$ then insert $x_i$ after $x_j$; return;
11         insert $x_i$ before $x_j$; return;
12     $i \leftarrow i+1$;
13     if entry $i$ is empty then insert $x_i$ in entry $i$; return;
14     else if $i >$ number of entries in node then
15         $i \leftarrow 1$;    $node \leftarrow node.next$;
16 until node is null ;
17 $node \leftarrow$ a new overflow node;
18 insert $x_i$ in the first entry of $node$;
19 attach $node$ to the last overflow node;
```

$\forall x_j, x_i.d_{m'} > x_j.d_{m'}$, then x_i be inserted into the last entry of an overflow node (line 15). If the entries are full, then x_i will be inserted into a new overflow node(line $20-22$). For each entry, its right pointer will point to an entry in the immediate next B'-tree that points to the same record. This process is trivial and can be performed in any stage during the insertion process. For deletion, its steps are similar to a B+tree, except that when an overflow node contains only one entry, then this entry will be propagated back to its parent node and the overflow node will be deleted. As this process is trivial, we do not present a detailed algorithm due to the limited space in this paper.

3.2 Extraction

Once LMB has indexed all points, we can extract C_p^k *dynamically* and *progressively* according to a reference point, r (e.g. r is the origin). Given a point x_i and a dimension d_m, suppose all points in $M(x_i, d_m)$ are ordered according to d_n ($d_n \neq d_m$) in an descending order. Let x^* be the point at position $\lceil p \times |M(x_i, d_m)| \rceil$ in $M(x_i, d_m)$. If $x_i.d_n < x^*.d_n$, then x_i obviously must be able to p-dominate-back $M(x_i, d_m)$ by using d_n. Hence, Condition (1) of Definition 8 can be verified quickly if x^* can be identified efficiently. Now, assume there is a point $x_j \in M(x_i, d_m)$. In order for x_i p-dominate-back $M(x_j, d_m)$ using d_n, $x_i.d_n$ must be less than or equal to $x_j^*.d_n$ where x_j^* is the point at position $\lceil p \times |M(x_j, d_n)| \rceil$ (remember $M(x_j,\, d_m)$ is sorted based on d_n). Hence, to verify Condition (2) of Definition 8 quickly, what we need to do is to check whether $\exists d_n, x_i.d_n \leq \min_j x_j^*.d_n$ where x_j^* is the point at position $\lceil p \times |M(x_j, d_n)| \rceil$. Once we can verify Condition (1) and Condition (2) of Definition 8 quickly, we can extract C_p^k efficiently. Hence, the major issue

Algorithm 3. `extract(`p,k,r`)`

```
input  : two user defined threshold, $p$ and $k$, and a reference point, $r$
output: $k$-dominant $p$-core skyline, $C_p^k$
1  for $m \leftarrow 1$ to $N$ do
2      $x^m \leftarrow$ the point closest to $r$ in $d_m$; // e.g. $r$ is the origin
3      $B_{mn} \leftarrow \emptyset, n = 1,2,\ldots,N$; // $B_{mn}$ is a BTree
4  $C_p^k \leftarrow \emptyset$;    $H \leftarrow \emptyset$;
5  for $i \leftarrow 1$ to $|X|$ do
6      for $m \leftarrow 1$ to $N$ do
7          $added \leftarrow false$;
8          for $n \leftarrow 1$ to $N, n \neq m$ do
9              if $B_{mn} = \emptyset$ or $|x^m.d_n - r_m.d_n| < B_{mn}.last$ then
10                 insert $|x^m.d_n - r_m.d_n|$ into $B_{mn}$;
11                 if $addded = false$ and $\nexists x \in H, x \prec x^m$ then
12                     $C_p^k \leftarrow C_p^k \cup$ update($H,k,x^m$);
13                     $added = true$;
14                 if $\lceil p \times i \rceil > |B_{mn}|$ then
15                     remove last element from $B_{mn}$;
16         $x^m \leftarrow$ the point closest to $x^m$ (besides itself) in $d_m$;
17     $i \leftarrow i + 1$;
18 return $C_p^k$;
```

we need to deal with is how to identify x^* and x_j^* with respect to d_m quickly. We extract C_p^k based on this idea.

Algorithm 3 outlines the steps for extracting C_p^k. Line $1-5$ initialize some parameters. In line 2, if there are more than one point closest to the reference point r in d_m, then x^m will be initialized to the one which is the first occurrence in an overflow node. With LMB, we can identify x^m very quickly. This process is similar to a B+tree. B_{mn} (line 3) is a BTree. It helps us to determine whether a point can p-dominant-back a dominant-set. If x_i can p-dominate-back $M(x_i,d_m)$ by using d_n, then x_i will be stored in B_{mn}. We may not always need to store x_i permanently in B_{mn}. We want to keep B_{mn} as small as possible so as to reduce memory consumption and computational time. We explain this step by step below. Let $\nu = |x^m.d_n - r_m.d_n|$. We can identify $x^m.d_n$ quickly by using LMB without accessing the record directly. Whenever B_{mn} is empty or ν is less than the last value in B_{mn} (i.e. the largest value in B_{mn}), then ν will be inserted into B_{mn} (line 10 and 11). Let i be the i^{th} point closest to the reference point r. Whenever $|B_{mn}| < \lceil p \times i \rceil$, then the last value in B_{mn} will be removed (line $16-18$). By doing so, we can keep the size of B_{mn} always not exceed $\lceil p \times |M(x^m,d_m)| \rceil$. We can do so because we extract x^m one by one according to the ascending distance to r (line 21). Furthermore, note that the last value in B_{mn} is in fact $\min\{x^*.d_n, x_j^*.d_n\}$ with respect to d_m. If $x^m < \min\{x^*.d_n, x_j^*.d_n\}$, it implies x^m satisfies Condition 1 and Condition 2 of Definition 8. So x^m will be added into H (line $12-15$). In line 12, the condition $\nexists x \in H, x \prec x^m$ guarantees x must be a skyline. The rest of Algorithm 3 should be self-explained. Finally, H (line 6) is a hash

Algorithm 4. `update`(H,k,x,v)

```
input  : A hashtable, $H$, a parameter, $k$, a point, $x$, a value, $v \in \{0,1\}$
output: A $k$-dominant $p$-core skyline point, $x$, or $\emptyset$
1 if $H$ does not contain element with key $x$ then
2     put $(x,V)$ into $H$; // $x$ is the key and $V$ (a set) is the element
3 add $v$ into $V$; // $V$ is the element with key $x$
4 $k' \leftarrow$ number of $v$ in $V$ equals 1;
5 if $k' \geq k$ and $x$ is not yet returned then
6     return $x$;
7 return $\emptyset$;
```

table. It stores how many dominant-sets that a point can p-dominant-back. The function `update` (line 24 and 26) is used to update the information stored in H so as to extract C_p^k. It is outlined in Algorithm 4. In Algorithm 4, V (e.g. line 2) is a set that stores the "p-dominate-back result" of x. If x can p-dominate-back a specified dominant-set, then 1 will be added into V; otherwise, 0 will be added (line 4). If the number of 1 in V is greater than or equals to k (k is a user-defined parameter to extract C_p^k), then x will be added into C_p^k (line $5-8$). Note that x is added into C_p^k progressively so we can return results to users immediately and progressively.

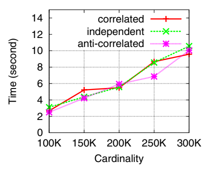
3.3 Further Analysis

Algorithm 3 tries to solve Problem 1 which is raised in Section 2. For Problem 2, we can solve it by: (1) Remove line 4 to line 8 in Algorithm 4; (2) The appropriate k' could be obtained easily by checking the number of $v = 1$ in each V in H after Algorithm 3 is completed. For example, when $N = 3$ (i.e. $|V| = 3$), suppose after completing Algorithm 3, there are five V having three $v = 1$, seven V having two $v = 1$ and ten V having one $v = 1$. Then, $|C_p^3| = 5$, $|C_p^2| = 12$, $|C_p^1| = 22$. If $\delta = 20$ (maximum number of core skyline points returned is 20), then $k' = 2$.

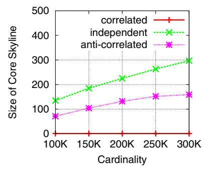
Furthermore, we can extend our algorithm to answer some complex queries easily. We give some examples here to illustrate how this can be done. If we want to return C_p^k for a particular range of k, then we simply change line 6 of Algorithm 4 to the appropriate range. For example, if we want to return C_p^k when $k = 2$ but exclude those points in C_p^k when $k = 4$, then we change line 6 to: "**If** $2 \leq k' < 4$ *and x is not yet returned* **then**". To deal with constraint skyline queries [16], we only need to pay attention to line 2 and line 33 of Algorithm 3. If the x_m returned is not within the constrained region, then we can immediately ignore that dimension d_m and do no need to conduct any further computation.

4 Experiment

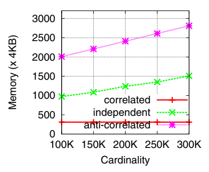
All experiments are conducted using an Intel XEON 2.5GHz CPU in Microsoft Windows Server 2003 R2 Enterprise x64 Edition. All programs are written in Java. We use



(a) CPU vs. Cardinality

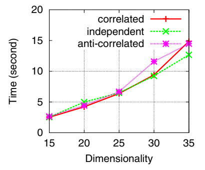


(b) $|C_p^k|$ vs. Cardinality

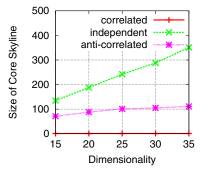


(c) Memory vs. Cardinality

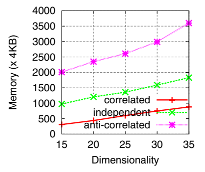
Fig. 3. Cardinality



(a) CPU vs. Dim.

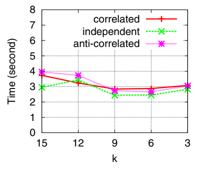


(b) $|C_p^k|$ vs. Dim.

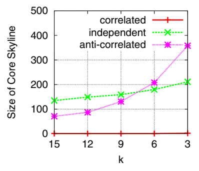


(c) Memory vs. Dim.

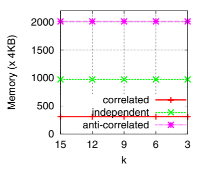
Fig. 4. Dim.



(a) CPU vs. k

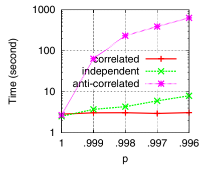


(b) $|C_p^k|$ vs. k

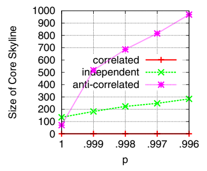


(c) Memory vs. k

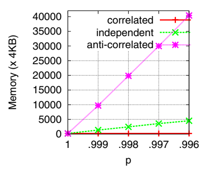
Fig. 5. Effect of k



(a) CPU vs. p



(b) $|C_p^k|$ vs. p



(c) Memory vs. p

Fig. 6. Effect of p

a page size of 4KB for each node of LMB. Following [5,16], we generate several independent, correlated and anti-correlated datasets. In order to evaluate the quality of C_p^k, we use two real life datasets.

4.1 Effect of Cardinality

We set $|X|$ (number of points) $= \{100000, 150000, 200000, 250000, 300000\}$, $N = 15$, $k = 15$ and $p = 1$. Figure 3 (a) shows the CPU time versus cardinality in different datasets. In the figure, there are three lines. They denote the results against independent dataset, correlated dataset and anti-correlated dataset. With LMB, we only need to spend less than 3 seconds to identify C_p^k from a dataset with 15 dimensions and 100,000 points and spend around 11 seconds to identify C_p^k from a dataset with 15 dimensions and 300,000 points. In general, the CPU time increases linearly when the dimensionality increases linearly. For a reference, BBS [8] (the most efficient algorithm to identify skyline) takes more than 1,000 seconds to identify skyline from an independent dataset with 15 dimensions and 100,000 points and takes more than 2,200 seconds to identify skyline from an anti-correlated dataset with the same setting. Figure 3 (b) shows the size of C_p^k versus cardinality. For the independent dataset, $|C_p^k|$ increases linearly. For the correlated dataset, $|C_p^k|$ almost constant (less than 5). For the anti-correlat-ed dataset, $|C_p^k|$ increases slowly when $|X| > 250K$. For the same $|X|$, the size of C_p^k in an independent dataset is always larger than the size of C_p^k in an anti-correlated dataset. Figure 3 (c) shows the memory consumption versus cardinality. The trend of is highly related to the size of C_p^k because we always need to keep a fix amount of information (B_{mn} and

H in Algorithm 3) in the main memory. For each B'-Tree, B_{mn} is more or less constant ($|B_{mn}|$ is usually around $p \times |M(x_i, d_m)|, \forall n$) but $|H|$ is highly related to the number of skyline points in the dataset.

4.2 Effect of Dimensionality

We set N (dimensionality) $= \{15, 20, 25, 30, 35\}$, $|X| = 100{,}000, p = 1$ and $k = 1$. Figure 4 (a), 4 (b) and 4 (c) respectively show the CPU time, the size of C_p^k and the max. memory consumption in different datasets. We can identify k-dominant p-core skyline points within 15 seconds for all datasets even when $k = 35$. The computational time roughly increases linearly. When the dimensionality increases, $|C_p^k|$ does not vary much in the anti-correlated and the correlated datasets. Note that more than 95% of points are skyline points when $|X| = 100{,}000$ and $N = 35$ in the anti-correlated dataset. For the independent datasets, $|C_p^k|$ increases linearly. Nevertheless, more than 90% of points are skyline points when $N = 35$, but the number of core skyline points is just less than 400. We can reduce the number of skyline points dramatically. For the memory consumption, when we compare Figure 3 (c) and Figure 4 (c), Figure 3 (c) is more flat because when the dimensionality increases, we need to store more p-dominate-back information (i.e. B_{mn} in Algorithm 3) for each B'-Tree.

4.3 Effect of k

We set k (number of dominant-sets a point has to p-dominate-back) $= \{15, 12, 9, 6, 3\}$, $|X| = 100{,}000$ $N = 15$ and $p = 1$. Figure 5 (a), 5 (b) and 5 (c) respectively show the CPU time, the size of C_p^k and the maximum memory consumption against k in different datasets. For the CPU time, all lines are roughly constant (or having a slightly decreasing trend) regardless of the choice of k. Technically, when k is small, we have less comparisons before we can decide whether a point should be returned. In practice, it seems that the computational time differences between a small k and a large k is negligible. For the size of C_p^k, the rate of increase of the anti-correlated dataset is much faster than the independent dataset. For the memory consumption, it is constant.

4.4 Effect of p

We set $p = \{1, 0.999, 0.998, 0.997, 0.996\}$, $|X| = 100{,}000$, $N = 15$ and $k = 1$. Figure 6 shows the results. It is quite obvious that p has a significant impact on C_p^k (especially the anti-correlated dataset). In Figure 6 (a), the time requires to identify C_p^k from an anti-correlated dataset increases exponentially. Fortunately, having a small p in our problem is unreasonable as our objective is to extract a small set of interesting skyline points. When the value of p decreases, the size of C_p^k increases dramatically. This is shown in Figure 6 (b). When $p = 1$, $|C_p^k|$ is less than 100 in the anti-correlated dataset; when $p = 0.996$, $|C_p^k|$ is around 900. For the memory consumption (Figure 6 (c)), the memory needed for correlated dataset and independent dataset do not vary much. However, the anti-correlated dataset does require a large amount of memory when p decreases.

Table 3. Players extracted by C_p^k

k	p	Players
11	1	Alvin Robertson 1985, Dennis Rodman 1991, Hakeem Olajuwon 1989, John Stockton 1990, Latrell Sprewell 1993, Michael Jordan 1986, Moses Malone 1979, Ray Allen 2005 (**8 records, 8 players**)
5	1	Allen Iverson 2002, Alvin Robertson 1985, Antoine Walker (2000, 2001), Charles Barkley 1988, Dennis Rodman 1991, Gary Payton 1999, George Mccloud 1995, Gilbert Arenas 2005, Hakeem Olajuwon (1988, 1989, 1992), Isiah Thomas 1984, Jason Richardson 2007, John Stockton (1988, 1990, 1991), Karl Malone 1989, Kevin Garnett (2002, 2003), Kevin Willis 1991, Kobe Bryant 2005, Latrell Sprewell 1993, Magic Johnson 1986, Mark Eaton 1984, Michael Jordan (1986, 1987, 1988, 1989), Michealray Richardson 1979, Moses Malone (1979, 1981, 1982), Predrag Stojakovic 2003, Ray Allen 2005, Shaquille O'neal 1999 (**36 records, 25 players**)
11	0.94	Adrian Dantley 1980, Allen Iverson (2002, 2007), Alvin Robertson (1985, 1986), Anthony Mason 1995, Antoine Walker 2001, Dennis Rodman 1991, Dennis Scott 1995, Gary Payton 1999, George Mccloud 1995, Gilbert Arenas 2005, Hakeem Olajuwon (1988, 1989), Jason Richardson 2007, John Stockton (1987, 1988, 1990, 1991), Kevin Garnett (2002, 2003, 2004), Kobe Bryant 2005, Latrell Sprewell 1993, Magic Johnson (1981, 1982), Michael Finley 1999, Michael Jordan (1986, 1987, 1988), Michealray Richardson 1979, Mitch Richmond 1995, Mookie Blaylock (1993, 1995), Moses Malone (1979, 1981,1982), Predrag Stojakovic 2003, Ray Allen 2005, Reggie Miller 1996, Shaquille O'neal 1999, Steve Nash (2006, 2007), Tim Hardaway 1991 (**44 records, 29 players**)

4.5 Quality of Result

We evaluate the quality of C_p^k using two real life datasets. One is called NBA dataset and the other one is called MovieLens. [3]. The NBA dataset has 11,301 records. It includes all NBA players from 1979 to 2007. Each record denotes the average performance of a player in a year. The MovieLens dataset has 3,952 movies. The schema for NBA dataset is: minutes played, points obtained, offensive rebound obtained, defensive rebound obtained, total rebound obtained, assistant made, steal made, block made, three points made, free throw made and field goal made. The schema for MovieLens dataset is: Female rating, Male rating, Female under, Male under, Female 19 - 25, Male 19 - 25, Female 26 - 35, Male 26 - 35, Female 36 - 45, Male 36 - 45, Female 46 - 50, Male 46 - 50, Female 51 - 55, Male 51 - 55, Female 56 or above, Male 56 or above and Overall rating. Due to the limited space, we only report some of the most interesting findings. Readers can download all results online. We implement the k-dominant skyline algorithm [5] for comparison. This algorithm is proved to be very effective in extracting interesting points from a large set of skyline points. *We do not compare this algorithm in the previous experiments because we are neither an extension of it nor targeting to obtain the same result.*

Table 3 shows the players extracted by C_p^k. All players are sorted by their first names. [5] We report C_1^{11}, C_1^5 and $C_{0.95}^{11}$. We choose them because: (1) C_1^{11} is the most tight constraint; (2) C_1^5 implies a player should p-dominate-back around half of dimensions in the

[3] www.databasebasketball.com and www.grouplens.org

Table 4. Players extracted by k-dominant skyline [5]

K	Players
9	Alton Lister 1986, Charles Barkley (1985, 1986, 1987, 1988), Charles Oakley 1986, David Robinson (1990, 1992, 1993, 1994, 1995), Dennis Rodman (1991, 1993), Dikembe Mutombo (1995, 1999), Dirk Nowitzki 2002, Dwight Howard 2007, Elvin Hayes 1979, Gary Payton 1999, Hakeem Olajuwon (1988, 1989, 1992, 1993, 1994), James Donaldson 1986, Julius Erving 1980, Karl Malone (1989, 1990, 1991, 1992, 1993), Kevin Garnett (1999, 2000, 2001, 2002, 2003, 2004, 2006), Larry Bird (1983,1986), Mark West 1989, Michael Jordan (1986, 1987, 1988, 1989), Moses Malone (1979, 1980, 1981, 1982), Patrick Ewing (1989, 1990), Samuel Dalembert 2006, Shaquille O'neal (1992, 1993, 1999), Shawn Marion 2002, Tim Duncan (2001, 2002) (**58 records, 24 players**)

dataset. This make sense in the basketball situation; (3) The number of skyline points returned by $C^5_{0.94}$ in this dataset is similar to the number of skyline points returned by C^5_1. We can compare the results returned by $C^5_{0.94}$ and C^5_1.

One frequently asked question is that "why X is not included? I think he plays equally well with Y!" Yet, this type of question is subjective. Furthermore, one cannot deny the fact that the players in Table 3 are all famous. In addition, there are more than 1,000 skyline points in the dataset. Some skyline points belong to some not-so-famous players. If we randomly extract some skyline points from the dataset, it is very likely to extract the not-so-famous players. When we compare $C^5_{0.94}$ and C^5_1, although most names are similar, some names appear in C^5_1 but not in $C^5_{0.95}$, such as Charles Barkley (another great NBA players). If we change $C^5_{0.95}$ to $C^5_{0.90}$, then Charles Barkley re-appears again. This shows that both k and p are useful in extraction. Finally, when $k = 1$ and $p = 1$, there are 132 records (67 players) extracted. When we apply [5], it extracts one record, Moses Malone, when $6 \leq K \leq 7$ (according to [5], it is meaningless to set $K \leq N/2$, so we do not test these cases), extracts 17 records when $K = 8$, extracts 58 records when $K = 9$, extracts 279 records when $K = 10$. The number of records increases exponentially when K increases linearly. We cannot have much control over the number of records to be extracted. However, in C^k_p we can set different value of p so that we can have more control about the number of records to be extracted. E.g. when $k = 11$ and $p = 0.99$, we can extract 19 records in C^k_p. In addition, Table 4 shows the records extracted by K-dominant skyline when $K = 9$. The records extracted by K-dominant skyline approach are not similar to ours. It extracts fewer players but more records. For example, Allen Iverson and Kobe Bryant, two very great NBA players, do not appear in K-dominant skyline when $K \leq 10$. This is because both players are shooters and do not perform very outstanding in rebound and block. It is very difficult for them to K-dominate other players for whatever K. This is why they cannot be extracted in K-dominant skyline. However, for our proposed work, we consider vertical relationships rather than only horizontal relationships.

5 Summary and Conclusion

We propose a novel concept called k-dominant p-core skyline for extracting interesting points from a skyline. k denotes the number of dimensions that a point has to dominate-back and *p* denotes the fraction of points in a given dimension that a point has to dominate-back. This work is different from the existing work as we further consider the vertical relationships among points. To the best of our knowledge, we are the first mover in this area.

References

1. Fung, G.P.C., Lu, W., Du, X.: Dominant and k nearest probabilistic skylines. In: Proceedings of the 14th International Conference on Database Systems for Advanced Applications, DASFAA 2009 (2009)
2. Agrawal, R., Wimmers, E.L.: A framework for expressing and combining preferences. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, SIGMOD 2000 (2000)
3. KieBling, W.: Foundations of preferences in database systems. In: Proceedings of the 28th International Conference on Very Large Data Bases, VLDB 2002 (2002)
4. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J.X., Zhang, Q.: Efficient computation of the skyline cube. In: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB 2005 (2002)
5. Chan, C.Y., Jagadish, H.V., Tan, K.L., Tung, A.K., Zhang, Z.: Finding k-dominant skylines in high dimensional space. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, SIGMOD 2006 (2006)
6. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: an online algorithm for skyline queries. In: Proceedings of the 28th Very Large Database Conference, VLDB 2002 (2002)
7. Chan, C.Y., Jagadish, H.V., Tan, K.L., Tung, A.K., Zhang, Z.: On high dimensional skylines. In: Proceedings of the 10th International Conference on Extending Database Technology, EDBT 2006 (2006)
8. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Transactions on Database Systems (TODS) 30(1), 41–82 (2005)
9. Zhang, Z., Guo, X., Lu, H., Tung, A.K.H., Wang, N.: Discovering strong skyline points in high dimensional spaces. In: Proceedings of the 14th ACM International Conference on Information and Knowledge Management, CIKM 2003 (2005)
10. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: The k most representative skyline operator. In: Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007 (2007)
11. Mouratidis, K., Bakiras, S., Papadias, D.: Continuous monitoring of top-k queries over sliding. In: Proceedings of the 2006 ACM SIGMOD international conference on Management of Data, SIGMOD 2006 (2006)
12. Das, G., Gunopulos, D., Koudas, N., Sarkas, N.: Ad-hoc top-k query answering for data streams. In: Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB 2007 (2007)
13. Bohm, C., Ooi, B.C., Plant, C., Yan, Y.: Efficiently processing continuous k-nn queries on data streams. In: Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007 (2007)
14. Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th International Conference on Data Engineering, ICDE 2001 (2001)
15. Ramakrishnan, R., Gehrke, J.: DatabaseManagement Systems, 3rd edn. McGraw-Hill, New York (2003)
16. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, SIGMOD 2003 (2003)

Transitivity-Preserving Skylines for Partially Ordered Domains

Henning Köhler, Kai Zheng, Jing Yang, and Xiaofang Zhou

The University of Queensland, Brisbane, Australia
{henning,kevin,jingyang,zxf}@itee.uq.edu.au

Abstract. The *skyline* of a set P of multi-dimensional points (tuples) consists of those points in P for which no clearly better point in P exists, using component-wise comparison on domains of interest. The guiding idea is to prune large data sets to a more manageable size, while ensuring that points of interest are preserved. However, when domains are only partially ordered, it easily happens that the skyline is nearly as large as the original set (or at least of the same order of magnitude), since most of the time points are incomparable in at least some dimension.

To obtain a smaller, more useful skyline set which better reflects actual user preferences, we propose a richer notion of dominance, based on two assumptions: that preference specifications are often incomplete, and that actual preferences are transitive.

1 Introduction

There are many applications where a user is interested in viewing the 'best' objects chosen from a large collection, based on multiple criteria, e.g. price and milage for used cars. There are multiple ways to approach this problem. Top-k queries [5] require a user to define a ranking function over the object collection, and return the k top-ranked objects. In contrast, skylines [4] only require users to express their preferences for each domain, e.g. price and milage should both be low. They then return all objects for which no object exists that is "clearly better": at least as good on every domain and strictly better in at least one.

The resulting skyline can provide a user with a better understanding of the trade-offs involved, without requiring any precise ranking functions to be specified. However, skyline sets can grow very large, for different reasons:

– large and/or anti-correlated data sets
– too many domains
– preferences are only partial

As a result the user is overwhelmed with information, which is particularly aggravating if only a small portion of the skyline is of actual interest to the user.

In the case of large and/or anti-correlated data sets, we can expect that many skyline points are really interesting, but often very similar. Here sampling and clustering approaches [11] can provide an overview. In [10] points are chosen to maximize the number of points dominated.

When skylines grow large because many domains are considered or because preferences are partial, the expectation is that only a small portion of the skyline points will really be interesting. Thus the usefulness of any filtering method must be judged by how well it manages to eliminate 'bad' points without eliminating 'good' points as well. Here the classic notion of Pareto dominance often fairs poorly, since it is too restrictive.

A number of approaches have been suggested for dealing with large numbers of domains. k-dominance [6] only requires that a point is better than or equal to another in at least k dimensions. The ε-skyline [12] allows a dominating point to be worse in some dimensions as well, though only by a small pre-defined ε value. Approximately dominating representatives [9] allow dominance by being no worse than a factor $1 + \varepsilon$ in any dimension, and the objective is to find small sets which approximately dominate all points. In [8] user-defined preference rankings between domains allow dominance by being better on preferred domains. User-defined preference comparisons between instances on subspaces are considered in [1]. The strong skyline [14] combines skyline points from subspaces where the skyline is small. Skyline frequency [7] ranks skyline points by how often they appear as skyline points in subspaces, while top-k-skyline [13] ranks them by the number of points they dominate in the given data set.

For partially ordered domains, Balke et. al. [3] proposed a new notion of dominance, called *weak Pareto dominance*, which treats incomparability on a domain as equality. While this idea is easy to understand, and results in a much smaller skyline, it suffers from cyclicity and intransitivity (as do k-dominance and ε-skyline). As already pointed out in [2], such properties are undesirable. They contradict the intuition that preferences are intrinsically transitive, and can lead to unexpected behavior. To avoid this, we will propose a new notion of dominance for partially ordered domains, based on the assumptions that

– preference specifications may be incomplete, but
– actual (hidden) preferences are transitive

Theoretical arguments as well as experimental evaluation (which had to be omitted due to space constraints) suggest that this leads to more useful skyline sets, i.e., that our new dominance notion reflects user preferences more closely than classic or weak Pareto dominance. As a nice side-effect, transitivity of dominance allows us to employ efficient algorithms for computing it.

2 Dominance

At the heart of the skyline problem lies the notion of dominance, indicating that a tuple $A = (a_1, \ldots, a_n)$ is strictly better than a tuple $B = (b_1, \ldots, b_n)$. Here the traditional definition, called Pareto dominance, is that $a_i \leq_i b_i$ for all i and $a_j <_j b_j$ for some j. We will use a different but equivalent formalization, which will make later comparison of dominance notions more elegant.

Definition 1 (Pareto Dominance).
The Pareto ordering *is defined as*

$$A \leq_P B :\Leftrightarrow a_i \leq_i b_i \text{ for all } i \in [1, n]$$

with the reduction

$$A <_P B :\Leftrightarrow A \leq_P B \wedge B \not\leq_P A$$

We say that A dominates *B if $A <_P B$.*

To reduce the size of the skyline set when domains are only partially ordered, Balke et. al. [3] proposed the notion of *weak Pareto dominance.*

Definition 2 (Weak Pareto Dominance).
The weak Pareto relation *is defined as*

$$A \leq_{WP} B :\Leftrightarrow a_i \not>_i b_i \text{ for all } i \in [1, n]$$

with the reduction

$$A <_{WP} B :\Leftrightarrow A \leq_{WP} B \wedge B \not\leq_{WP} A$$

We say that A weakly dominates *B if $A <_{WP} B$.*

To better distinguish the two notions, we will sometimes refer to Pareto dominance as *strong Pareto dominance*. Strong and weak Pareto dominance coincide for totally ordered (also called linear ordered) domains, but for domains with only partial orders weak Pareto dominance allows elements to be incomparable as well, resulting in a much smaller skyline set.

In essence, the two approaches differ in how they view incomparable elements. Strong Pareto dominance takes the stance that when in doubt (i.e., if two points are incomparable in some dimension), return both points. On the other hand, weak Pareto dominance simply ignores dimensions where elements are incomparable (treating incomparable elements as equivalent). In particular, when a domain has no preferences specified at all, strong Pareto dominance handles this as ‘comparable values must be identical on that domain’, whereas weak Pareto dominance treats it as ‘we don’t care about this domain’ and returns the skyline w.r.t. the remaining domains only. We believe that the latter interpretation is more suitable in many situations.

However, while weak Pareto dominance is an intuitive way for comparing tuples, it suffers from two significant problems:

– lack of transitivity: from $A <_{\mathrm{WP}} B$ and $B <_{\mathrm{WP}} C$ it does *not* follow that $A <_{\mathrm{WP}} C$.
– cycles: it can happen that $A <_{\mathrm{WP}} B <_{\mathrm{WP}} C <_{\mathrm{WP}} A$.

In addition to the algorithmic challenges that non-transitivity brings, skylines w.r.t. weak Pareto dominance (or any other intransitive, cyclic dominance notion) have unexpected properties. In particular, the addition of new points can lead to the removal of points from the skyline, without any of the new points getting added. The skyline set can even be empty.

Example 1. Consider a database which stores information on cars: make (M), color (C) and transmission type (T). For each domain, a user provides us with some partial order relation expressing his preferences:

$$\begin{aligned} \leq_M &:= \{BMW < Kia, Mercedes < Daewoo\} \\ \leq_C &:= \{white < grey < black\} \\ \leq_T &:= \{automatic < manual\} \end{aligned}$$

Each domain may contain further elements as well which don't occur in the preferences. Consider the following sample database:

$Mercedes$	red	$manual$
BMW	$grey$	$automatic$
$Daewoo$	$white$	$semi-auto$
$Daewoo$	$grey$	$manual$
Kia	$black$	$semi-auto$
$Ford$	$black$	$automatic$

Using strong Pareto dominance, every point is a skyline point, since no point dominates any other. With weak Pareto dominance, we obtain the following dominance relationship, abbreviating values by their initial characters:

$$\begin{tikzcd}
(M,r,m) \arrow[rr, bend left] \arrow[d] & (B,g,a) \arrow[l] \arrow[dl] \arrow[d] \arrow[dr] & (D,w,s) \arrow[l] \arrow[dll] \arrow[dl] \arrow[d] \\
(D,g,m) \arrow[r] & (K,b,s) & (F,b,a)
\end{tikzcd}$$

Here every point is dominated by at least one other, due to the cycle between the first three tuples, resulting in an empty skyline set. Clearly, both approaches are problematic. A more reasonable skyline set might be the following:

$$\{(M,r,m),(B,g,a),(D,w,s)\}$$

Intuitively, the tuples above must be in the skyline since Mercedes and BMW are preferred makes, and white is the preferred color. The remaining three tuples should be considered worse than the grey BMW automatic (and possibly others as well), and thus not appear in the skyline set.

To further illustrate the unexpected behavior of weak Pareto skylines caused by intransitivity, even if no cycles are present, consider the sub-relation containing only (M,r,m) and (F,b,a). Its weak Pareto skyline is the entire relation. If we now add the tuple (D,w,s), the tuple (F,b,a) disappears from the skyline, without the new tuple getting added. Also, it seems odd that the new skyline, consisting only of (M,r,m), does not dominate every point in the original set.

In the following we will propose a new notion of dominance which is transitive, and consequently also acyclic. Our notion lies between strong and weak Pareto dominance, and produces a more suitable skyline set than the two extremes, in the sense that it captures user preferences more accurately.

2.1 Transitivity-Preserving Extensions

In order to define our dominance notions, we will introduce the concept of *transitivity-preserving extension.* This is first done for arbitrary relations, and will then be applied to the partial orders on the domains. We work with quasi-orderings (also called pre-orderings) rather than partial orders since they are more flexible and arise naturally: the transitivity-preserving extension of a partial order is only a quasi-ordering.

Definition 3 (quasi-ordering). *A relation $\leq$ is a* quasi-ordering *if it is reflexive and transitive.*

In the following let U be our universe, and $r \subseteq U \times U$ a relation over U.

Definition 4 (transitivity-preserving extension). *The* ancestors *and* descendants[1] *of an element $a \in U$ w.r.t. r are*

$$anc_r(a) := \{x \in U \mid (x,a) \in r \wedge (a,x) \notin r\}$$
$$desc_r(a) := \{x \in U \mid (a,x) \in r \wedge (x,a) \notin r\}$$

Furthermore we define the transitivity-preserving extension *of r as*

$$TX(r) := \{(a,b) \in U \times U \mid anc_r(a) \subseteq anc_r(b) \wedge desc_r(b) \subseteq desc_r(a)\}$$

The guiding idea behind Definition 4 is the following: If we are to consider a to be smaller than b, then every value smaller than a had better be smaller than b as well. Similarly, every value larger than b must also be larger than a.

Figure 1 shows transitive relations r_1, r_2 and their transitivity-preserving extensions. Note that we omitted loops in the TX($r_{1/2}$)-graphs for readability.

$$\begin{tikzcd}
a \arrow[d] \arrow[dr] & b & a \arrow[r] \arrow[d] \arrow[dr] & b \arrow[d] \arrow[dl] & a \arrow[d] \arrow[dr] & b & a \arrow[r] \arrow[d] \arrow[dr] & b \arrow[d] \\
d & c & d \arrow[r] & c \arrow[l, bend left] & d \arrow[r] & c & d \arrow[r] & c \\
r_1 & & \mathrm{TX}(r_1) & & r_2 & & \mathrm{TX}(r_2) &
\end{tikzcd}$$

Fig. 1. transitive relations and transitivity-preserving extensions

If we interpret arcs as 'better than' relationships, we get the following intuition for TX(r_1): a is better than b since in r_1 it is better than c and d, while b is better than c and d since in r_1 these are both worse than a. Finally c and d are equally good, since in r_1 they are both worse than a. Similar for TX(r_2).

The transitivity-preserving extensions of the domain orderings from Example 1 can be found in Example 2. We named the relation TX(r) transitivity-preserving extension, since it is transitive and extends transitive relations r:

[1] For intransitive relations the names 'ancestor' and 'descendent' may be misleading. However, the preference relations we use are assumed to be transitive, so terms like 'predecessor' and 'successor' may be even more misleading here.

Theorem 1. *$TX(r)$ is a quasi-ordering (for arbitrary relations r). If r is transitive then $r \subseteq TX(r)$ and $r^{<} \subseteq TX(r)^{<}$, where $r^{<}$ (respectively $TX(r)^{<}$) denotes the reduction $r^{<} := \{(a,b) \in r \mid (b,a) \notin r\}$.*

Note that transitivity of r is necessary for TX(r) to extend r (hence the name transitivity *preserving* extension). E.g. for $r = \{(a,b),(b,c)\}$ on $\{a,b,c\}$ we get TX$(r) = \{(a,a),(b,b),(c,c)\}$, which does not contain r.

2.2 TX-dominance

We can now define our new dominance notion as follows:

Definition 5 (TX-dominance). *The* TX-ordering *is the Pareto ordering over the transitivity-preserving extensions of the domain orderings:*

$$\leq_{TX} := \ P(TX(\leq_1), \ldots, TX(\leq_n))$$

We say that A TX-dominates *B if $A <_{TX} B$.*

We get the following relationship between strong, weak and TX-dominance:

Theorem 2. $\leq_P \subseteq \leq_{TX} \subseteq \leq_{WP}$

The three skyline notions differ in how they interpret values which are incomparable w.r.t. preference specifications. For each such pair $a, b \in D$ we have four options for comparing them: $a < b$, $a > b$, $a \equiv b$ and incomparability $a||b$. Strong Pareto dominance always interprets them as truly incomparable ($a||b$), while weak Pareto dominance always treats them as equivalent ($a \equiv b$). For TX-dominance any of the four interpretations is possible, and the choice depends on how a and b compare to other values.

Example 2. Consider the setup from Example 1. Domain elements not occurring in the preference specification (e.g. Ford) are always equivalent in the transitivity-preserving extension, so we denote them all by '$*$'. This gives us the quasi-orderings

$$TX(\leq_M) = \{\, (BMW||Mercedes) < * < (Kia||Daewoo)\,\}$$
$$TX(\leq_C) = \{\, white < (grey||*) < black\,\}$$
$$TX(\leq_T) = \{\ automatic < * < manual\ \}$$

resulting in the following TX-dominance relationship:

$$\begin{tikzcd}
(M,r,m) & (B,g,a) \arrow[dl] \arrow[d] \arrow[dr] & (D,w,s) \arrow[dll] \\
(D,g,m) & (K,b,s) & (F,b,a) \arrow[l]
\end{tikzcd}$$

This in turn leads to the skyline

$$\{(M,r,m),(B,g,a),(D,w,s)\}$$

which we argued to be a more reasonable answer earlier.

3 Conclusion

We have motivated why often neither strong Pareto nor weak Pareto dominance are suitable for generating good skylines when some domains are only partially ordered. Strong Pareto dominance returns too many points, while weak Pareto dominance misses too many points of interest, effectively generating the skyline on numerical domains only. To remedy this situation, we introduced the notion of TX-dominance, based on the intuitions that preference specifications may be incomplete, and that actual preferences are transitive.

References

1. Balke, W.-T., Güntzer, U., Lofi, C.: Eliciting matters - controlling skyline sizes by incremental integration of user preferences. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 551–562. Springer, Heidelberg (2007)
2. Balke, W.-T., Güntzer, U., Siberski, W.: Exploiting indifference for customization of partial order skylines. In: IDEAS, pp. 80–88 (2006)
3. Balke, W.-T., Siberski, W., Güntzer, U.: Getting prime cuts from skylines over partially ordered domains. In: BTW, pp. 64–81 (2007)
4. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE, pp. 421–430 (2001)
5. Carey, M.J., Kossmann, D.: On saying "enough already!" in sql. In: SIGMOD, pp. 219–230 (1997)
6. Chan, C.Y., Jagadish, H.V., Tan, K.-L., Tung, A.K.H., Zhang, Z.: Finding k-dominant skylines in high dimensional space. In: SIGMOD, pp. 503–514 (2006)
7. Chan, C.Y., Jagadish, H.V., Tan, K.-L., Tung, A.K.H., Zhang, Z.: On high dimensional skylines. 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. 478–495. Springer, Heidelberg (2006)
8. Georgiadis, P., Kapantaidakis, I., Christophides, V., Nguer, E.M., Spyratos, N.: Efficient rewriting algorithms for preference queries. In: ICDE, pp. 1101–1110 (2008)
9. Koltun, V., Papadimitriou, C.H.: Approximately dominating representatives. In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, pp. 204–214. Springer, Heidelberg (2005)
10. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: The k most representative skyline operator. In: ICDE, pp. 86–95 (2007)
11. Tao, Y., Ding, L., Lin, X., Pei, J.: Distance-based representative skyline. In: ICDE, pp. 892–903 (2009)
12. Xia, T., Zhang, D., Tao, Y.: On skylining with flexible dominance relation. In: ICDE, pp. 1397–1399 (2008)
13. Yiu, M.L., Mamoulis, N.: Efficient processing of top-k dominating queries on multi-dimensional data. In: VLDB, pp. 483–494 (2007)
14. Zhang, Z., Guo, X., Lu, H., Tung, A.K.H., Wang, N.: Discovering strong skyline points in high dimensional spaces. In: CIKM, pp. 247–248 (2005)

Finding the Most Desirable Skyline Objects

Yunjun Gao[1], Junfeng Hu[2], Gencai Chen[1], and Chun Chen[1]

[1] College of Computer Science and Technology, Zhejiang University
{gaoyj,chengc,chenc}@zju.edu.cn
[2] School of Computing, National University of Singapore
hujunfeng@comp.nus.edu.sg

Abstract. This paper introduces a new operator, namely the *most desirable skyline object* (MDSO) query, to identify manageable size of *truly interesting* skyline objects. Given a set of multi-dimensional objects and an integer k, a MDSO query retrieves the *most preferable* k skyline objects, based on the newly defined ranking criterion that considers, for each skyline object s, the number of objects dominated by s and their accumulated (potential) weight. We present the ranking criterion, formalize the MDSO query, and develop two algorithms for processing MDSO queries assuming that the dataset is indexed by a traditional data-partitioning index. Extensive experiments demonstrate the performance of the proposed algorithms.

1 Introduction

Given a set P of multi-dimensional data objects, a *skyline query* returns all the data objects from P, called *skyline objects*, which are *not dominated* by any other objects in P. Here, an object p *dominates* another object p' iff p is not worse than p' in all dimensions and strictly better than p' in at least one dimension. The skyline query is useful in many real-life applications. Consider, for instance, a classical example of *hotel reservation system*. Figure 1 illustrates this case in a 2-dimensional space, where each point corresponds to a hotel record. The room *price* of a hotel is represented as the x-axis, and the y-axis captures its *distance* to the beach. Hotel p_2 dominates p_5 since the former is cheaper and closer to the beach. As hotels p_1, p_2, p_3, and p_4 are not dominated by any other hotel, they form the *skyline* of dataset $P = \{p_1, p_2, \ldots, p_8\}$.

Since the skyline operator was first introduced to the database community in [2], it has received considerable attention. A large number of algorithms [2, 6, 9, 11] have been proposed for the efficient *traditional* skyline computation. Other versions include *subspace* skyline retrieval [10], *reverse* skyline query [5], etc. However, the skyline query may output an overwhelming number of skyline objects, and thus no longer offer any interesting insights, especially in high dimensional spaces. To address this, several efforts have been proposed in the literature. In particular, they control the size of skyline objects either by *relaxing the dominance relationship* [3, 4, 8, 12] or by *integrating user-specific preference* [7]. In this paper, we introduce a new operator, namely, the *most desirable skyline object* (MDSO) query, to identify *manageable* size of *truly interesting* skyline objects.

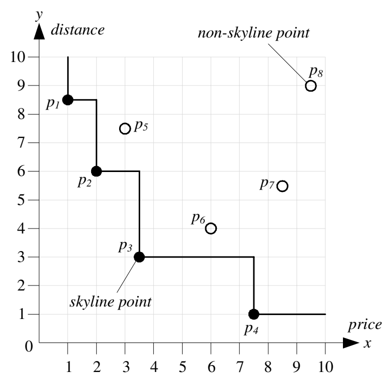


Fig. 1. Example of dataset and skyline

Given a set of multi-dimensional objects and an integer *k*, a MDSO query retrieves the *most preferable k* skyline objects, based on the *ranking criterion* (defined in Definition 6) that considers, for each skyline object *s*, the number of objects dominated by *s* and their accumulated (potential) weight. The MDSO query operator is different from those studied in [3, 4, 7, 8, 12]. Hence, the existing techniques are not directly applicable to tackle the MDSO query efficiently. Motivated by this, in this paper, we develop two efficient algorithms, namely *Cell Based Algorithm* (CB) and *Sweep Based Algorithm* (SB), to obtain the most desirable *k* skyline objects. Our methods are based on a conventional data-partitioning index (i.e., R*-tree [1]) and do not require any preprocessing. Extensive experiments demonstrate that our proposed algorithms are efficient and scalable.

2 Problem Statement

Let P be a set of data objects in an n-dimensional space $D = (d_1, d_2, \ldots, d_n)$, where d_i ($i \in [1, n]$) is the domain of number. For any object $p \in P$, we use $p.d_i$ to denote the i-th dimensional value of p. Assume that there exists a total order relationship, either '<' or '>', on each dimension. Without loss of generality, in this paper, we consider '<' relation, i.e., smaller values are more preferable.

Definition 1 (Dominance). *For any two objects p, $p' \in P$, p is said to* dominate *p', denoted by $p \prec p'$, iff (i) $\forall\, d_i \in D$, $p.d_i \leq p'.d_i$, and (ii) $\exists\, d_j \in D$, $p.d_j < p'.d_j$.*

Definition 2 (Skyline Object, Skyline, and Non-skyline Object). *An object $p \in P$ is a* skyline object *s iff p is not dominated by any other object p' ($\neq p$) $\in P$, i.e., $\nexists\, p' \in P - \{p\}$, $p' \prec p$. The* skyline *of P is the set S of skyline objects on dataset P in space D. An object p'' is a* non-skyline object *ns iff there exists an object p' ($\neq p''$) $\in P$ dominating p'', i.e., $\exists\, p' \in P - \{p''\}$, $p' \prec p''$.*

Definition 3 (Dominating Score). *Let S be the skyline of P, for a skyline object $s \in S$, the* dominating score *of s, denoted by $\mu(s)$, could be defined as:*

$$\mu(s) = |\{ns \in P - S \mid s \prec ns\}| \tag{1}$$

In words, the score $\mu(s)$ is the number of non-skyline objects dominated by a skyline object s. The higher the $\mu(s)$ is, the more interesting the skyline object is, as pointed out in [8]. In Figure 1, for instance, p_3 is more desirable than p_1 as $\mu(p_3) = 3$ and $\mu(p_1) = 1$. Thus, we can derive a natural ordering of skyline objects, according to dominating score. Nevertheless, when two skyline objects share the same dominating score, the tie needs to be broken. Towards this, one possible approach is to request users to provide some weight assignments for their preferred attributes. Unfortunately, providing such weight assignment is not always easy without any initial knowledge about the data. On the other hand, as stated in [2], the aim of providing the skyline to the users is to help them determine the weight assignment. Another possible work-around is to take the potential weight of every non-skyline object into consideration, by computing the number of objects dominated by it. However, this method incurs expensive computational cost, as it has to count the number of objects that are dominated by each non-skyline object.

Definition 4 (Dominated Score). *Let S be the skyline of P, for a non-skyline object ns* $\in P - S$*, the* dominated score *of ns, denoted by* $\omega(ns)$*, could be defined as:*

$$\omega(ns) = \frac{1}{\left|\{s \in S \mid s \prec ns\}\right|} \tag{2}$$

In fact, the dominated score of a non-skyline object considers the number of skyline objects dominating it. Intuitively, a non-skyline object might have larger weight if it is dominated by as few skyline objects as possible, indicating that it may dominate more other non-skyline objects. As an example, in Figure 1, the weight of p_5 is greater than that of p_8, since $\omega(p_5) = 1$ and $\omega(p_8) = 1/4$. Therefore, we define the preference score of a skyline object based on the intuition: a skyline object is more interesting if it dominates as many non-skyline objects with higher dominated scores as possible.

Definition 5 (Preference Score). *Let S be the skyline of P, for a skyline object* $s \in S$*, the* preference score *of s, denoted by* $\tau(s)$*, is defined as:*

$$\tau(s) = \sum_{ns' \in \{ns \in P - S \mid s \prec ns\}} \omega(ns') \tag{3}$$

For example, in Figure 1, p_2 is more preferred than p_4 as $\tau(p_2) = 5/4$ and $\tau(p_4) = 3/4$, although $\mu(p_2) = \mu(p_4) = 2$. Actually, the preference score of a skyline object takes into account the accumulated (potential) weight of all non-skyline objects dominated by it. A skyline object s having a higher score $\tau(s)$ might be more preferable for users. Based on equations (1), (2), and (3), the *ranking criterion* is formalized as follows:

Definition 6 (Ranking Criterion). *Let S be the skyline of P, for two skyline objects s,* $s' \in S$*, s is* more desirable than *s', denoted by* $s \vdash s'$*, iff*

$$s \vdash s' \Leftrightarrow \begin{cases} \mu(s) > \mu(s') \\ \mu(s) = \mu(s') \wedge \tau(s) > \tau(s') \end{cases} \tag{4}$$

We now present the formal definition of the MDSO query.

Definition 7 (Most Desirable Skyline Object Query). *Given a set P of data objects, an integer k* (≥ 1)*, and let S be the skyline of P, the* most desirable skyline object (MDSO) query *retrieves the set Res of skyline objects, such that* (*i*) *Res contains k skyline objects, i.e.,* $|Res| = k$*; and* (*ii*) *none of object* $p' \in S - Res$ *is superior to any result object* $p \in Res$ *according to the ranking criterion* (*Definition 6*)*, i.e.,* $\forall\, p \in Res$ *and* $p' \in S - Res$, $\mu(p) > \mu(p')$ or $\mu(p) = \mu(p') \wedge \tau(p) > \tau(p')$. *Note that when* $|S| \le k$, *S is the result set.*

Take Figure 1 as an example again. The MDSO query on the data of Figure 1 returns p_3 (with $\mu(p_3) = 3$ and $\tau(p_3) = 7/4$) and p_2 (with $\mu(p_2) = 2$ and $\tau(p_2) = 5/4$) as the most preferable 2 skyline objects.

3 Algorithms for MDSO Queries

In this section, we propose two algorithms for processing MDSO queries. For the following presentation, a running example, as shown in Figure 2, is employed, where the two-dimensional (2D) data point set of Figure 2(a) is organized in the R-tree of Figure 2(b) with node capacity set to three. Assume that the number *k* of required skyline objects is 2 (i.e., $k = 2$). The query result contains points p_9 and p_2.

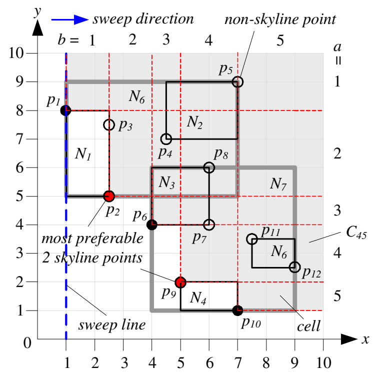
3.1 Cell Based Algorithm

Once the skyline *S* of a specified data set *P* is obtained, the region dominated by all skyline objects in *S* can be divided into a lot of *cells*. For simplicity, we consider the partition of cells in a 2D space (*x*, *y*). The presented concepts, however, can be easily extended to high dimensional spaces.

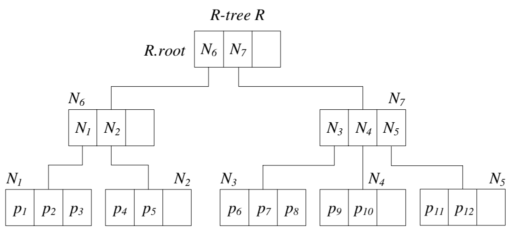
Suppose $S = \{s_1, s_2, \ldots, s_m\}$ is the skyline of a given data set *P* in a 2D space. The region dominated by skyline objects in *S* can be partitioned into $m \cdot (m + 1) / 2$ cells $\{C_{ab} \mid 1 \le a \le b \le m\}$. The lower-left corner and the upper-right corner of every cell C_{ab} are $(s_b.x, s_a.y)$ and $(s_{b+1}.x, s_{a-1}.y)$, respectively. Note that, when $a = 1$ ($b = m$), $s_0.y$ ($s_{m+1}.x$) is defined as the maximal value of *y* (*x*) coordinate in the data space. As depicted in Figure 2(a), for instance, the shadowed region dominated by $\{p_1, p_2, p_6, p_9, p_{10}\}$ is divided into 15 cells.

Data objects that are within a single cell, but not on the boundaries of the cell, have *the same* (*potential*) *weight*. For example, in Figure 2(a), points p_{11} and p_{12} contained in an R-tree node N_6 are dominated by the same skyline points p_9 and p_{10} since they are located inside cell C_{45}. Nevertheless, when an R-tree node intersects *multiple* cells, the data objects included in the R-tree node may be dominated by *different* skyline objects. Take Figure 2(a) as an example again. As the R-tree node N_2 crosses four cells, point p_4 in N_2 are only dominated by skyline points p_2 and p_6, whereas point p_5 in N_2 is dominated by skyline points p_2, p_6, p_9, and p_{10}.

Our first approach, namely *Cell Based Algorithm* (CB), utilizes the above cell properties. The basic idea of CB is to compute the skyline *S* using BBS algorithm [9],



(a) The data point placement



(b) The R-tree *R*

Fig. 2. A running example

and then, for each skyline object $s \in S$, calculate its dominating score $\mu(s)$ and preference score $\tau(s)$ respectively, and return the top-k skyline objects in S according to our proposed ranking criterion (Definition 6).

Back to the running example depicted in Figure 2. First, CB uses BBS algorithm to compute skyline $S = \{\langle p_9, 0, 0\rangle, \langle p_2, 0, 0\rangle, \langle p_{10}, 0, 0\rangle, \langle p_6, 0, 0\rangle, \langle p_1, 0, 0\rangle\}$. Then, for each skyline object $s \in S$, it calculates $\mu(s)$ and $\tau(s)$ by traversing the R-tree R, after which $S = \{\langle p_9, 5, 61/30\rangle, \langle p_2, 4, 61/30\rangle, \langle p_{10}, 3, 6/5\rangle, \langle p_6, 4, 23/15\rangle, \langle p_1, 1, 1/5\rangle\}$. After sorting, points p_9 and p_2 are output as the most desirable 2 skyline points.

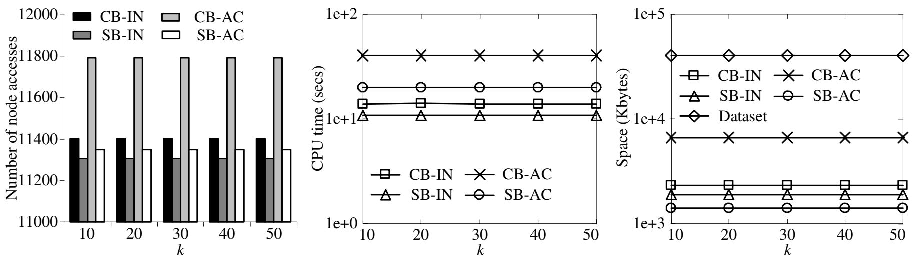
3.2 Sweep Based Algorithm

CB may access some entries (R-tree nodes or data objects) *two* times. Hence, it is not efficient in terms of the I/O cost (i.e., the number of node accesses) and CPU time, especially in high dimensional spaces. Motivated by this, we present an alternative, namely *Sweep Based Algorithm* (SB). The main idea of SB is to identify skyline objects and calculate their dominating scores and preference scores simultaneously, via a *single* traversal of the R-tree R used to organize dataset P.

Again, back to our running example. First, point p_1 is encountered by the sweep line (blue dashed line in Figure 2(a)). Since the current skyline S is empty, p_1 is added to $S = \{\langle p_1, 0, 0\rangle\}$ as the first skyline point. The second point accessed is p_2. It is also inserted into $S = \{\langle p_1, 0, 0\rangle, \langle p_2, 0, 0\rangle\}$, as p_2 is not dominated by p_1. Then, SB visits p_3 and updates S to $\{\langle p_1, 0, 0\rangle, \langle p_2, 1, 1\rangle\}$. The algorithm proceeds in the same manner until all data points are accessed, after which $S = \{\langle p_1, 1, 1/5\rangle, \langle p_2, 4, 61/30\rangle, \langle p_6, 4, 23/15\rangle, \langle p_9, 5, 61/30\rangle, \langle p_{10}, 3, 6/5\rangle\}$. Finally, SB reports points p_9 and p_2 as the final query result after sorting S.

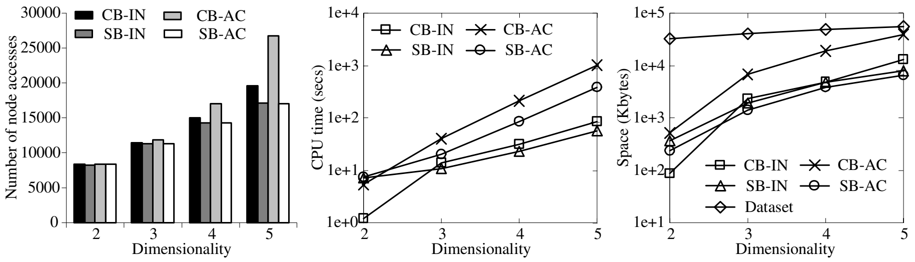
4 Experimental Evaluation

This section experimentally evaluates the performance of our proposed algorithms in terms of both efficiency and scalability. Our experimentation used synthetic datasets.



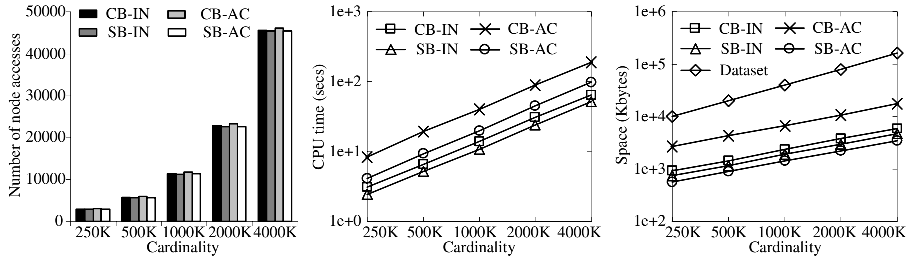
(a) I/O cost on IN & AC (b) CPU time on IN & AC (c) Space on IN & AC

Fig. 3. Performance vs. *k* (*dim* = 3, *CN* = 1000K)



(a) I/O cost on IN & AC (b) CPU time on IN & AC (c) Space on IN & AC

Fig. 4. Performance vs. dimensionality *dim* (*k* = 30, *CN* = 1000K)



(a) I/O cost on IN & AC (b) CPU time on IN & AC (c) Space on IN & AC

Fig. 5. Performance vs. cardinality *CN* (*k* = 30, *dim* = 3)

Specifically, we created *Independent* (IN) and *Anti-correlated* (AC) datasets with dimensionality varied from 2 to 5 and cardinality in the range [250K, 4000K]. Our generation follows exactly the description in [2]. Every dataset is indexed by an R*-tree [1] with a disk page size of 4096 bytes. The number of node/page accesses (i.e., I/O cost), CPU time, and the maximum space consumption are employed as the major performance metrics. All algorithms were implemented in C++, and all experiments were conducted on the PC with an Intel Core 2 Duo 2.13GHz CPU and 2GB RAM, running Microsoft Windows XP Professional Edition.

We investigate the performance of our proposed algorithms under several factors, including *k*, dimensionality *dim*, and cardinality *CN*. The results are plotted in Figures 3 through 5. To summarize, from these experimental results on synthetic datasets, we can conclude that: (i) the I/O overhead of SB is always less than that of CB; (ii) in terms of CPU time and space cost, CB is comparable to SB in a 2D space, while SB clearly outperforms CB in high dimensions; and (iii) the space requirement of both CB and SB is negligible compared to the dataset size.

5 Conclusion

The skyline of a dataset might have an overwhelming number of skyline objects. Returning all of them may make it difficult for a user to make a good, quick selection. In this paper, we introduce a new operator, namely, the most desirable skyline object (MDSO) query, for finding manageable size of the most preferable/interesting skyline objects. First, we formalize the ranking criterion and the MDSO query, respectively. Then, two algorithms, i.e., CB and SB, are developed for efficiently processing MDSO queries. Finally, extensive experimental evaluations using synthetic datasets demonstrate that our proposed algorithms are efficient and scalable.

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. Chan, C.-Y., Jagadish, H.V., Tan, K.-L., Tung, A.K.H., Zhang, Z.: Finding k-Dominant Skylines in High Dimensional Space. In: SIGMOD, pp. 503–514 (2006)
4. Chan, C.-Y., Jagadish, H.V., Tan, K.-L., Tung, A.K.H., Zhang, Z.: On High Dimensional Skylines. 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. 478–495. Springer, Heidelberg (2006)
5. Dellis, E., Seeger, B.: Efficient Computation of Reverse Skyline Queries. In: VLDB, pp. 291–302 (2007)
6. Kossmann, D., Ramsak, F., Rost, S.: Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In: VLDB, pp. 275–286 (2002)
7. Lee, J., You, G.-W., Hwang, S.-W.: Personalized Top-k Skyline Queries in High-Dimensional Space. Inf. Syst. 34(1), 45–61 (2009)
8. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting Stars: The k Most Representative Skyline Operator. In: ICDE, pp. 86–95 (2007)
9. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive Skyline Computation in Database Systems. ACM Trans. Database Syst. 30(1), 41–82 (2005)
10. Pei, J., Yuan, Y., Lin, X., Jin, W., Ester, M., Liu, Q., et al.: Towards Multidimensional Subspace Skyline Analysis. ACM Trans. Database Syst. 31(4), 1335–1381 (2006)
11. Tan, K.-L., Eng, P.-K., Ooi, B.C.: Efficient Progressive Skyline Computation. In: VLDB, pp. 301–310 (2001)
12. Xia, T., Zhang, D., Tao, Y.: On Skylining with Flexible Dominance Relation. In: ICDE, pp. 1397–1399 (2008)

Multiple Sensitive Association Protection in the Outsourced Database[⋆]

Xiao Jiang, Jun Gao, Tengjiao Wang, and Dongqing Yang

Key Laboratory of High Confidence Software Technologies
Department of Computer Science, Peking University
Beijing, China, 100871
{jiangxiao,gaojun,tjwang,dqyang}@pku.edu.cn

Abstract. With the popularity of the outsourced database, protecting sensitive associations in this environment is greatly desired and receives high attention. Table decomposition based method, which is suited for the current query evaluation of the database engine, provides an alternative to the conventional encryption method. Although some work on table decomposition has been done to handle single sensitive association in data publishing scenario or multiple associations in multiple servers, fewer attempts have been made to deal with multiple associations in single outsourced database. In this paper, we first illustrate that the simple extension of existing work will lead to new kinds of information leakages, and then we propose a novel table decomposition method, which achieves l-diversity, defeats the new information leakages, while at the same time, considers the query efficiency over the decomposed sub-tables. The final experimental results validate the effectiveness and efficiency of our method.

1 Introduction

With the popularity of the outsourced database [1,2], ensuring security of sensitive associations in this environment receives high attention. On the one hand, the outsourced database provides scalable, stable, and low-priced services, as well as migrates the database administration tasks from end users. All these merits lead to a wide application of the outsourced database in various domains. On the other hand, the database service provider might not be fully trusted. As a result, protecting sensitive information from the service provider in addition to unauthorized users comes out to be a crucial challenge.

The conventional way to cope with this problem is encryption. Specifically, all the data are encrypted before being outsourced to the service provider. The encrypted data prevents sensitive information leakages both to the unauthorized users and to the service provider. In addition, the query evaluation over encrypted database has also been

[⋆] This work was supported by the National High Technology Research and Development Program("863"Program) of China under Grant No. 2009AA01Z150, 2007AA01Z191, 2006AA01Z230; the National Natural Science Foundation of China under Grant No. 60873062; the Peking University−Morgan Stanley Research Aid Program. Jun Gao is the contact author for this paper.

extensively studied[7,4,10]. However, the decryption of candidate results shifts much of the burden back to the client, which defeats the goal of data outsourcing, making the approach really expensive to be practical. Besides, the security level partially depends on the key management service, as well as the strength of the cryptographic functions.

Table decomposition based method[3] is an alternative to the encryption method. The basic idea is to partition the table across two (or more) logically independent servers, in such a way that the sensitive associations cannot be reconstructed with the data in any single server. However, it assumes that there is no communication between servers, which is clearly a strong assumption and hard to achieve in real environment.

The table decomposition based approach is also studied in data publishing scenario. The existing work[11] mainly focuses on a single sensitive association between quasi-identifier and the sensitive attribute(donated as $q - s$ association in the following). It splits the original relation into a quasi-identifier table (QIT) and a sensitive table (ST) with an extra group column to keep their correlation. In this way, the data owner can publish his data and allow others to guess the correct association only under a controllable probability. However, the sensitive associations in real life applications are not restricted to the $q - s$ association, and there are always functional dependencies among attributes. It is especially meaningful yet not trivial to make the current method feasible for multiple sensitive associations. As we can see in the following illustration, the straightforward extension will lead to new kinds of information leakages if the sensitive associations are also functional dependencies.

Example 1. Given an employee table as shown in Fig. 1(a) with the following sensitive associations: from *name* to *position*, from *position* to *salary*, and from *name* to *salary*. The result of simple decomposition method is illustrated in Fig. 1(b). We can observe that three sensitive attributes are separated into three sub-tables, and each with an extra *gid* attribute, which is used to join with other sub-tables. The tuples are divided into 2 groups, assigned with *gid* $= 1$ or 2. For simplicity, we call them g_1 and g_2 respectively. Note that this decomposition satisfies 3-diversity, as for any sensitive association, there are 3 distinct sensitive values involved for the dominant set, with a probability not greater than 1/3. Specifically, for the association between *position* and *salary*, if we want to locate the *salary* for *position* =“$manager$”, we can find that 3 distinct *salary*s in g_1 are involved. They are 4000, 6000 and 3000. However, suppose the salary is determined by the position as always the case in real life, the attacker can easily infer that the salary of $salesman$ is 3000, since $salesman$ appears twice in g_2 in the second sub-table, and 3000 is the only salary appears more than once in g_2 in the third sub-table.

This paper attempts to address the protection of multiple sensitive associations in outsourced databases based on table decomposition. Besides making the decomposed sub-tables satisfying l-diversity, this paper also studies how to counter the new kinds of information leakages and to improve the precise query performance. Overall, our contributions include:

- We show that the straightforward extension of the existing method will lead to three new kinds of information leakages in the presence of functional dependencies, namely, the inter-group leakage, intra-group leakage and non-sensitive association transitive leakage.

name	age	gender	zipcode	telephone	position	salary
Jim	21	male	100091	1231223	manager	4000
Alice	22	female	210210	1328228	assistant	4000
Bob	30	male	100087	1242245	CEO	20000
James	22	male	327612	1311245	director	6000
Rob	31	male	321212	1241223	salesman	3000
Kate	29	female	100976	1588228	guarder	2000
David	22	male	200987	1332245	cleaner	1800
Sam	21	male	200543	1341245	salesman	3000
Lily	30	female	233420	1381162	salesman	3000

(a) Sample data of employee

name	age	gender	zipcode	telephone	gid
Jim	21	male	100091	1231223	1
Alice	22	female	210210	1328228	2
Bob	30	male	100087	1242245	2
James	22	male	327612	1311245	1
Rob	31	male	321212	1241223	2
Kate	29	female	100976	1588228	2
David	22	male	200987	1332245	2
Sam	21	male	200543	1341245	2
Lily	30	female	233420	1381162	1

gid	position
1	manager
2	assistant
2	CEO
1	director
2	salesman
2	guarder
2	cleaner
2	salesman
1	salesman

gid	salary
1	4000
2	4000
2	20000
1	6000
2	3000
2	2000
2	1800
2	3000
1	3000

(b) Straightforward Decomposition Result

Fig. 1. Illustration of Example

– We design a new table decomposition method. The method has the following features: First, the decomposed sub-tables meet the requirement of l-diversity for each sensitive association. Second, the decomposed results can avoid the new information leakages incurred by multiple sensitive associations. Third, we design two optimization strategies, including the multiple grouping strategy and the sub-table merging strategy, to improve the query performance over the decomposed sub-tables.
– We propose a query evaluation method over the decomposed sub-tables in the outsourced database and the meta data in the trusted database. The complete candidate results are retrieved and we use the statistical data to determine the result filter approach.

The remainder of the paper is organized as follows: First, Section 2 reviews preliminary knowledge and shows the architecture of sensitive association protection. Then, Section 3 analyzes the new kinds of information leakages and presents a novel table decomposition method. Section 4 discusses the corresponding query evaluation strategy. Section 5 reports the experimental results. Finally, Section 6 reviews the related work and Section 7 concludes the whole paper with discussions for future work.

2 Preliminary Knowledge

In this section, we introduce basic notations of sensitive associations, and outline the architecture of the sensitive association protection in the outsourced database.

2.1 Sensitive Association Rules

The associations to be protected in the outsourced database should not be restricted to the $q - s$ associations due to the general purpose of this model. Thus, the sensitive association in this paper takes the form of functional dependency rule and can be specified by end users.

Definition 1. ***Sensitive Association.*** *Given a table T with functional dependency set F over T, any $f \in F$ takes the form of $\alpha \to \beta$, where α and β are two attribute sets. f is called a sensitive association if the association between α and β needs to be protected. α is the determinant attribute set and β is the dependent set.*

The other dependencies that need not to be protected are called non-sensitive associations. In order to distinguish them, we use $\alpha \xrightarrow{p} \beta$ and $\alpha \rightarrow \beta$ to represent sensitive and non-sensitive associations respectively. Another thing noticed is that sensitive associations in this paper are specified over a single table, and our method can be extended easily to handle sensitive associations across multiple tables.

2.2 Architecture of Sensitive Association Protection in Outsourced Databases

The protection of sensitive associations and support to the precise query evaluation need the combination of the outsourced database and a trusted database, as described in Fig. 2. At the running time, the end users specify the sensitive and non-sensitive associations, and the table decomposition is implemented. When the end user submits a query q, it is first rewritten into a new query q_w against the sub-tables in the outsourced database. The outsourced database executes q_w and retrieves a set of candidate results. The trusted database refines the results returned to pick out the precise results.

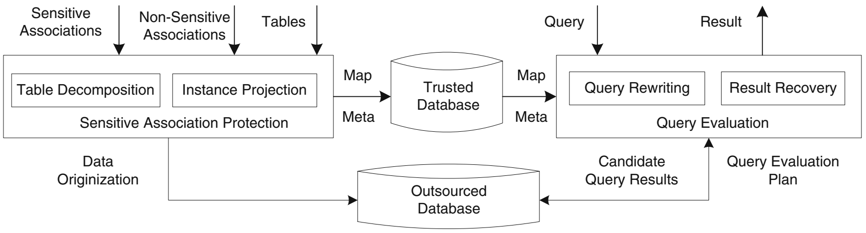


Fig. 2. Architecture of Sensitive Association Protection in Outsourced Database

The architecture in this paper has the following features: First, end users or applications can access the sensitive or non-sensitive data with unified interface provided by the trusted database. Second, the capability of trusted database is not as powerful as that of the outsourced database. It only needs to carry out the query reformulation and the result recovery in a pipeline fashion. Most of the existing databases in current information system meet the requirement.

3 Table Decomposition for Sensitive Association Protection

In this section, we formulate the information leakages incurred by multiple sensitive associations, propose a basic table decomposition method, design two optimization strategies with query performance considerations, and finally show the data placement across the outsourced database and the trusted database.

3.1 Information Leakage Incurred by Multiple Associations

We discuss the new information leakages on the decomposed sub-tables which meet the requirement of l-diversity. That is, for each sensitive association in the form of

$\alpha \xrightarrow{p} \beta$, given a specific value of α in one sub-table, an adversary cannot identify the corresponding value of β in another sub-table through the extra group ID column with a confidence higher than $1/l$. We can notice that the decomposition in Fig. 1(b) satisfies 3-diversity.

Due to the interactions of multiple sensitive associations, we observe three kinds of information leakages on the results of l-diversity decomposition. The first two result from the frequency of data values in the same group or different groups. The third one comes from the transitive result of the functional dependencies.

The first kind of information leakage is called inter-group leakage. Refer back to the motivating example, we can notice that $salesman$ appears in both g_1 and g_2 in the second sub-table in Fig. 1(b). Since there is functional dependency *position* $\rightarrow$ *salary*, the salary of $salesman$ must also occur in both g_1 and g_2. With this inference, the potential salary of $salesman$ can only be 3000 or 4000, with a probability of $1/2$ each, which is greater than $1/3$, resulting in a breach of 3-diversity. This kind of information leakage is referred to as inter-group leakage. The formal definition is given below:

Definition 2. ***Inter-Group Leakage.*** *Consider $\alpha \xrightarrow{p} \beta$ over table T, where α is in decomposed sub-table T_1 and β in T_2. Let g_1 and g_2 be two tuple groups, $V_{\alpha 1}$ be the values of α in g_1 and $V_{\alpha 2}$ be the values in g_2.($V_{\beta 1}$ and $V_{\beta 2}$ take similar meaning). Inter-group leakage occurs if $V_{\alpha 1} \cap V_{\alpha 2} \neq \varnothing$ and $|V_{\beta 1} \cap V_{\beta 2}| < l$.*

The second kind of the information leakage is called intra-group leakage. Consider again the sensitive association *position* $\xrightarrow{p}$ *salary*, we can see $salesman$ appears twice in g_1. As a result, the adversary can be convinced that the salary of $salesman$ is 3000, which is the only value with more than one occurrence. Formally, the intra-group leakage and its condition can be stated as follows.

Definition 3. ***Intra-Group Leakage.*** *Given $\alpha \xrightarrow{p} \beta$ over table T, let $v_\alpha(v_\beta)$ be a value of $\alpha(\beta)$. Suppose the frequency of v_α in group g is $f(v_\alpha, g)$ and the frequency of v_β in g is $f(v_\beta, g)$. Let n be the number of values $v_\beta \in \beta$ satisfying $f(v_\beta, g) \geq f(v_\alpha, g)$. Then, the intra-group leakage occurs when $n < l$, resulting in a breach of l-diversity.*

The third kind of the information leakage is on the scheme level, called non-sensitive association transitive leakage. It results from the transitivity of non-sensitive associations. Let *name*$\rightarrow$*position* and *position*$\rightarrow$*salary* be two non-sensitive associations stored in two separated sub-tables T_1 and T_2 respectively. Suppose that *name*$\xrightarrow{p}$*salary* is a sensitive association, the join of T_1 and T_2 on *position* will disclose the relationship between *name* and *salary*. Formally, it can be described as follows.

Definition 4. ***Non-Sensitive Association Transitive Leakage.*** *Let $\alpha \xrightarrow{p} \gamma$ be a sensitive association, $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$ be two non-sensitive associations. Then, non-sensitive association transitive leakage takes place when there is a sub-table T_1 containing α and β, another sub-table T_2 containing β and γ.*

Notice that all the three kinds of information leakages do not occur in data publishing scenario with single sensitive association[11]. There are only two decomposed sub-tables, QIT and ST. Each value appears only once in QIT, which leads to no intra-group or inter-group leakages. In addition, each attribute belongs to one sub-table only, and there is no scheme level leakage either.

3.2 Basic Table Decomposition Method

Given sensitive and non-sensitive associations, this subsection aims at finding a decomposition satisfying the following properties: 1)it is a l-diversity decomposition; 2)it can counter the new information leakages established in the previous subsection; 3)it supports efficient query evaluation. To fulfill these goals, we first unify all associations into a functional dependency tree.

Definition 5. ***Functional Dependency Tree.*** *Given a table T and the functional dependency set F over T, a functional dependency tree takes the form of $fdt = (V, E, S, r)$, where $S \subseteq F$ is the sensitive association set, each $v \in V$ represents an attribute set in T, each $e \in E$ represents a dependency relationship between two attribute sets, $r \in V$ is the identifier attributes.*

The functional dependency tree can be constructed with the canonical cover of the functional dependency set. In the definition above, we make a restriction that there are no partial functional dependencies to the key. We can notice that this requirement is not too rigorous for most of the tables in real life applications. Figure 3 shows an functional dependency tree with the sensitive associations on the right.

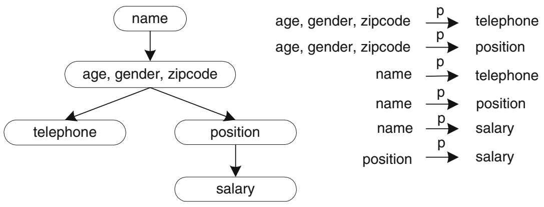


Fig. 3. Example of Functional Dependency Tree

The basic table decomposition method is to build a sub-table for each node in the functional dependency tree, and add extra group IDs into the sub-tables so as to link with each other, and then assign the values of group IDs which guarantees the satisfaction of l-diversity for each sensitive association. Since the first step is trivial, we mainly focus on the latter two tasks, or called "grouping strategy".

One method of grouping strategy is, informally, "one group ID per edge". That is, for each edge in the functional dependency tree, suppose it represents the association $\alpha \to \beta$, we append a group ID column to the sub-table for α and the sub-table for β. Take the root-leaf functional dependency list $L = \alpha_1 \to \alpha_2 \to \alpha_3$ as an example. Let T_i be the sub-table for α_i. We attach both T_1 and T_2 with G_{12} and attach T_2 and T_3 with G_{23}. Under this strategy, suppose a user issues a query which involves α_1 and α_3(*e.g.,* select α_1 from T where α_3=v), we must refer to T_2 to get the correlation of α_1 and α_3 for there is no direct link between them. Thus a lot of extra join cost is incurred.

To remedy this disadvantage, we take another grouping strategy in this paper, or concisely "one group ID per root-leaf list". Consider a more general root-leaf association list $L = \alpha_1 \to \ldots \to \alpha_n$ in fdt, we add group ID G_{1-n} on each sub-table

$T_i(i = \{1, \ldots, n\})$. Therefore, we can link any two sub-tables T_i and T_j $(0 < i, j \leq n)$on this list without referring to other sub-tables. On obtaining the sub-table scheme, we take a bottom up fashion to assign the group ID for each sub-table. First, we make the initial partition by iteratively selecting l different values of α_n into a group until the distinct values remained is less than l. The rest values are added into an existing group. Then, for each value v_{α_n} of α_n in a group g, we enumerate all possible value v in $\alpha_i(1 \leq i \leq n-1)$ where $v \rightarrow v_{\alpha_n}$, and assign them with the same group ID g. This strategy obviously achieves l-diversity decomposition, for each value v in a group of $\alpha_i(1 \leq i \leq n-1)$, there are at least l different values in the same group of $\alpha_j(i < j \leq n)$.

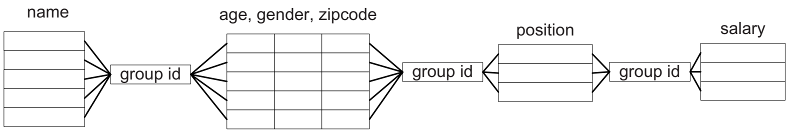


Fig. 4. Grouping on a Root-leaf List in Functional Dependency Tree

Example 2. We make a more specific illustration with the functional dependency list *name* $\rightarrow$ $\{$*age,gender,zipcode*$\}$$\rightarrow$ *position* $\rightarrow$ *salary* as shown in Fig. 4. Suppose the diversity parameter L=3. We select 3 different values from *salary* and compose a group g. Then, we assign the same group to all possible values in *position*, $\{$*age,gender,zipcode*$\}$, and *name*, which can be mapped to the values of *salary* in group g.

3.3 Optimization Strategy 1: Multiple Grouping Annotation

The basic table decomposition method fulfills the first two goals, and partially considers the query performance with the grouping strategy at the granularity of root-leaf list. However, we can notice that from the size of candidate results, it is still too massive in some cases. Refer to Example 2, we will locate 3 distinct *salary* for a given name, whereas the related *position* for a given *name* can be much more. Based on this observation, we define a criteria, termed maximum-diversity, to gauge the grouping.

Definition 6. ***Maximal Diversity.*** *For a functional dependency list* $L = \alpha_1 \rightarrow \ldots \rightarrow \alpha_n$*, we employ* $divs(\alpha_i, \alpha_j)$ *to represent the maximum number of values in* α_j *, which can be linked to a given value in* α_i*. Then, the maximum diversity of* L*, donated as* $md(L)$*, can be defined as* $md(L)= \max\limits_{0<i<j\leq n} divs(\alpha_i, \alpha_j)$.

We take the multiple grouping annotation to overcome the large size of intermediate results indicated by the large maximal diversity. Let $L_1 = \alpha_1 \rightarrow \ldots \rightarrow \alpha_n$ be a functional dependency list, G_{1-n} be the group column added with the basic decomposition. We implement another pass of grouping for $L_2 = \alpha_1 \rightarrow \ldots \rightarrow \alpha_{n-1}$ inside the tuples in the same group under G_{1-n}, and generate another group ID $G_{1-(n-1)}$ for each sub-table in L_2. This annotation continues until $L_{(n-1)} = \alpha_1 \rightarrow \alpha_2$. Then the maximal diversity of L can be reduced approximately to l. For instance, we can optimize the grouping strategy in Example 2 by implementing another pass of grouping for *name* $\rightarrow$ $\{$*age,gender,zipcode*$\}\rightarrow$ *position*, thus a finer group can be achieved inside the original group of *position*, resulting in a reduction of $divs($*name,position*$)$.

3.4 Optimization Strategy 2: Merging of Sub-tables

The query evaluation over the decomposed sub-tables involves join operation in the outsourced database. Besides, the larger size of candidate results comes with join operation. In order to reduce the evaluation cost in the outsourced database as well as to reduce the size of candidate results retrieved, we should minimize the number of the decomposed sub-tables.

It can be seen from the example that some of the sub-tables can be merged into other sub-tables without violating the privacy constraint, while some others can not. Based on this observation, this optimization strategy is designed to merge the compatible sub-tables without privacy breach. We define the following rules to determine whether two sub-tables are compatible.

Definition 7. ***Compatible Sub-Tables.*** *Given a table T with functional dependency set F, and the sensitive association set $S \subseteq F$, let $T.A$ be the attribute set of table T, two sub-tables T_0 and T_1 are compatible with each other, when the following conditions are satisfied:*

(1) $T_0.A \cup T_1.A$ does not contain a sensitive association in S;
(2) $T_0.A \cup T_1.A$ does not contain a non-sensitive association r, with which the functional dependency list $\alpha_0 \rightarrow \ldots r \ldots \rightarrow \alpha_n$ can be built, where $\alpha_0 \xrightarrow{p} \alpha_n$ is a sensitive association, and the attributes of α_i and $\alpha_{i+1} (0 \leq i < n-1)$ are in the same sub-table;
(3) $T_0.A \cup T_1.A$ does not contain a non-sensitive association $\alpha \rightarrow \beta$, while there is another sub-table $T^{'}$ containing $\gamma \rightarrow \delta$, and $\alpha \xrightarrow{p} \gamma$ and $\beta \xrightarrow{p} \delta$ are sensitive.

The first condition ensures that there is no direct information leakage for a sensitive association. The second one thwarts the non-sensitive association transitive leakage. The third requirement can avoid the intra-group leakage, since adding β with α may result in multiple same values of β in one group. Taking the example in Fig. 3, the sub-table for $\{$*name*$\}$ and $\{$*age, gender, zipcode*$\}$ can be merged without violating the above requirements.

3.5 Data Organization across Outsourced Database and Trusted Database

The original table T is distributed across the outsourced database and the trusted database after decomposition. The outsourced database holds the decomposed sub-tables, while the trusted database keeps the meta data of the outsourced sub-tables as well as the necessary information for correct results recovery.

Suppose the base table T is decomposed into a set of sub-tables $(T_0, \ldots, T_n)$. For each sub-table T_i in the outsourced database, the attributes include $\{E, T_i.vid, G_0, \ldots, G_k\}$, where E is the original attribute set in table T, $T_i.vid$ is the extra unique ID for each tuple in T_i, and $G_0, \ldots, G_k$ are the group ID columns generated with the grouping strategy above.

The trusted database maintains two kinds of information: one is meta data and the other is map table. Specifically, the meta data describes the schema of the outsourced sub-tables, which can be used to generate the query evaluation plan against the outsourced database when accepting a query. And, the map table records the tuple relationship between sub-tables in the form of $M = (T_0.vid, \ldots, T_n.vid)$, where $(T_0, \ldots, T_n)$ are all decomposed sub-tables.

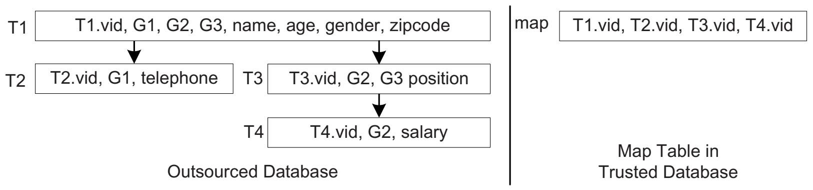


Fig. 5. Example of Data Organization

Figure 5 shows the data distribution across the outsourced database and trusted database. The sub-tables for $\{name\}$ and $\{age, gender, zipcode\}$ are merged. Multiple group IDs are assigned on sub-tables T_1, T_3 and T_4. We can see that G_2 is their first common group ID and G_3 is a finer group implemented on G_2, which can only link T_1 and T_3.

4 Query Evaluation over Protected Sensitive Association

We support the simple selection-projection-join query, and simple aggregation query. The query predicate takes the form of α *op* b, where α is an attribute, b is an explicit value, and *op* is one of $\{<, \leq, =, >, \geq\}$. The target list can include attributes or aggregation functions, such as max, min. Besides, our method naturally supports the join operation among tables.

As discussed in Section 2, the query evaluation over decomposed sub-tables takes place as follows: when accepting a query q, the trusted database exploits the meta data to rewrite it into q_w, which is appropriate to execute over the decomposed sub-tables. The outsourced database executes q_w and returns a set of candidate results. Then, the trusted database figures out the correct results from the candidate ones with the map table maintained. Since the outsourced database offers more significant computational resource, we should alleviate the burden of trusted database by pushing as much work as possible to the outsourced database. In what follows, we discuss the query plan rewriting and the precise results recovery.

4.1 Query Plan Rewriting

The query plan rewriting is based on the meta data produced during the table decomposition. The meta data describes the scheme information of the sub-tables in the outsourced database, from which we can easily get all the tables containing a given attribute or the common group ID columns for two specified sub-tables. The goal of this step is to transform the original query into a new one which can be correctly executed in the outsourced database and at the same time, minimize the size of the candidate results to be returned. We use the following query as an example to illustrate the processing:

Example 3. For the decomposed sub-tables illustrated in Fig. 5, the end user submits the following query:

select *name*, *telephone*, *salary* from *employee* where *position*= "$manager$".

We first determine the minimal sub-table set which covers all the attributes involved in the query q. The related attributes, donated $q.A$, include the attributes in selection clause and the ones in where clause. For each attribute $a \in q.A$, we can locate a list of sub-tables containing a with the meta data maintained. So, there might be several sub-table sets containing all the attributes in $q.A$. As we can see, all the sub-tables should be joined together on the group ID column to find the candidate results. In order to reduce the evaluation cost, we calculate the $linkcost$, which is the number of join operations needed to link all the sub-tables, for each candidate set, and choose the one with minimal $linkcost$ to be the final sub-table cover. As for the example, $q.A$=$\{$*name, salary, telephone, position* $\}$, thus all the sub-tables T_1, T_2, T_3 and T_4 are selected into the sub-table cover.

We then identify the group IDs used to link sub-tables. Due to the multiple grouping strategy, there might be more than one group ID which can be used to join two sub-tables. As we take a bottom-up fashion in the annotation step, the group ID latest generated will be selected, for it minimizes the maximal diversity between the two sub-tables, resulting in a smaller candidate result set. Revisiting Example 3, although both G_2 and G_3 can be used as join condition for T_1 and T_3, we will select G_3. Besides, we select G_1 to link T_1 and T_2, select G_2 to link T_1 and T_4.

Now, we can rewrite the query using the information obtained. The new from clause is made up by all the sub-tables in the sub-table cover. The new selection clause contains all the target attributes of the original query, as well as the vid attributes of the related sub-tables in the from clause, which are necessary for the trusted database to figure out the correct results. Finally, we rewrite the where clause. The original predication in q is preserved, and new conditions for the join of sub-tables are added. As for the example, the final transformed query takes the form of:

select *name*, *telephone*, *salary*, T_1.*vid*, T_2.*vid*, T_3.*vid*, T_4.*vid*
from T_1, T_2, T_3, T_4 where *position*="$manager$" and $T_1.G_1 = T_2.G_1$ and $T_1.G_3$
$= T_3.G_3$ and $T_1.G_2 = T_4.G_2$.

What's more, we optionally add an order-by clause to speed up the result recovery. The addition depends on the size of the candidate results, which can be predicted by the statistical data stored in the trusted database. When the predicted size is massive, sorting is recommended and when it is small, the clause is needless. We will detail this in the following subsection.

4.2 Query Results Recovery

The candidate result set (donated as RS_c) returned from the outsourced database is a superset of the correct results, and needs to be refined with the map table stored in the trusted database.

There exists multiple approaches to recover the precise results from the candidate ones. When the statistical information indicates a small-size candidate result set, the query has no order-by clause. The trusted database directly selects the tuples in the map table with *vid* value on predicted attribute, donated as RS_{map}. Since the result on predicted attribute is accurate, RS_{map} is the correct join relation of the precise results. Then, it checks each candidate tuple with the *vid*s attached, figures out the tuples with *vid*s contained in RS_{map}, and discards the ineligible ones.

When a large-size candidate result set is indicated by statistical data, the better way is to implement the merge join operation between the candidate results and the map table. During the query rewriting, an order-by clause is appended to q_w. The outsourced database executes q_w, generating candidate results with the specified order list. And meantime, the trusted database sorts the map table with the same *vid* list. Upon receiving the two sorted result sets, the trusted database performs a merge join to filter out the correct results in a pipeline way.

We can see the query evaluation in our framework is correct. First, the table decomposition in Section 3 and the rewriting processing guarantee the completeness of the candidate result, namely, all correct results will be retrieved. Second, the recovery method ensures that all and only the results matching the *vid*s in map table will be filtered out. Besides, it is clear that the one-way information flow during the whole evaluation will not lead to information leakages.

5 Experiments

This section experimentally evaluates the efficiency and effectiveness of our table decomposition technique. The experiments are carried out on two Dell computers running Windows XP Professional operating system. The one with a 1.6GHz CPU and 1.5G RAM performed as the trusted server, and the other with a 1.8GHz CPU and 16G RAM acted as the outsourced service provider.

5.1 Experimental Setup

DataSet. Since there is no public dataset with specified functional dependencies, all experiments are conducted on two synthetic employee datasets, with the same attributes but different functional dependencies. There are 100k tuples in each dataset. Table 1 summarizes the 9 interval attributes, and Table 2 details the two datasets, donated as EMP_1 and EMP_2 respectively, with their association set, as well as the corresponding sub-table schemes.

Table 1. Summary of Attributes

attribute	# of distinct values	attribute	# of distinct values	attribute	# of distinct values
eid	100k	*age*	40	*telephone*	100k
birthday	700	*gender*	2	*zipcode*	100
companyid	800	*position*	120	*salary*	60

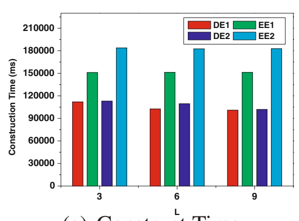
Competitor. We compare our method with encrypted database. For generalization, we encrypt table in the granularity of column using AES algorithm, and employ the bucket-based index in [7] to speed up the query evaluation. To be more comparative, we keep the size of each bucket the same as the diversity parameter L used in our table decomposition method. To offer the true information-theoretic privacy, the encoding method must satisfy: 1)not equality preserving, 2)not order preserving, 3)encoded as fixed-length. Taking all these factors into consideration, we encode a value v to get its cipher data v_e in the following way: $v_e = E_k(v||r)$, where r is a random value and E_k is the AES encryption function with key k. What's more, we use a greedy algorithm to minimize the number of encrypted attributes on condition that the two parts of each rule will not appear in the same table. For EMP_1, the greedy algorithm chooses to encrypt *eid* and *salary*. And for EMP_2, it selects *eid*, *companyid* and *position*.

Table 2. Association Set and Decomposed Sub-table Scheme

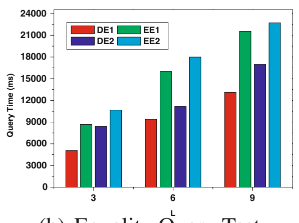
Set	Association set	Decomposed sub tables
EMP_1	*eid*$\xrightarrow{p}$*salary*	T_0(*salary*, G_1)
	eid$\xrightarrow{p}$*telephone*	T_1(*companyid*, *position*, G_2, G_3)
	eid$\rightarrow$(*companyid,position*)	T_2(*telephone*, G_3)
	(*companyid,position*)$\xrightarrow{p}$*salary*	T_3(*eid*, *birthday*, *gender*, *age*, *zipcode*, G_1, G_2, G_3)
EMP_2	*eid*$\xrightarrow{p}$*companyid*	T_0(*salary*, G_1)
	eid$\xrightarrow{p}$*salary,position*	T_1(*position*, G_1, G_2)
	companyid$\xrightarrow{p}$*position,salary*	T_2(*companyid*, G_1, G_2, G_3)
	position$\xrightarrow{p}$*salary*	T_3(*eid,birthday,gender,age,zipcode,telephone*,G_1,G_2,G_3)

5.2 Experimental Results

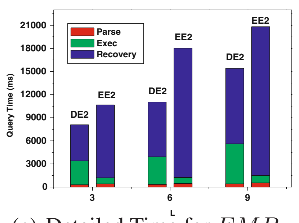
Data Placement Cost. In the first set of experiments, we study the efficiency of our table decomposition algorithm, by comparing the time used to place the data into the outsourced database. The result is illustrated in Fig. 6(a). Notice that EEi means the results of **E**ncrypted **E**MP_i, and DEi presents the results from the **D**ecomposed **E**MP_i, where $i \in \{1, 2\}$. From Fig. 6(a), we can see that the table construction time for decomposed database is less than that of encryption. The advantage is more obvious on EMP_2, since one more attribute should be encrypted compared with EMP_1. Also, we can see that the time costs for both methods decrease slightly as L increases, since less group time is needed.



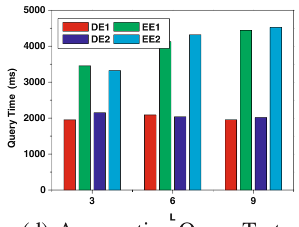
(a) Construct Time



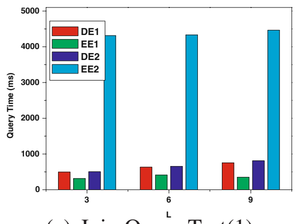
(b) Equality Query Test



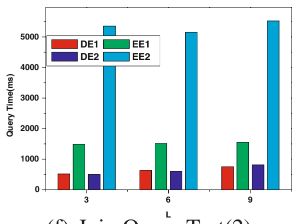
(c) Detailed Time for EMP_2



(d) Aggregation Query Test



(e) Join Query Test(1)



(f) Join Query Test(2)

Fig. 6. Experimental Results

Query Evaluation Time. Next, we conduct three kinds of query tests, basic equality query, aggregation query, as well as join query, to analyze the efficiency of the two different methods. In each test, we randomly generate 100 queries of specific type and then evaluate on each dataset. All the processing time is measured in ms.

Figure 6(b) shows the results of the basic equality query test. We can see that the table decomposition method outperforms encryption method on both datasets as expected. In fact, the parse time of both methods is nearly the same. The main difference comes from the execution step and the result recovery(or filtering) step.

In order to understand the underlying reason behind such a result in total cost, we illustrated the detailed composing of the time costs for both methods on EMP_2 with various L in Fig. 6(c). From the perspective of execution time, encryption performs better than decomposition since we need to do some join operations on the outsourced database side. However, when it comes to the result filtering step, decomposition method is preponderant. We can see that result recovery is the dominating factor of the total costs. As decryption takes most of the evaluation time for the encrypted database, the query processing responsibility is mainly at the trusted database side, which is not preferred since it mitigates most advantages of data outsourcing. Besides, we observe the evaluation cost increases as L increases. This is due to the fact that the candidate results become larger when the size of each group or bucket is enlarged. Thus, more work should be done to seek out the exact results.

To further test our technique, we generated 100 simple aggregation queries, including max, min etc., and evaluated on both datasets. Figure 6(d) shows the results. We can see that the time cost of the decomposed database is nearly half that of the encrypted database. Because in the decomposed database, we can get the max or min value of each attribute directly while in an encrypted database which is not order-preserving, if we want to get the max or min value of an encrypted column, we should first find out the bucketid of which the max(or min) element belongs to, and then, fetch all the data in that bucket, do the decryption to get the correct result. A large L yields a large set of candidate results. That's why the evaluation time for DE_i keeps steady while that of EE_i increases with the increase of parameter L.

We finally analyze the cost for answering join queries. We generated another table *Company* with scheme *(companyid, name, foundyear, workforce)*. There are 800 distinct tuples in *Company* and it can join with *Employee* on *companyid*. For the decomposed database, it is easy to do the join, just as the normal database. On the contrary, it is a big challenge for the encrypted database if the join condition is happened to be encrypted. In the first case, if the *companyid* of *Company* is in plaintext, it is easy for EMP_1 to join with *Company*. However, the easiest way for the encrypted EMP_2 to join with *Company* is to fetch all the tuples of *Company* and *Employee* back to the trusted database, and decrypt *Employee* before join, which is obviously too expensive to be executed. In another case, if the attribute *companyid* in *Company* is encrypted, it is difficult for both *Employee* sets to do the join. For EMP_1, all the *companyid*s in *Company* must be decrypted, while for EMP_2, both *companyid*s in *Company* and in EMP_2 should be decrypted, due to the non-equality preserving property of the encoding method. Figure 6(e) shows the query costs when *companyid* in *Company* is in plaintext and Fig. 6(f) shows the costs when it is encrypted. We can observe that the decomposition offers much better performance.

To sum up, the decomposed database outperforms the encrypted database in both aspects. First, it takes less time in data placement. Second, it supports more efficient query evaluation in all query tests, especially when the join condition is located on an encrypted attribute.

6 Related Work

Existing solutions to the privacy preserving problem in outsourced database is mainly based on encryption. Many researches have been done on query processing over the encrypted database[7,8], and various index schemes have been proposed in literature to speed up query evaluation[4,10,5], or balance the confidentiality and efficiency[6]. The main disadvantage of this method is that the decryption step burdens the local processor, leading to a bottleneck on the client. An alternative approach is proposed in [3], which suggests partitioning the relation over two non-communicating servers. However, the encryption is still unavoidable. In this paper, different from previous works, we aim at solving the problem without resorting to encryption.

The table decomposition method is also proposed in [11] to achieve l-diversity while to improve the utility of the published data. However, it is not applicable in our context, since it only considers single sensitive association and does not take into account the functional dependencies among attributes. Moreover, it does not support the precise query. In this paper, we address the privacy problem in the presence of multiple sensitive associations, and provide efficient evaluation for precise query.

On another side of related work, the distributed query processing focuses on the vertical table decomposition or the horizontal table decomposition, or the hybrid decomposition. The semi-join algorithm is proposed to improve the query performance in the distributed database [9]. Our paper also discusses the distributed query processing across the trusted database and the outsourced database. However, the current distributed query processing methods cannot be used directly in our environment since the outsourced database is not fully trusted. The candidate result set is generated first in the outsourced database, and the interaction between two sides are forbidden in our context.

7 Conclusion and Future Work

In this paper, we learn from the privacy conscious data publishing, and propose a novel privacy preserving method based on table decomposition to protect the multiple sensitive associations in the outsourced database, in the presence of functional dependency. The empirical study indicates that our method is superior to the encryption method.

The future work contains a list of interesting directions: First, this paper discusses the sensitive associations expressed with the functional dependency. We can extend it to support more complex associations such as probabilistic association. Second, our method can also be adapted to data publishing scenario, whereas it remains open how to implement the data analysis over decomposed sub-tables, without referring to the trusted database.

References

1. Microsoft sql server data service, http://www.microsoft.com/azure/data.mspx
2. Simple database in amazon, http://aws.amazon.com/simpledb/
3. Aggarwal, G., Bawa, M., Ganesan, P., et al.: Two can keep a secret: A distributed architecture for secure database services. In: Proc. of CIDR, pp. 186–199 (2005)
4. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proc. of SIGMOD, pp. 563–574 (2004)
5. Chung, S.S., Ozsoyoglu, G.: Processing Aggregation Queries over Encrypted Databases. In: Proc. of ICDE (2006)
6. Damiani, E., Vimercati, S.D.C., Jajodia, S., et al.: Balancing confidentiality and efficiency in untrusted relational DBMSs. In: Proc. of CCS, pp. 93–102 (2003)
7. Hacigümüş, H., Iyer, B., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In: Proc. of SIGMOD, pp. 216–227 (2002)
8. Hore, B., Mehrotra, S., Tsudik, G.: A privacy-preserving index for range queries. In: Proc. of VLDB, pp. 720–731 (2004)
9. Kambayashi, Y., Yoshikawa, M., Yajima, S.: Query processing for distributed databases using generalized semi-joins. In: Proc. of SIGMOD, pp. 151–160 (1982)
10. Li, J., Omiecinski, E.R.: Efficiency and security trade-off in supporting range queries on encrypted databases. In: DBSec, pp. 69–83 (2005)
11. Xiao, X., Tao, Y.: Anatomy: Simple and effective privacy preservation. In: Proc. of VLDB, pp. 139–150 (2006)

A Semantic Information Loss Metric for Privacy Preserving Publication*

Yu Liu, Ting Wang, and Jianhua Feng

Department of Computer Science and Technology
Tsinghua National Laboratory for Information Science and Technology
Tsinghua University, Beijing 100084, China
{liuyu-05,wangting09}@mails.tsinghua.edu.cn,
fengjh@tsinhgua.edu.cn

Abstract. Data distortion is inevitable in privacy-preserving data publication and a lot of quality metrics have been proposed to measure the quality of anonymous data, where information loss metrics are popularly used. Most of existing information loss metrics, however, are non-semantic and hence are limited in reflecting the data distortion. Thus, the utility of anonymous data based on these metrics is constrained. In this paper, we propose a novel semantic information loss metric **SILM**, which takes into account the correlation among attributes. This new metric can capture the distortion more precisely than the state of art information loss metrics especially for the scenario where strong correlations exist among attributes. We evaluated the effect of **SILM** on data quality in terms of the accuracy of aggregate query answering and classification. Comprehensive experiments demonstrate that **SILM** can help improve the quality of anonymous data much more especially if integrated with proper anonymization algorithms.

Keywords: k-anonymity, information loss metric, data distortion, data utility.

1 Introduction

In recent years, large amount of personal data has been collected, stored, processed and published for both scientific and business purposes. Such personal data, however, usually contains sensitive information of individuals, so sharing and publishing these data pose a threat to individual privacy. To solve this problem, a large number of privacy-preserving publishing models based on anonymity techniques have been proposed, such as *k*-anonymity[1] and so on. In anonymization, generalization and suppression are two widely employed anonymity techniques. Meanwhile, information loss is inevitable in anonymization. Vague meanings caused by information loss usually decrease the quality of the data and affect data utility. Therefore, the contradiction between the privacy and the data quality always exists.

* This work is partly supported by the National Natural Science Foundation of China under Grant No. 60873065, the National High Technology Development 863 Program of China under Grant No.2007AA01Z152 and 2009AA011906.

Given privacy constraints, most studies try to improve data quality through adopting a more legitimate quality metric or by devising optimal algorithms and recoding methods if a quality metric has been designated. Methods for measuring data quality can be categorized into two categories: task-independent and task-dependent.

Task-independent metrics evaluate data quality by measuring information loss, which consists of generic metrics and distance-based metrics. These metrics are usually applied when data publishers do not know the ultimate use of the published data. Generic metrics measure information loss by the size of QI group or the total number of generalization/suppression [1,2,3]. While distance-based metrics reflect data distortion by the distance among distinct values [4].

Task-dependent metrics declare that the best way of measuring quality is based on the ultimate application of the data. These metrics are proposed for the scenario where anonymous data is released for particular analysis, such as data mining, aggregate query and so on. Task-dependent metrics usually incorporate a target workload into the anonymization process[5,6,7]).

Table 1. An Original Table T_1

	Age	Gender	Zip	Hobby
t_1	10	M	21008	H_1
t_2	10	M	21003	H_1
t_3	20	F	22002	H_2
t_4	30	M	20003	H_2
t_5	40	F	20016	H_2
t_6	60	F	21002	H_3
t_7	70	F	21029	H_3
t_8	50	F	21023	H_1
t_9	5	F	20013	H_3

Table 2. An Original Table T_2

	Age	Gender	Zip	Hobby
t_1	10	F	21008	H_1
t_2	10	M	20003	H_1
t_3	20	F	22002	H_2
t_4	30	M	20003	H_2
t_5	40	F	20016	H_2
t_6	60	F	21002	H_3
t_7	70	F	21029	H_3

Table 3. 3-anonymous table Y_1

		Age	Gender	Zip	Hobby
	t_1	[10, 50]	*	21000	H_1
G1	t_2	[10, 50]	*	21000	H_1
	t_8	[10, 50]	*	21000	H_1
	t_3	[20, 40]	*	20000	H_2
G2	t_4	[20, 40]	*	20000	H_2
	t_5	[20, 40]	*	20000	H_2
	t_6	[5, 70]	F	20000	H_3
G3	t_7	[5, 70]	F	20000	H_3
	t_9	[5, 70]	F	20000	H_3

Table 4. Another 3-anonymous table Y_1'

		Age	Gender	Zip	Hobby
	t_9	[5, 10]	*	20000	H_3
G1	t_1	[5, 10]	*	20000	H_1
	t_2	[5, 10]	*	20000	H_1
	t_3	[20, 40]	*	20000	H_2
G2	t_4	[20, 40]	*	20000	H_2
	t_5	[20, 40]	*	20000	H_2
	t_6	[50, 70]	F	21000	H_3
G3	t_7	[50, 70]	F	21000	H_3
	t_8	[50, 70]	F	21000	H_1

Obviously, an anonymous table generated based on a specific metric may not perform well when being evaluated by another metrics. A typical case is shown in Table 1. The task here is to do 3-anonymization in T_1 based on some classification-dependent metrics. And the goal of classification is to predict the type of *Hobby*. Table 3 is a 3-anonymous table generated from T_1, and it shows that age 5 and 70, age 10 and 50 are generalized together since they share the same hobby. Obviously, over generalization occurred and the distortion is great when measured by distance-based metrics.

Table 5. 3-anonymous table Y_2

		Age	Gender	Zip	Hobby
G1	t_1	[10,20]	*	20000	H_1
	t_2	[10,20]	*	20000	H_1
	t_3	[10,20]	*	20000	H_2
G2	t_3	[30,40]	*	20000	H_2
	t_5	[30,40]	*	20000	H_2
G3	t_6	[60,70]	F	21000	H_3
	t_7	[60,70]	F	21000	H_3

Table 6. Another 3-anonymous table Y_2'

		Age	Gender	Zip	Hobby
G1	t_1	[10,10]	*	20000	H_1
	t_2	[10,10]	*	20000	H_1
G2	t_3	[20,40]	*	20000	H_2
	t_4	[20,40]	*	20000	H_2
	t_5	[20,40]	*	20000	H_2
G3	t_6	[60,70]	F	21000	H_3
	t_7	[60,70]	F	21000	H_3

In comparison with task-dependent metrics, task-independent metrics especially those distance-based metrics can intuitively reflect the data distortion caused by generalization. Most of them, however,neither consider the application scenarios nor the correlations between different attributes in distance measuring. In other words, this kind of distance metrics is non-semantic and only reflects the data distortion symbolically. Take Table 2, Table 5 and Table 6 for an example, both Y_2 and Y_2' are 2-anonymous tables of Y_2. If measured by a conventional distance-based metric, the information loss of $[10, 20]$ is equal to that of $[60, 70]$, which is 1/10. But it is obvious that people in age 10 and age 20 are much more different in hobby than those of in age 60 and 70. For classification purpose, it seems better to merge age 20 and 30 instead of merging age 10 and 20, just as shown in Y_2'.

To overcome these shortcomings of existing metrics, in this paper we define a new semantic information loss metric-**SILM**. In a sum, the main contributions of this paper are as follows:

- We propose a novel semantic information loss metric **SILM** for numeric attributes and categorical attributes respectively, which can measure the information loss of an anonymous data table more precisely.
- To evaluate the impact of SILM on data quality, we improved and implemented two anonymizaiton algorithms by incorporating SILM.
- Extensive experiments are conducted under various settings, and results show that SILM outperformed state-of-the-art task-independent and task-dependent metrics in terms of both aggregate query answerability and classification accuracy no matter what anonymity algorithms are used or what classifiers are built.

The rest of this paper is organized as follows. In Section 2, the related work is introduced in brief. Basic preliminaries and conceptions are presented in Section 3. We review previous quality metrics and propose SILM in Section 4. Two improved anonymization algorithms based on SILM is devised in Section 5. At last we discuss experimental results in Section 6 and conclude in Section 7.

2 Related Work

So far, literatures on privacy-preserving data publishing can be approximately classified into two categories. The first category aims at devising various anonymity principles w.r.t. different kinds of attacks, such as k-anonymity[1]. Recently, plenty of new principles are proposed for some special applications, such as *personalized anonymity* proposed in [8] which can provide multi-level security for each tuple, and m-invariance in [9] which effectively decreases the risk of privacy disclosure in re-publication with insertions and deletions. Other works also suggest partition the data into "sensitive" and QI tables, and then permute sensitive values[10].

The second category focuses on anonymization algorithms and recoding methods for generalization. Various recoding approaches have been studied, such as global recoding generalization [1,2,6], multidimensional recoding generalization[3], local recoding generalization[4,11]. Global recoding and local recoding work on one attribute while multidimensional recoding works on a set of QI attribute. Accompanied with recoding methods, many optimization algorithms have been developed[6,12] to minimize given quality metrics.

To measure data qualities, most of existing studies intended to employ some general-purpose quality metrics without considering data utilities[2,4,13,14]. These metrics usually evaluated data quality by measuring the information loss. In response to these general-purpose quality metrics, much more researchers argued that the quality is best judged w.r.t. the workload for which the anonymous data will be used [7,6,5]. However, to the best of our knowledge, when defining quality metrics, all previous studies concern neither the value distribution of each QI attribute nor correlations among different attributes.

3 Preliminaries

Next, we introduce some preliminaries and concepts to facilitate our following discussion.

In this paper, we only focus on privacy-preserving data publishing where generalization-based anonymization is employed, specifically, we conduct our studies on k-anonymity publishing. Similar to previous works, we made an assumption that data for publishing is usually stored and released in the form of relational tables. In order to define anonymity on a relational table T, attributes in T usually can be classified into four types: (1) **Identification attribute (IA):** *the identification attribute* is an attribute that can identify an individual uniquely. IA is usually removed entirely from the published table. (2) **Quasi-identifier(QI) attributes:** *quasi-identifier attributes* are a set of attributes which can potentially identify an individual when being linked to external data sources. (3) **Sensitive attribute(SA):***sensitive attribute* is an attribute whose values are not allowed to be uniquely associated with an unique individual. (4) **Ordinary attributes (OA):***ordinary attributes* refer to attributes which do not belong to any of the above three types.

In an anonymous table Y, a group of tuples with identical quasi-identifier values composes a ***quasi-identifier(QI) group***. For example, in Table 3 where the QI attributes consist of $\{Age, Gender, Zip\}$, G_1, G_2, G_3 are three QI groups.

Definition 1. ***Information Loss***
Given two values v and v' where v is the original value and v' is a generalized value of v, the deviation of v' from v is the information loss, denoted by $loss(v, v')$.

The concept of *information loss* or *data distortion* often be used to reflect the data quality in privacy-preserving publishing. The data quality is also evaluated by classification accuracy in this paper, some preliminaries related to classification will be introduced.

The attribute in a data table with nominal target class labels is called ***target attribute***, the set of discrete or continuous attributes used to predict the target attribute are called ***predictor attributes***. In the remaining of this paper, the QI attributes are assumed as predictor attributes.

4 Measuring Data Quality

In this section, we will discuss several typical data quality metrics in detail. Without loss of generality, let $T(Q_1, Q_2, ..., Q_d, S, C)$ be an original table, $Q_i(1 < i < d)$ is a quasi-identifier attribute and used as predictor attribute, S is a sensitive attribute and C is the target attribute. $|T|$ denotes the size of table T and $t \in T$. $t_{i,j}$ denotes the value of the i_{th} tuple in the attribute Q_j. Y is a k-anonymous table of T consisting of m QI groups $\{G_1, G_2, ..., G_m\}$. $G(y)$ refers to the QI group y belongs to where $y \in Y$.

4.1 Traditional Data Quality Measurement

The most popular task-independent metrics are *DM(Discernability Metric)* and *CAVG*[2], where the information loss is determined by the size of QI groups. Many work, however, have pointed out that such generic metrics are mathematically sound but not intuitive to reflect changes caused by anonymization[13]. Therefore, they are not necessarily indicative of data distortion. To overcome this, Iyengar[14] defined a general loss metric(LM) based on domain generalization hierarchies which are also called taxonomy trees. In this metric, domain generalization hierarchies are predefined and used to build mappings between specific and general values. LM is defined as follows.

Definition 2. *(LM)*
Let $\mathcal{LM}(y_{i,j})$ denotes the information loss of a generalized item $y_{i,j}$ in Y, $\mathcal{LM}(Y)$ denotes the total information loss of Y, then

$$\mathcal{LM}(y_{i,j}) = \begin{cases} \dfrac{b-a}{max_j - min_j}, & Q_j \text{ is a numeric attributes} \\ \dfrac{|leaf(y_{i,j})| - 1}{|Q_j| - 1}, & Q_j \text{ is a categorical attributes} \end{cases} \tag{1}$$

$$\mathcal{LM}(Y) = \sum_{i=1}^{|Y|} \sum_{j=1}^{d} \frac{\mathcal{LM}(y_{i,j})}{|Y| * d} \tag{2}$$

Here, $t_{i,j}$ is generalized into $y_{i,j}$ which is denoted by an interval [a, b]. $[min_j, max_j]$ stands for the domain range of numerical attribute Q_j. LM is a task-independent metric, and it can capture the distortion caused by generalization more precisely than DM and CAVG.

For task-dependent metrics, Iyengar[14] proposed an interesting cost metric *CM (classification metric)* to produce an optimal k-anonymous table for building a classifier.

Definition 3. ***CM***
Let y be a tuple of Y, the function $cls(y)$ *returns the class label of y,* $maj(G)$ *returns the majority class label of a QI group G, then*

$$\mathcal{CM} = \sum_{i=1}^{|Y|} pen(y_i)/|Y| \tag{3}$$

here $pen(y_i) = 1$ *if* $cls(y_i) \neq maj(G(y_i))$*; else* $pen(y_i) = 0$.

Iyengar has demonstrated that anonymous tables based on this metric can yield better classification accuracy than those based on generic metrics. Furthermore, to improve the data utility for particular data mining, some works have tried to incorporate data mining models into anonymity algorithms [5,6].

4.2 Shortcomings of Traditional Data Quality Metric

We have following observations about previous data quality metrics.

Observation 1. *Task-dependent metrics cannot reflect the data distortion for they ignore QI attributes during anonymization.*

Task-dependent metrics usually incorporate expressive workload characteristics into the procedure of anonymization. For example, the algorithm of TDS and Median Mondrian[5] partition QI groups based on information gain, which is reminiscent of decision tree construction. Apparently, these task-aware metrics do not take into account QI attributes at all, but focus on the impurity of target attribute in a QI group. According to the definition of information loss, such metrics can not reflect the information loss at all.

Considering Table 3 and Table 4, Y_1 is based on a classification conscious metric while Y_1' is based on a task-independent metric. Because information loss of *Gender* and *Zip* are equal in two tables, only information loss for *Age* are computed as follows: $\mathcal{CM}(Y_1)$= 0; $\mathcal{CM}(Y_1')$= $\frac{2}{9}$; $\mathcal{LM}(Y_1)$= 3.75; $\mathcal{LM}(Y_1')$= 1.35. Here, when evaluated by CM, Y_1 performs much better than Y_1'. But when measured by LM, the information loss of Y_1 is larger than that of Y_1'. It can be inferred that although Y_1 could yield better classification accuracy than Y_1', it would not perform so well when used for other applications such as aggregate query answering due to the great distortion. So, the inherent limitation of task-dependent metrics restricts the anonymous data to be widely exploited.

Observation 2. *Task-independent metrics are limited in reflecting the data distortion caused by anonymization for they are non-semantic.*

It is understandable that the ultimate usage of anonymous data is uncertain in most cases. To enhance the data utility, the intuitive way is to preserve as more original information as possible. Although existing distance-based task-independent metrics satisfy such requirements to some extent, they are lack of semantics when measuring the information loss. Take the definition of LM for an example. For numerical attribute N, when two values v_1 and v_2 are generated into an interval $[v_1, v_2]$, the distortion is measured by Equation 1. The formula implies three layers of meanings. First, the domain values of N conform to uniform distribution. Second, each value of N is only taken as non-semantic number. Third, attribute N is irrelevant to other attributes. But this is not the case in real world. If such correlations and their impacts on the value distribution of attributes are not considered, more errors would occur in information loss measurement. Such errors will directly decrease the utility of the anonymous data.

For categorical attributes, We have pointed that LM is not an advisable way to measure information loss[15].

In summary, these observations inspire us to search for better metric such as semantic information loss metric SILM in this paper to help produce anonymous data with less distortion and more utility.

4.3 SILM - The Semantic Information Loss Metric

SILM for Numerical Attributes. For simplicity, we only studied the case where only one numerical QI attribute Q_i has correlation with the target attribute C and the distribution of Q_i is affected by C. We also assume domain values of all numerical attributes are discrete. suppose $m \in domain(Q_i)$, $n \in domain(C)$, $f_i(m, n)$ denotes the number of tuples satisfying $t.Q_i = m$ and $t.C = n$, $cnt_i(m)$ denotes the number of tuples satisfying $t.Q_i = m$.

Definition 4. *$g_i(m, n)$ is the distribution probability of n with respect to m, written as*

$$g_i(m, n) = \frac{f_i(m, n)}{cnt_i(m)}$$

In fact, $g_i(m, n)$ is the probability of $t.C = n$ under the condition that $t.Q_i = m$.

Definition 5. *$h_i(m, n)$ is the distribution probability of n with respect to the neighborhood of m, written as*

$$h_i(m, n) = \frac{\sum_{j=m}^{m+\Delta} f_i(j, n)}{\sum_{j=m}^{m+\Delta} cnt_i(j)}$$

Δ refers to the range of neighborhood of m. In contrast to $g_i(m, n)$ which represents the distribution around a single point, $h_i(m, n)$ is introduced to reflect the distribution of n w.r.t. a small range around m. When $\Delta = 0$, $h_i(m, n)$ equals $g_i(m, n)$. The value of Δ is determined by $|Q_i|$. If $|Q_i|$ is large, $\Delta = 0$ may result in a singular point and affect the accuracy of $h_i(m, n)$. So, Δ should be assigned with a proper value larger than zero.

Now we define $R_i(m, n)$ to capture the change ratio of distribution of $n \in domain(C)$ over the neighborhood of a specific m.

Definition 6. *$R_i(m)$ is the change ratio of distribution of the attribute C w.r.t. the neighborhood of m, written as:*

$$R_i(m) = max\{|\ln \frac{h_i(m+\Delta, n_j)}{h_i(m, n_j)}| + \varepsilon\} | h_i(m, n_j) \neq 0, n_j \in domain(C), 0 < j < |C|.$$

If $h_i(m, n_j) = 0$, then we think the item $|ln\frac{h_i(m+\Delta,n_j)}{h_i(m,n_j)}| = 0$. ε belongs to $(0, 1)$ and is introduced to avoid the case where $R_i(X)$ is denominator and $R_i(X) = 0$. Here, if $R_i(m)$ equals ε, it means the change ratio of values in the attribute C around m is zero.

Based on Definition 4 to 6, the novel semantic information loss for a numerical attribute Q_i is defined as follows:

Definition 7. $SILM_N$.
Suppose the domain of a numerical attribute Q_i is $[\min_i, \max_i]$, $v \in domain(Q_i)$ and is generalized into an interval which is denoted by v', $lower(v')$ and $upper(v')$ return the lower and upper bounds of v' respectively. Then the information loss of v' is denoted as follows:

$$\mathcal{SILM}_{\mathcal{N}} = \frac{\sum_{j=lower(v')}^{upper(v')} R_i(j)}{\sum_{j=min_i}^{\max_i} R_i(j)}$$

We can prove that in the case where a numerical attribute satisfies uniform distribution w.r.t the target attribute, or the numerical attribute is unrelated to the target attribute, the value of $SILM_N$ equals to that of traditional LM metric.

Corollary 1. *Let Q_i be a numerical attribute, the value distribution of Q_i is uniform w.r.t. the target attribute C, then the value of $\mathcal{SILM}_{\mathcal{N}}$ equals that of $\mathcal{LM}$.*

Corollary 2. *Let Q_i be a numerical attribute, Q_i is independent on the target attribute C, then the value of $\mathcal{SILM}_{\mathcal{N}}$ equals that of $\mathcal{LM}$.*

Proof. Let $p_r(n)$ be the original distribution probability of n in $domain(C)$. Note that Q_i satisfying the uniform distribution w.r.t. C equals Q_i being independent with C, thus $\forall m \in domain(Q_i)$ and $\forall n \in domain(C)$ we have $h_i(m + \Delta, n) = h_i(m, n) = p_r(n)$, so $R_i(x) = \max\{\ln |\frac{g_i(m,n)}{h_i(x,y)}| + \varepsilon\} = \max\{\ln 1 + \varepsilon\} = \varepsilon$, then we can induce

$$\mathcal{SILM}_{\mathcal{N}} = \frac{\sum_{j=lower(v')}^{upper(v')} R_i(j)}{\sum_{j=\min_i}^{\max_i} R_i(j)} = \frac{upper(v') - lower(v')}{\max_i - \min_i}$$

Obviously, according to the definition of LM, the proof of Corollary 1 and 2 is completed.

SILM for Categorical Attributes. To measure the information loss for categorical attributes which are denoted by $SILM_C$, we adopt Dissimilarity Matrix introduced in [15] to capture the semantic correlations between distinct values.

Definition 8. *(Dissimilarity Matrix). If a categorical attribute A has n distinct values, v_1, v_2, ..., v_n, respectively, dissimilarity matrix D(A), is a matrix having the following form where d(v_i, v_j) denotes the dissimilarity degree between v_i and v_j.*

$$\mathbf{D(A)} = \begin{pmatrix} 0 & d(v_1,v_2) & \dots & d(v_1,v_n) \\ d(v_2,v_1) & 0 & \dots & d(v_2,v_n) \\ \vdots & \vdots & \ddots & \vdots \\ d(v_n,v_1) & d(v_n,v_2) & \dots & 0 \end{pmatrix}, \qquad d(v_i,v_j) = d(v_j,v_i)$$

Assume that a set of distinct categorical values are generalized together, denoted by v' = {v$_1$, v$_2$, ..., v$_k$}. Then the loss caused by the generalization can be evaluated as follows:

$$\mathcal{SILM}_C = \frac{\sum_{p,q=1}^{|v'|} d(v_p,v_q)^2}{\sum_{p,q=1}^{|Q_c|} d(v_p,v_q)^2}$$

We can verify that the formula is reasonable by two special cases. First, suppose k similar values are generalized together, i.e., $v' = \{v_1, v_2, ..., v_k | v_i = v_j, 1 < i, j < k\}$, then according to the definition of Dissimilarity Matrix, d(v$_i$,v$_j$) = 0 for 1< i, j<k, thus we have $\mathcal{SILM}_C = 0$, which means no information loss. This result is in accordance with actual semantic meaning. Second, suppose all the domain values are generalized into one bucket, then $|v'| = |Q_c|$, thus we have $\mathcal{SILM}_C = 1$, that means the information loss is 100%. Again, this result agrees with the actual semantic.

SILM for Anonymous Tables. In summary, we give a formal definition of the semantic information loss for an anonymous table Y.

Definition 9. ***SILM***
Let $SILM(y_{i,j})$ denotes the semantic information loss of each generalized item $y_{i,j} \in Y$, $SILM(Y)$ denotes the total semantic information loss of Y, then we have:

$$\mathcal{SILM}(y_{i,j}) = \begin{cases} \dfrac{\sum_{k=lower(y_{i,j})}^{upper(y_{i,j})} R_j(k)}{\sum_{k=min_j}^{max_j} R_j(k)}, & Q_j \textit{ is a numeric attribute} \\[2ex] \dfrac{\sum_{p,q=1}^{|y_{i,j}|} d(v_p,v_q)^2}{\sum_{p,q=1}^{|Q_j|} d(v_p,v_q)^2}, & Q_j \textit{ is a categorical attribute} \end{cases} \tag{4}$$

$$\mathcal{SILM}(Y) = \sum_{i=1}^{|Y|} \sum_{j=1}^{d} \mathcal{SILM}(t_{i,j}) \tag{5}$$

5 s-RAC and s-DataFly

To evaluate the performance of our new metric SILM, we modified two anoymizaition algorithms, RAC(Random Clustering)[15] and DataFly[1], and replaced the original information loss metrics LM in them with SILM. The resulted two variation algorithms are called s-RAC and s-DataFly respectively. The reason for why s-RAC and s-DataFly are exploited here is because they are two typical algorithms in the field of anonymization, specifically, s-RAC adopts clustering and local recoding techniques, and s-DataFly is based on partitioning and global recoding techniques.

The implementation of s-RAC can refer to [15] and is omitted here. The main idea of s-RAC is very similar to the *k*-means clustering algorithm. For each tuple, the algorithm finds a suitable cluster to put the tuple into, where suitability is defined by SILM. If no such cluster is available, it creates a new cluster. When a cluster satisfies the *k*-anonymity principle, tuples in this cluster are generalized together and the cluster becomes a QI group. After all QI groups are formed, left tuples are inserted into existing QI groups according to the suitability defined by SILM. Because DataFly is one of the earliest and most widely-known *k*-anonymization solutions, the detailed implementation of s-DataFly is also omitted here.

6 Experiments

To evaluate various aspects of SLIM, we have conducted comprehensive experiments on both synthetic and real datasets. For anonymous data tables generalized by various metrics in the experiments, we compared their qualities in terms of accuracies of aggregate query answering and classification. We also investigated relationships among application accuracy, algorithms, and anonymization parameters(e.g., the anonymity requirement, *k*).

6.1 Experimental Settings

For the synthetic dataset, *StudentScore* consisting of 5 attributes *college major, study hours, credit, score, score class* was exploited, whose characters is shown in Table 7. The first three attributes were considered as QI attributes as well as predictor attributes, *score* was the sensitive attribute, and *score class* was the target attribute. To simulate the scenario where attribute *study hours* is strongly correlated with the target attribute *score class*, the values for *score* were generated according to Equation 6 and Equation 7. Totally 20000 tuples were generated for *StudentsScore*.

Table 7. StudentScore Dataset

Attribute	Distribution	Type of Attribute
college major	random one from set of { computer, math, physics, biology}	categorical
study hours	uniform integer in $[0, 50]$	numerical
credit	uniform integer in $[0, 4]$	numerical
score	uniform integer in $[S_l, S_h]$	numerical
score class	$[0, 60)$, $[60, 80)$, $[80, 100]$	target

$$S_l = \lfloor \sqrt{\frac{study\ hours + 50}{100}} \times factor(colledge\ major) \times 0.8 \rfloor \tag{6}$$

$$S_h = \lfloor \sqrt{\frac{study\ hours + 50}{100} \times factor(colledge\ major)} \rfloor \tag{7}$$

$$factor(x) = \begin{cases} 1, & x = computer \\ 0.9, & x = mathematics \\ 0.8, & x = physics \\ 0.8, & x = biology \end{cases}$$

For the real dataset, *Adult* from the UCI Machine Learning Repository[16] was exploited, which consists of 30162 records after removing records with missing values. In our experiments, only eight attributes *age, education, martial status, race, gender, hours per week, work class, salary* are retained, among which *age, education, martial status, race, gender, hours per week, work class* were considered as QI attributes as well as predictor attributes, and *salary* was designated as the sensitive attribute. For classification, a new target attribute *salary class* was added by transforming the numerical *salary* into a *salary class* ($< 50K$ or $\geq 50K$).

In our experiments, we adopted k-anonymity as the anonymity principle. Besides of algorithms s-RAC, s-DataFly, RAC and DataFly, we also implemented the Top-Down Specialization (TDS) algorithm[6] which incorporates a single target classification model to investigate the impact of a task-dependent metric. All of algorithms were built in Eclipse 3.1.2 with JDK 5.0, and executed on a dual-processor Intel Pentium D 2.8 GHz machine with 1 GB main memory running Microsoft Windows Server 2003.

6.2 Aggregate Query Answering Accuracy

Aggregate query answering is one of the most significant applications of anonymous datasets, so in this experiment we compared aggregate query answering accuracies of anonymous datasets generated based on different metrics to evaluate the performance of SILM. We used *StudentScore* as the tesing dataset and designed the query $SUM(A_i)$ in the following form:

$Select\ SUM(A_i)\ From\ Y\ Where\ A_1 = a_1\ And\ A_2 = a_2$

Here, A_i denote QI attribute. *Relative error rate* ($e - ratio$) was employed to measure the quality of returned results of a query. Specifically, let *est* and *act* be the results of a query q running over the anonymous table and an original table respectively, then $e - ratio = \frac{|act-est|}{act}$. The none/all principle was used to calculate *est*[15]. Suppose G' represents a QI group that satisfying the query, the value of A_i of G' is generalized to $[L_i, H_i]$, then we have $est = \sum_{\forall G'} |G'| \times H_i$.

Using different five algorithms(i.e., s-RAC, RAC, s-DataFly, DataFly and TDS), *StudentScore* was generalized into 5 anonymous tables. We processed a workload of 10,000 queries on these tables, and used *average error rate* to evaluate their answering ability. Figure 1 reports the result of the experiment. As shown in the figure, the query accuracy of the TDS-based table is much lower than that of the other four tables for different k values. This phenomenon conforms to what we have discussed in Section 4. When doing generalization, TDS does not consider the distortion of QI attributes at all and only focuses on the information intuitively related to the task of classification, such as information gain. Thus, compared to those task-independent metrics such as LM and SILM, task-dependent metrics would bring about much more information loss which furthermore affects the query answering accuracy.

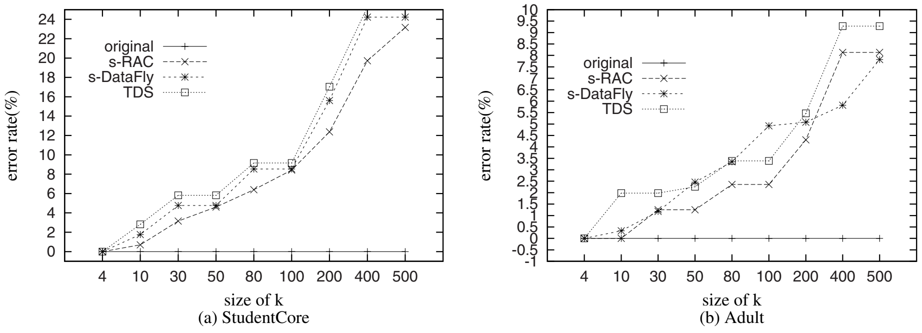


Fig. 1. Aggregate Query Answering Accuracy

6.3 Classification Accuracy

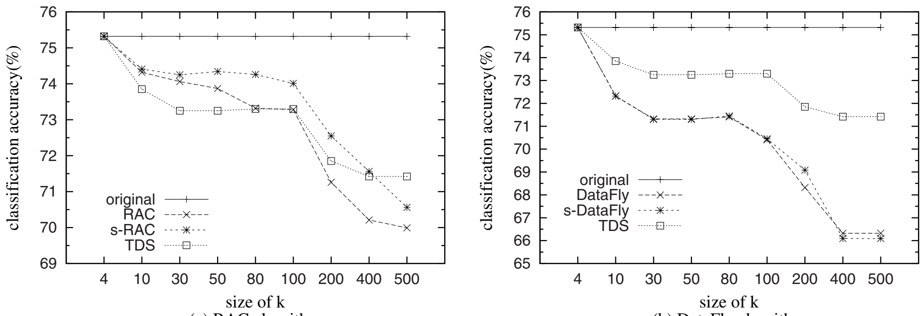
Methodology. To evaluate data qualities of anonymous data generated using SILM and other metrics, we also used the workload of classification. In comparison with works in [7], here only classifiers built on anonymous data were exploited to classify the original data. The original dataset was randomly partitioned into two equal sets, one for training and the other one for testing. Note that the training set must be anonymized before training. After then, the classifier built on anonymous training set was used to classify the testing set consisting of original data. We repeated the classification 10 times and shown average results in experimental figures.

To compare the classification accuracy of classifiers trained on anonymous data produced by different anonymization algorithms with different quality metrics, we also conducted a series of experiments on both real datasets and synthetic datasets. For classification models, we used the *libSVM* library[17] and the Naive Bayes classifier from Weka software package[18].

Comparison with previous task-independent metric. First, we evaluated effects of SILM through comparison with a typical task-independent information loss metric, i.e., LM. Four *k*-anonymous datasets were generated using four algorithms (s-RAC, RAC, s-DataFly, DataFly) based on the same training set. Then we compared classification accuracies yielded by these four anonymous data sets. The classification accuracy over the original data set was used as as the baseline, which is denoted as *Orignal*. The results are shown in Figure 2, Figure 3, Figure 4 and Figure 5. From the figures, we can see that no matter which algorithm was adopted, classifiers trained on SILM-based data set always outperformed than those classifiers trained on LM-based data set. Figure 2(a) depicts that when *k* becomes larger, SILM performs better(about 1.5% on the average, when k=80) than LM. Furthermore, when the dataset is synthetic, as shown in Figure 4, SILM almost achieve the same performance as baseline *Original* when $k < 400$. This is not surprising because SILM can capture the real correlation between predictor attributes and the target attribute.

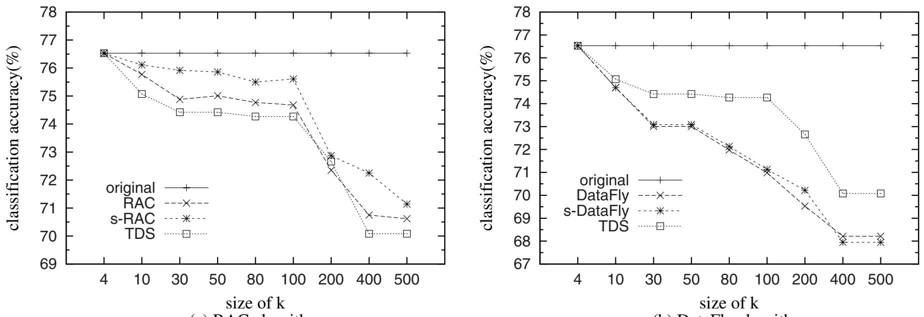
As shown from Figure 2 to Figure 5, when employing the same data quality metric, s-RAC always outperformed than s-DataFly. The superiority is more prominent when $k >$ 10. This is because s-DataFly is a partition-based algorithm and uses global recoding while s-RAC is cluster-based and uses local recoding. As we known, partitioning is less flexible and precise than clustering when grouping tuples according to their relationship. So s-DataFly is weak in producing high quality anonymous data.

From Figure 2 to Figure 5, all show that classification accuracies decreased while k increases. This is reasonable because larger QI group means more tuples are generalized together, more information loss is inevitable.



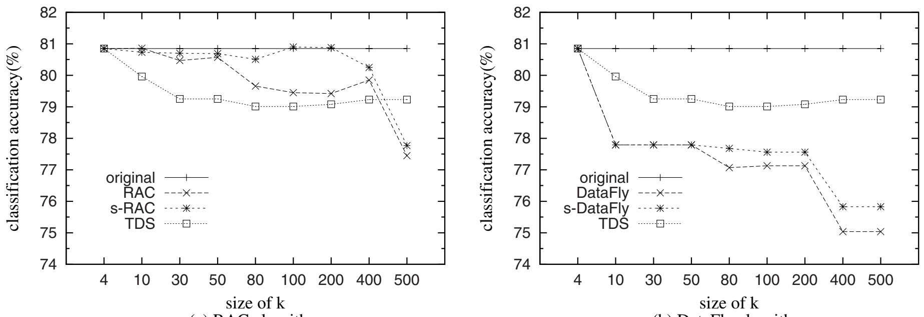
(a) RAC algorithm (b) DataFly algorithm

Fig. 2. Classification Accuracy of Naive Bayes (*Adult*)



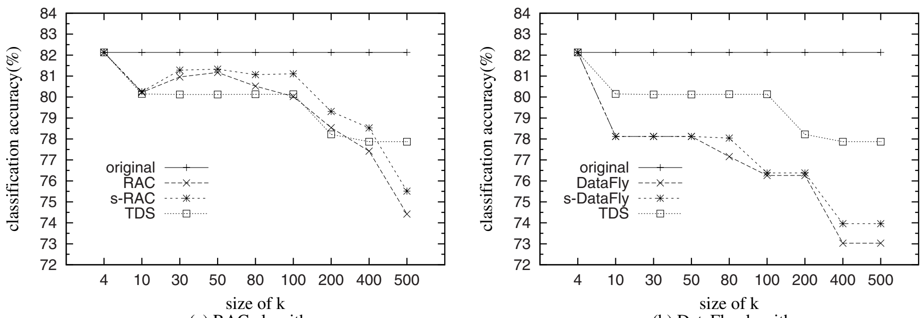
(a) RAC algorithm (b) DataFly algorithm

Fig. 3. Classification Accuracy of SVM (Adult)



(a) RAC algorithm (b) DataFly algorithm

Fig. 4. Classification Accuracy of Naive Bayes (StudentScore)



(a) RAC algorithm (b) DataFly algorithm

Fig. 5. Classification Accuracy of SVM (StudentScore)

Finally, comparing Figure 4 (Figure 5) with Figure 2 (Figure 3), it is shown that the effect of SILM on classification accuracy is more significant for the synthetic dataset *StudentScore* than for the real dataset *Adult*. This is because we intentionally emphasized the correlation between *score* and *studyhours* when generating the table. In other words, as a semantic metric, SILM is more sensitive to the dataset where strong correlations among attributes exist, and certainly performs much better for such data set especially in applications which aims to mine the correlations among attributes.

Comparison with previous task-dependent metric. Second, we compared SILM-based and LM-based algorithms with TDS to gage the performance of SILM in term of classification accuracy. Note that only TDS employs classification-conscious metric when doing generalization. Look at Figure 2(b), Figure 4(b), Figure 3(b) and Figure 5(b). It is not surprising to see that s-DataFly and DataFly performed not so well as TDS, because TDS is specially developed for classification and DataFly often causes large information loss due to its inherent limitations. But when referring to Figure 2(a), Figure 4(a), Figure 3(a) and Figure 5(a), it is unexpected that s-RAC almost always yields much higher accuracy than TDS. This phenomenon gives us such an insight that the superiority of task-dependent metric in generating task-specific anonymous data set is not deterministic. For example, TDS tried to achieve higher accuracy in classification at the expense of high information loss in anonymization and this even limited its utility for other applications. We think that the utility of anonymous data is in fact affected by multiple elements such as anonymization algorithms, recoding techniques, information loss metrics, potential applications and so on. Proper choice of these elements would help improve the utility of anonymous data. For example, by exploiting SILM and s-RAC algorithm, we can achieve better classification accuracy at the cost of lower information loss.

7 Conclusion and Future Work

In this paper, we propose a novel semantic information loss metric (SILM), which takes into account the correlations between QI attributes and target attribute when measuring the information loss. Theoretical analysis reveals that our metric is sound in semantic and is in accordance with the widely accepted LM metric in special cases. Extensive experiments also confirm that our metric can produce more useful anonymous datasets for more applications than previous metrics can.

Now we only consider the scenario that only one QI attribute is correlated with the target attribute. In fact, in most cases, more than one QI attribute have correlations with target attribute and the impact of these correlations on target attribute are various. In the future, we will investigate how to model these correlations and impacts in information loss metric.

References

1. Sweeney, L.: Achieving k-anonymity privacy protection using generalization and suppression. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems 10(5), 571–588 (2002)
2. Roberto, J., Bayardo Jr., Agrawal, R.: Data privacy through optimal k-anonymization. In: ICDE 2005, pp. 217–228 (2005)
3. LeFevre, K., DeWitt, D.J., Ramakrishnan, R.: Mondrian multidimensional k-anonymity. In: ICDE 2006, p. 25 (2006)
4. Xu, J., Wang, W., Pei, J., Wang, X., Shi, B., Fu, A.W.C.: Utility-based anonymization using local recoding. In: SIGKDD 2006, pp. 785–790 (2006)
5. LeFevre, K., DeWitt, D.J., Ramakrishnan, R.: Workload-aware anonymization. In: SIGKDD 2006, pp. 277–286 (2006)
6. Fung, B.C.M., Wang, K., Yu, P.S.: Top-down specialization for information and privacy preservation. In: ICDE 2005, pp. 205–216 (2005)
7. Inan, A., Kantarcioglu, M., Bertino, E.: Using anonymized data for classification. In: ICDE 2009, pp. 429–440 (2009)
8. Xiao, X., Tao, Y.: Personalized privacy preservation. In: SIGMOD 2006, pp. 229–240 (2006)
9. Xiao, X., Tao, Y.: M-invariance: towards privacy preserving re-publication of dynamic datasets. In: SIGMOD 2007, pp. 689–700 (2007)
10. Xiao, X., Tao, Y.: Anatomy: Simple and effective privacy preservation. In: VLDB 2006, pp. 139–150 (2006)
11. Du, Y., Xia, T., Tao, Y., Zhang, D., Zhu, F.: On multidimensional k-anonymity with local recoding generalization. In: ICDE 2007, pp. 1422–1424 (2007)
12. Aggarwal, G., Feder, T., Kenthapadi, K., Khuller, S., Panigrahy, R., Thomas, D., Zhu, A.: Achieving anonymity via clustering. In: Proc. of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 153–162 (2006)
13. Li, J., Wong, R.C.W., Fu, A.W.C., Pei, J.: Anonymization by local recoding in data with attribute hierarchical taxonomies. IEEE Trans. Knowl. Data Eng. 20(9), 1181–1194 (2008)
14. Iyengar, V.S.: Transforming data to satisfy privacy constraints. In: SIGKDD 2002, pp. 279–288 (2002)
15. Liu, Y., Lv, D., Ye, Y., Feng, J., Hong, Q.: Set-expression based method for effective privacy preservation. In: Proc. of the Ninth International Conference on Web-Age Information Management, pp. 325–332 (2008)
16. Newman, D., Hettich, S., Blake, C., Merz, C.: Uci repository of machine learning databases (1998), http://www.ics.uci.edu/~mlearn/MLRepository.html
17. Chang, C.C., Lin, C.J.: Libsvm: a library for support vector machines (2001), http://www.csie.ntu.edu.tw/cjlin/libsvm
18. Witten, I.H., Frank, E.: Datamining:Practiacal machine learning tolls and techniques, 2nd edn. Prentice-Hall, Englewood Cliffs (2000)

On t-Closeness with KL-Divergence and Semantic Privacy

Chaofeng Sha[1], Yi Li[1], and Aoying Zhou[2]

[1] School of Computer Science, Fudan University, China
{cfsha,liy}@fudan.edu.cn
[2] Software Engineering Institute, East China Normal University, China
ayzhou@sei.ecnu.edu.cn

Abstract. In this paper, we study how to sanitize the publishing data with sensitive attribute to achieve t-closeness and δ-disclosure privacy under Incognito framework. t-closeness is a privacy measure proposed to account for skewness attack and similarity attack, which are limitations of l-diversity. Under the t-closeness model, the distance between the privacy attribute distribution and the global one should be under the threshold t. Whereas semantic privacy (δ-disclosure privacy) is used to measure the incremental information gain from the anonymized tables. We use the Kullback-Leibler divergence to measure the distance between distributions and discuss the properties of the semantic privacy. We also study the relationship between t-closeness with KL-divergence and semantic privacy, and show that t-closeness with KL-divergence and δ-disclosure privacy satisfy the generalization property and the subset property, which entail us to use the Incognito algorithm. Experiments demonstrate the efficiency and effectiveness of our approaches.

1 Introduction

Many organizations and corporations publish data for information sharing and research. It results in the issues of privacy protection. Researchers [16,18] showed that simple removal of identifier attributes, such as name and social security number, can not protect the privacy of individual sensitive information because a specific individual can be identified by joining several public data sets. In the point of view of data management, we study how data could be sanitized to meet the wide requirements of privacy preservation under the assumption that identifier attributes have been deleted and remainder attributes including quasi-identifier and sensitive attributes.

Now there are already several privacy models or measures such as k-anonymity, l-diversity, t-closeness and δ-disclosure privacy. k-anonymity requires that every record is indistinguishable with at least $k - 1$ other records in the table, i.e., have the same quasi-identifier attributes, which is called an equivalence class. Some common techniques to achieve k-anonymity are generalization, suppression and clustering. While k-anonymity is vulnerable to the homogeneous attack, which means an adversary can infer exactly the individual's sensitive value when the sensitive values in an equivalence class are same. So l-diversity is proposed, which requires the sensitive values in each equivalence class can not be too skew, for example, the entropy of sensitive attribute distribution is larger than a threshold. Although l-diversity solved the problem with k-anonymity, it can be attacked by skewness attack and similarity attack through analyzing the semantic of sensitive attributes. The skewness attack is possible if there is a significant difference between the sensitive attribute distribution of some equivalence classes and global one. And similarity attack is possible if the sensitive attribute values in an equivalence class are similar. And then adversary can infer the sensitive value of some individuals.

Through analyzing the risk of privacy leakage in all previous methods, [11] proposed t-closeness, which requires that the distance between the distribution of sensitive value in every equivalence class and the global one is less than a threshold t. [11] uses Earth Mover's Distance to measure the distance between two distributions. Although it can handle several attribute types, there is no explicit expression for general case and the ground distance should be manually set for specific applications.

Our work is mainly based on works of [4,11] and the framework of Incognito algorithm[9]. We propose the t-closeness with Kullback-Leibler divergence, which has two necessary properties required by Incognito algorithm. Then we study semantic privacy (disclosure privacy), through analyzing its strength and properties. Furthermore, the experiments demonstrate the efficiency of our algorithm and quality of anonymized data.

The main contributions of this paper are as follows: 1). We use Kullback-Leibler divergence to measure the distance between distributions of sensitive value, and prove that Kullback-Leibler divergence has the generalization property and subset property required by Incognito algorithm. 2). We reexamine semantic privacy, named δ-disclosure privacy proposed in [4], and furthermore discuss its properties, includes proving that semantic privacy is stronger than t-closeness with KL-divergence. 3). We implement t-closeness with KL-divergence and δ-disclosure privacy under the framework of Incognito algorithm, and demonstrate the performance of our approach and verify the quality of the anonymized data.

The rest of this paper is organized as follows. In Section 2 we introduce several privacy models and discuss their potential drawbacks, and focus on t-closeness with EMD. In Section 3 we propose the t-closeness with KL-divergence, and study its properties. We reexamine the δ-disclosure privacy, KL-divergence and their properties, study the relationship between t-closeness with KL-divergence and δ-disclosure privacy. In Section 5 we report experiments on real data sets and demonstrates the performance of our methods and the quality of anonymized data. Section 6 summarizes the related work and applications of KL-divergence. We conclude our work in Section 7.

2 Some Basic Models

This section is devoted to a brief review of several privacy preservation models. Given a data table $T = \{t_1, \cdots, t_n\}$, where a record t_i represents an individual with several attributes. Let $A = \{A_1, \cdots, A_a\}$ be the attribute set and $t[A_i]$ be the value of attribute A_i of record t. The common model is adopted, where a table T consists of a quasi-identifier attribute set, QI, such as Zip Code, gender, and age, which can be used to identify individual, and a set S consists of the sensitive value such as disease. As mentioned in previous section, an equivalence class consists of records with the same

QI, namely if two records t_i, t_j satisfy $t_i[QI] = t_j[QI]$, they belong to the same equivalence class. We also adapt S in this paper to denote the set of all sensitive value, e.g. a set of all diseases. And we only discuss the case with single sensitive attribute. These concepts, the k-anonymity model and generalization will be introduced through a simple example. In Table 1a, name and the other information used for identification have been deleted, quasi-identifier is $\{$ZIP Code, Age, Sex$\}$, sensitive attribute is Disease. Even identity information has been removed, Cancer disease can still be inferred if a male has the properties of ZIP code 47902 and age 50.

Due to the risk of privacy leakage, [16] introduced k-anonymity model. If an equivalence class E in table T has at least k records, it satisfies k-anonymity. If all equivalence classes satisfy k-anonymity, the table satisfies k-anonymity also. Generalization and suppression techniques have been developed to achieve k-anonymity [17]. By generalization, the QI value is replaced by more general one, such as ZIP Code with 47612 and 47620 would be replaced as 476** and gender $\{$Female,Male$\}$ would be replaced as *. Generally, there is a hierarchical generalization tree defined on each attribute. So we can define total order on generalization among attribute sets, where G_2 is more generalized than G_1 on attribute set A, denoted as $G_1 \leq G_2$, if G_1 is less generalized than G_2 for every attribute in A. For more information about hierarchical generalization tree the readers can be referred to [9]. While through suppression we just delete a quasi-identifier value or a whole record from a table.

Table 1. Original Data and Anonymized Table

ZIP Code	Age	Sex	Disease
47612	25	Male	Viral Infection
47620	28	Female	Viral Infection
47901	45	Female	Heart Disease
47902	50	Male	Cancer

(a) Original Table

ZIP Code	Age	Sex	Disease
476**	2*	*	Viral Infection
476**	2*	*	Viral Infection
4790*	≥ 40	*	Heart Disease
4790*	≥ 40	*	Cancer

(b) Anonymized Table: 2-anonymous

After generalization of data in Table 1a, we have the new data in Table 1b, where the first two records and the last two records form an equivalence class respectively. It is obvious that the table 1b is 2-anonymous. Now we can not infer the exact disease of the male with ZIP Code 47902 and age 50. But k-anonymity still has a flaw. It produces a group with the same sensitive value, where all sensitive attributes have the same value in some equivalence classes. Using homogeneous attack, an adversary can still successfully deduce the sensitive value. Looking at the first equivalence class in Table 1b, if we know someone is in this equivalence class, we know that he has Viral Infection.

This flaw was first found in [13]. And l-diversity was proposed consequently. Before the formal definition, several notations will be first introduced. $p_E(s)$ is the probability of sensitive attribute $s \in S$ in equivalence class E, namely $p_E(s) = \frac{f_E(s)}{|E|} = \frac{|\{t \in E : t[S]=s\}|}{|E|}$, where $f_E(s)$ is the frequency of sensitive value s in E. Similarly, $p_T(s)$ is the probability of sensitive attribute $s \in S$ in table T, namely $p_T(s) = \frac{f(s)}{|T|} = \frac{|\{t \in T : t[S]=s\}|}{|T|}$, where $f_T(s)$ is the frequency of sensitive value s in T.

Definition 1 (Entropy l-diversity). *A table is Entropy l-diverse if for each equivalence class E: $H(p_E) = -\sum_{s \in S} p_E(s) \log p_E(s) \geq \log l$.*

Many works (e.g. [7,19]) devised some anonymization methods for l-diversity, which also has flaws and suffers to some attacks like homogeneous attack to k-anonymity. Two important attacks are skewness attack and similarity attack, where the sensitive attribute of l different values is similar in an extent or naturally the same in some equivalence classes for l-diversity. For example, in an anonymized salary table satisfying l-diversity, all salary values are in the range of 1000-1500 in an equivalence class which are much lower than the average salary 3000. So an adversary can infer that they are low incomes employee. The more skewness attack and similarity attack can be referred to [11]. Consequently, [11] proposed t-closeness privacy measure. It strictly requires that the distance between distribution of sensitive value of each equivalence class and the global one is less than a threshold, which means that an adversary can only gain much less posterior knowledge from anonymized data and hence enhances the privacy.

Definition 2 (t-closeness). *An equivalence class E is said to have t-closeness if the distance between the distribution of a sensitive attribute in this class, p_E, and the distribution of the attribute in the whole table, p_T, is no more than a threshold t. A table is said to have t-closeness if all equivalence classes have t-closeness.*

The parameter t in t-closeness enables one to trade off between utility and privacy. Now the problem is to measure the distance between two probabilistic distributions. There are a number of ways to define the distance between them, which can be variational distance, Hellinger distance, Kullback-Leibler divergence, or Earth Mover's Distance (EMD) in [11]. The EMD is based on the minimal amount of work needed to transform one distribution to another by moving distribution mass between each other. While the original definition of EMD does not give a explicit expression of EMD distance between any two distributions. Some explicit formulas were given in [11] for special cases with numerical and categorical type. The readers can be referred to it further. And as pointed out in [4], the EMD is an additive (as opposed to multiplicative) measure, and does not translate directly into a bound on the adversary's ability to learn sensitive attributes associated with a given quasi-identifier.

3 t-Closeness with KL-Divergence

This paper focuses on anonymization problem of discrete (categorical) sensitive attribute. We will study the properties of t-closeness with KL-divergence and semantic privacy. Here we use KL-divergence for it is multiplicative, relatively efficient to calculate and does not need to define ground distance. Then we can establish the relationship between t-closeness with KL-divergence and semantic privacy.

Definition 3. *Given two probability distributions $p = (p_1, ..., p_N)$ and $q = (q_1, ..., q_N)$, the Kullback-Leibler (KL) divergence between p and q is defined as follows:*

$$KL(p||q) = \sum_{i=1}^{N} p_i \log \frac{p_i}{q_i}$$

Here $0 \log 0 = 0$ *by convention.*

Now we consider the t-closeness with KL-divergence as distance. The t-closeness can be reformulated as follows:

Definition 4 (t-closeness with KL-divergence). *An equivalence class E has t-closeness if the KL-divergence $KL(p_E||p_T)$, between the sensitive attribute distribution of class E and the one of whole table T is less than a threshold t. A table T satisfies t-closeness if all equivalence classes have t-closeness.*

Consider the example in [11], which has the EMD distance 0.1 between two distribution $(0.01, 0.99)$ and $(0.11, 0.89)$ and the EMD distance is also 0.1 between distribution $(0.4, 0.6)$ and $(0.5, 0.5)$. For these two pairs, we have $\max\left\{\left|\log \frac{0.11}{0.01}\right|, \left|\log \frac{0.89}{0.99}\right|\right\} = 3.4594$ and $\max\left\{\left|\log \frac{0.5}{0.4}\right|, \left|\log \frac{0.5}{0.6}\right|\right\} = 0.3219$ respectively.

Obviously there is a remarkable difference between two pairs of distributions. Furthermore, we can also demonstrate an application of KL-divergence to t-closeness through another example with data set as Table 2 [13], where Disease is the sensitive attribute.

Table 2. Anonymized Table

	ZIP Code	Age	Nationality	Disease
1	476**	2*	*	Heart Disease
2	476**	2*	*	Viral Infection
3	476**	2*	*	Cancer
4	476**	2*	*	Cancer
5	4790*	≥ 40	*	Viral Infection
6	4790*	≥ 40	*	Heart Disease
7	4790*	≥ 40	*	Viral Infection
8	4790*	≥ 40	*	Cancer
9	476**	3*	*	Cancer
10	476**	3*	*	Cancer
11	476**	3*	*	Viral Infection
12	476**	3*	*	Heart Disease

The distributions in Table 3 can be computed from Table 2, where E_1, E_2, E_3 correspond to 3 equivalence classes separated by horizontal line respectively. After simple calculation, the KL-divergences between sensitive attribute distribution of each equivalence class and the global one are $KL(p_{E_1}, p_T) = KL(p_{E_3}, p_T) = 0.0278$ and $KL(p_{E_2}, p_T) = 0.1082$ respectively. Therefore the anonymized table has 0.1082-closeness.

3.1 Properties of t-Closeness with KL-Divergence

In this subsection, we will show that t-closeness with KL-divergence holds several properties, such generalization subset and rollup. Then we can implement it in the Incognito algorithm framework.

Table 3. Table of Probability Distribution

	Cancer	Heart Disease	Viral Infection
p_T	5/12	1/4	1/3
p_{E_1}	1/2	1/4	1/4
p_{E_2}	1/4	1/4	1/2
p_{E_3}	1/2	1/4	1/4

Lemma 1. *Given two equivalence classes E_1 and E_2, the sensitive attribute distributions on E_1, E_2, and $E_1 \cup E_2$ are denoted as P_1, P_2, and P respectively. We have:*

$$KL(P||p_T) \leq \frac{|E_1|}{|E_1| + |E_2|} KL(P_1||p_T) + \frac{|E_2|}{|E_1| + |E_2|} KL(P_2||p_T)$$

Proof. By the condition in the lemma,

$$P(s) = \frac{|E_1|}{|E_1| + |E_2|} P_1(s) + \frac{|E_2|}{|E_1| + |E_2|} P_2(s)$$

KL-divergence is convex in terms of first variable [6]. That means for any three distributions p_1, p_2 and q, we have that $KL(\lambda p_1 + (1 - \lambda)p_2||q) \leq \lambda KL(p_1||q) + (1 - \lambda)KL(p_2||q)$, where $0 \leq \lambda \leq 1$.

Combining the previous two facts, we prove the inequality in this lemma.

It follows that $KL(P||p_T) \leq \max\{KL(P_1||p_T), KL(P_2||p_T)\}$. This means that when two equivalence classes are being merged, the maximum KL-divergence between any sensitive attribute distribution of equivalence class from the overall distribution can never increase.

With Lemma 1, we can prove that t-closeness with KL-divergence has two properties, which are required to use the Incognito generalization algorithm which has been used in previous works to achieve k-anonymity, l-diversity or t-closeness.

Property 1 (Generalization Property). Let T be a table, and let G_1 and G_2 be two generalizations on T such that G_2 is more general than G_1 ($G_1 \leq G_2$). If T satisfies t-closeness using G_1, then T also satisfies t-closeness using G_2.

Proof. Assume that generalized by G_1, $T = E_{1,1} \cup E_{1,2} \cup \cdots \cup E_{1,g_1}$, and T satisfies t-closeness, i.e., for each equivalence class $E_{1,j}$, $(1 \leq j \leq g_1)$ we have $KL(p_{E_{1,j}}||p_T) \leq t$. Then by generalization G_2, $T = E_{2,1} \cup E_{2,2} \cup \cdots \cup E_{2,g_2}$. Due to the fact that $G_1 \leq G_2$, then we know that each equivalence class $E_{2,i}$ is a union of some equivalence classes $E_{1,j}$s. It means that there exist $E_{1,j_1}, \cdots, E_{1,j_m}$ such that $E_{2,i} = E_{1,j_1} \cup \cdots \cup E_{1,j_m}$. Therefore,

$$KL(p_{E_{2,i}}||p_T) \leq \max\{KL(p_{E_{1,j_1}}||p_T), \cdots, KL(p_{E_{1,j_m}}||p_T)\} \leq t$$

namely equivalence class $E_{2,i}$ satisfies t-closeness. And then table T satisfies t-closeness.

Property 2 (Subset Property). Let T be a table and let A_1 be a set of attributes in T. If T satisfies t-closeness with respect to A_1, then T also satisfies t-closeness with respect to any set of attributes A_2 such that $A_2 \subseteq A_1$.

Proof. The equivalence class with respect to attribute set A_2 is a union of some equivalence classes with respect to attribute set A_1 in table T. If every equivalence class has t-closeness with respect to A_1, the equivalence class also has t-closeness with respect to A_2. It means table T has t-closeness with respect to A_2.

During the generalization, we need the following auxiliary information of equivalence class: the number of records and frequency of each sensitive value for computing probability. Therefore we need rollup property [9], which is similar to the operation along dimension hierarchies in OLAP processing.

Property 3 (Rollup Property). Given a table T and two attribute sets A_1 and A_2 of T with $A_1 \subseteq A_2$. If we have the frequency set f_1 of T with respect to A_1, then the frequency set f_2 of T with respect to A_2 is a summation of item in frequency set f_1 along the path of generalization from A_1 to A_2.

3.2 t-Closeness with JS-Divergence

In this subsection, we briefly discuss t-closeness with JS-divergence. Privacy leaks occur only when the adversary learns sensitive information beyond the overall distribution $p_T(s)$. To measure the privacy loss, the following JS-divergence was introduced in [12]:

$$JS(p,q) = \frac{1}{2}\left(KL\left(p||\frac{p+q}{2}\right) + KL\left(q||\frac{p+q}{2}\right)\right).$$

This divergence measure avoids the illness of KL-divergence when there are zero probabilities in the second distribution p. And we have the following property that the JS-divergence is also convex.

Property 4 (Convexity of JS-divergence). For any three distributions p_1, p_2 and q, we have:

$$JS(p, \lambda q_1 + (1-\lambda)q_2) \leq \lambda JS(p,q_1) + (1-\lambda)JS(p,q_2)$$

where $0 \leq \lambda \leq 1$.

Therefore, we can also use the Incognito algorithm to implement the t-closeness with JS-divergence as the discussion of t-closeness with KL-divergence in the above subsection. We omit the details due to the space limit.

3.3 Relationship with t-Closeness with EMD

To use t-closeness with EMD, we need to calculate the EMD between two distributions. Although we can calculate EMD using min-cost flow algorithms, these algorithms do not provide an explicit formula [11]. Therefore some explicit formula was provided in [11] to compute the EMD between distribution. For categorical attributes, the equal distance is as follows [11]: $D[p,q] = \frac{1}{2}\sum_{i=1}^{m}|p_i - q_i|$.

By Pinsker's inequality [6], we know that $KL(p||q) \geq 2D^2[p,q]$. Therefore, t-closeness with KL divergence is $\frac{1}{2}\sqrt{t}$-closeness with EMD.

3.4 Relationship with Entropy l-Diversity

According to the definition of KL-divergence, we have that

$$KL(p_E||p_T) = \sum_{i=1}^{N} p_E(s_i) \log \frac{1}{p_T(s_i)} - H(p_E).$$

Therefore, to achieve l-diversity and t-closeness with KL-divergence simultaneously for some publishing data, the following condition should be satisfied for each equivalence class E in the sanitized table: $\sum_{i=1}^{N} p_E(s_i) \log \frac{1}{p_T(s_i)} \leq t + l$.

4 Semantic Privacy

As shown in [4], an adversary can still obtain the global sensitive value distribution, namely ground distribution $(p_T(s_1), \cdots, p_T(s_N))$, even if all microdata are published by generalizing all quasi-identifier as the most general one. We hope that the published table T', which is a sanitization of original table T, has the property that the distribution of sensitive attribute is very close to ground distribution in each equivalence class [11]. In order to measure the incremental gain obtained by an adversary through sanitized table T', [4] introduced the following definition.

Definition 5. *(δ-DISCLOSURE PRIVACY). We say that an equivalence class E is δ-disclosure private with respect to the sensitive attributes S if, for any $s \in S$ which appearing in E*

$$DP(E, s) = \left| \log \frac{p_E(s)}{p_T(s)} \right| \leq \delta$$

A table T is δ-disclosure private if any equivalence E is δ-disclosure private.

It is important to note that the above inequality should be satisfied for all sensitive values $s \in S$ according to the definition in [4]. This requirement is too strong. For example, some equivalence class may not include all sensitive value. While the sensitive value with small frequency $p_T(s)$ in a table may not occur in some equivalence class in the case of t-closeness, which means $p_E(s) = 0$. Here we only require that the sensitive attribute values s which appear in some equivalence class satisfy that inequality.

In the rest of this subsection, S is a random variable of sensitive attribute, s is a specific sensitive value, E is a random variable denoting equivalence class, and e is a equivalence class instance, where $p(s) = p_T(s)$ and $p(s|e) = p_E(s)$. With these notations, δ-disclosure privacy is reformulate as $\left|\log \frac{p(s|e)}{p(s)}\right| \leq \delta$. Firstly, we introduce the following lemma established in [4].

Lemma 2. *If table T satisfies δ-disclosure privacy, then we have $H(S) - H(S|E) \leq \delta$.*

As noted in [4], this lemma shows that when a table satisfies δ-disclosure privacy, the ability to build a predictor for sensitive attribute S based on the quasi-identifier QI is bounded by δ. That means that the requirement of δ-disclosure privacy is stronger than the bound provided in the above lemma. Because in disclosure privacy definition, we force the distributions $\{p(s) : s \in S\}$ and $\{p(s|E) : s \in S\}$ to be similar, not just have the bounded difference between their entropy.

Even though t-closeness does not directly bound the gain in adversary's knowledge, it is similar in its spirit to semantic privacy; it, too, attempts to capture the difference between the adversary's baseline knowledge and the knowledge he gains from the quasi-identifier of some equivalence classes in the sanitized table. As parameters (t and δ, respectively) approach 0, both t-closeness and δ-disclosure privacy converge to statistical independence of quasi-identifiers and sensitive attributes within the sanitized database [4]. While compared to semantic privacy, t-closeness with KL-divergence in previous section is relatively weaker. That we have the following result.

Lemma 3. *If T satisfies δ-disclosure privacy, then it has δ-closeness. For any equivalence class E in T, we have*

$$KL(p_E||p_T) \leq \delta.$$

Proof. By the fact that KL-divergence is nonnegative, we have

$$\begin{aligned} KL(p_E||p_T) &= \sum_{i=1}^{N} p_E(s_i) \log \frac{p_E(s_i)}{p_T(s_i)} = \left| \sum_{i=1}^{N} p_E(s_i) \log \frac{p_E(s_i)}{p_T(s_i)} \right| \\ &\leq \sum_{i=1}^{N} p_E(s_i) \left| \log \frac{p_E(s_i)}{p_T(s_i)} \right| \\ &= \sum_{i:p_E(s_i)\neq 0} p_E(s_i) \left| \log \frac{p_E(s_i)}{p_T(s_i)} \right| + \sum_{i:p_E(s_i)=0} p_E(s_i) \left| \log \frac{p_E(s_i)}{p_T(s_i)} \right| \end{aligned}$$

By the fact that for any sensitive attribute s_i in the equivalence class E, we have $p_E(s_i) \neq 0$, and $\left|\log \frac{p_E(s)}{p_T(s)}\right| \leq \delta$. While for any sensitive attribute s_i absent in E, we have $p_E(s_i) = 0$. And by the convention that $0 \log 0 = 0$, we have

$$\begin{aligned} KL(p_E||p_T) &\leq \sum_{i:p_E(s_i)\neq 0} p_E(s_i) \left| \log \frac{p_E(s_i)}{p_T(s_i)} \right| \\ &\leq \sum_{i:p_E(s_i)\neq 0} p_E(s_i)\delta \leq \delta \sum_{i=1}^{N} p_E(s_i) = \delta \end{aligned}$$

Now we examine some properties of semantic privacy, which have not been developed explicitly in the original work [4].

Lemma 4. *Given two equivalence classes E_1 and E_2, let E be the union of E_1 and E_2, then for any sensitive attribute $s \in S$ we have*

$$DP(E, s) \leq \max\{DP(E_1, s), DP(E_2, s)\}$$

Proof. The equivalence class E is the union of equivalence classes E_1 and E_2, means that for any $s \in S$: $p(s|E) = \frac{|E_1|}{|E_1|+|E_2|}p(s|E_1) + \frac{|E_2|}{|E_1|+|E_2|}p(s|E_2)$.

Thus, we have

$$\begin{aligned} DP(E,s) &= \left|\log \frac{p(s|E)}{p(s)}\right| = \left|\log \frac{\frac{|E_1|}{|E_1|+|E_2|}p(s|E_1) + \frac{|E_2|}{|E_1|+|E_2|}p(s|E_2)}{\frac{|E_1|}{|E_1|+|E_2|}p(s) + \frac{|E_2|}{|E_1|+|E_2|}p(s)}\right| \\ &\le \left|\log \max\left\{\frac{\frac{|E_1|}{|E_1|+|E_2|}p(s|E_1)}{\frac{|E_1|}{|E_1|+|E_2|}p(s)}, \frac{\frac{|E_2|}{|E_1|+|E_2|}p(s|E_2)}{\frac{|E_2|}{|E_1|+|E_2|}p(s)}\right\}\right| \\ &\le \max\left\{\left|\log \frac{p(s|E_1)}{p(s)}\right|, \left|\log \frac{p(s|E_2)}{p(s)}\right|\right\} \\ &= \max\{DP(E_1,s), DP(E_2,s)\} \end{aligned}$$

where the first inequality is due to the fact $\frac{a+b}{c+d} \le \max\left\{\frac{a}{c}, \frac{b}{d}\right\}$ for any $a,b,c,d \ge 0$.

According to the Lemma 4, semantic privacy (δ-disclosure privacy) has two properties, which are necessary to use Incognito framework to generalize microdata so that the anonymized table satisfies k-anonymity, l-diversity or t-closeness.

Property 5 (Generalization Property). Given the table T, G_1 and G_2 are generalizations of T, and $G_1 \le G_2$, if T is δ-disclosure private with respect to G_1, T is δ-disclosure private with respect to G_2.

Proof. With generalization G_1, assume $T = E_{1,1} \cup E_{1,2} \cup \cdots \cup E_{1,g_1}$, which is of δ-disclosure privacy, So for every equivalence class $E_{1,j}$, $(1 \le j \le g_1)$ and any sensitive value $s \in S$, $DP(E_{1,j}, s) \le \delta$ holds. While with generalization G_2, there is $T = E_{2,1} \cup E_{2,2} \cup \cdots \cup E_{2,g_2}$ because of $G_1 \le G_2$, Because each equivalence class $E_{2,i}$ is a union of some equivalence classes $E_{1,j}$s, which means that there are $E_{1,j_1}, \cdots, E_{1,j_m}$ such that $E_{2,i} = E_{1,j_1} \cup \cdots \cup E_{1,j_m}$. Then we have:

$$DP(E_{2,i}, s) \le \max\{DP(E_{1,j_1}, s), \cdots, DP(E_{1,j_m}, s)\} \le \delta.$$

It means that the equivalence class $E_{2,i}$ is δ-disclosure private. So the table T has δ-disclosure private.

Property 6 (Subset Property). Given table T, A_1 is a subset of attributes in T. If T is δ-disclosure private with respect to A_1, for any $A_2 (A_2 \subseteq A_1)$, T is also δ-disclosure private with respect to A_2.

Proof. Since each equivalence class with respect to A_2 is the union of some equivalence classes with respect to A_1 and each equivalence class with respect to A_1 satisfies δ-disclosure privacy, we conclude that each equivalence class with respect to A_2 also satisfies δ-disclosure privacy. Thus T satisfies δ-disclosure privacy with respect to A_2.

With these two Properties and Rollup Property mentioned before, we can utilize the Incognito algorithm. For more details of Incognito algorithm, the reader is refereed to [9].

5 Experimental Evaluation

In this section, we will demonstrate the performance of our method and verify the quality of the anonymized data. We first show that our method is as efficient as the other methods. Then we verify the data generated by anonymization algorithm, which satisfies t-closeness or δ-disclosure privacy, has rather good quality. The implementation of Incognito algorithm is based on the Java source code implemented by database group of Cornell University [1], which is modified to achieve t-closeness with KL-divergence and semantic privacy. All experiments are running on a machine with Intel Pentium4 2.4GHz CPU, 2GB memory, and Microsoft Windows XP Professional operating system.

In all experiments, we also use the Adult data set from UCI machine learning repository, which is composed of 45222 records. After removal of records with some missing value, there are still 30162 records. We run experiments on 8 attributes of the data set with the same generalization in [11]. Table 4 lists attributes used, the number of different value of each attribute, the type of generalization and the height of hierarchical generalization tree of each attribute.

Table 4. UCI Adult Data Set

Attribute	Values	Generalization	Height
Age	74	numeric	5
Workclass	7	categorical	3
Education	16	categorical	4
Country	41	categorical	3
Martial Status	7	categorical	3
Race	5	categorical	3
Sex	2	categorical	2
Salary	2	sensitive	2
Occupation	14	sensitive	3

5.1 Performance

The time complexity of Incognito algorithm and the others, is an exponential function of the number of quasi-identifiers. But in practice, the real running time is acceptable. We have compared the time performance of 4 privacy measures: k-anonymity without t-closeness, entropy l-diversity, t-closeness with KL-divergence and δ-disclosure privacy, where the setting of specific parameters is given in the figures of experiments.

For Adult data set, Occupation is set as the sensitive attribute and the set of quasi-identifier varies from 2 to 7 attributes. If the size of QI is i, the first i attributes are taken as quasi-identifier. Similar to the experiments in [11], we set $k = 5, l = 5, t = 1.4, \delta = 1.4$ to k-anonymity, entropy l-diversity, t-closeness, δ-disclosure privacy, respectively. As can be seen from the figures, the time of generalization for δ-disclosure privacy and t-closeness is longer than it for k-anonymity without t-closeness

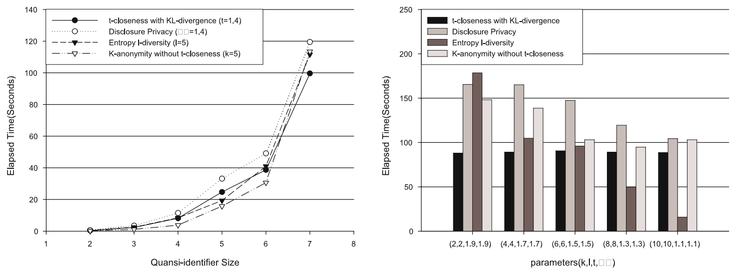


Fig. 1. Varied QI Size for $k = 5, l = 5$

and entropy l-diversity. But it is still acceptable. And the time curve of k-anonymity and entropy l-diversity respectively is nearly the same as the result given by experiments in [13].

We now fix the number of quasi-identifier as the first 4 of 7 attributes and vary the parameters k, l, t, δ. And we get the result shown in Figure 1. As shown in figure, entropy l-diversity spends least time for generalization, this maybe due to the pruning process will be executed much earlier while l increasing during the generalization. For generalization of t-closeness with KL-divergence, t has no notable effect on running time over Adult data set.

5.2 Data Quality

We take first 4 of 7 attributes in table as quasi-identifier and Occupation as sensitive attribute, which is the same as setting in [11]. Discernibility[3] and Minimal Average Size [9] are taken to measure the quality of data generated by anonymization with 4 privacy models. The discernibility metric measures the number of tuples that are indistinguishable from each other. Each tuple in an equivalence class E incurs a cost $|E|$ and each tuple that is completely suppressed incurs a cost $|D|$ (where D is the original dataset). Since we did not perform any tuple suppression, the discernibility metric is equivalent to the sum of the squares of the sizes of the equivalence classes. The average size of the equivalence classes generated by the anonymization algorithm. To evaluate the average size of equivalence classes generated by anonymization algorithm, [13] even discussed the effects of data skewness to these metrics.

With different value of parameter k, l, t, δ, we get results in Figure 2 for Discernibility metric. It is observed that the value of metric is relative small of δ-disclosure privacy and t-closeness. Whereas the quality is worse than others even entropy l-diversity has less generalization time. For δ-disclosure privacy and t-closeness, the quality of anonymized data is improved with decreasing parameter value but with longer generalization time.

With the concern of minimized average size metric, Figure 2 shows that the quality of anonymized data is relatively stable and is acceptable with δ-disclosure privacy and t-closeness and with k-anonymity and entropy l-diversity respectively.

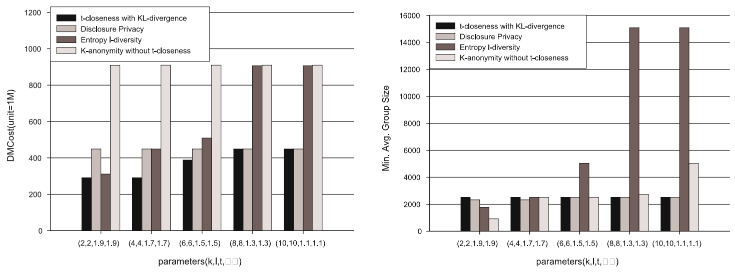


Fig. 2. Value of Discernibility with Different Configurations of Parameters

6 Related Work

When public data are published, how to preserve privacy of individual, which means that the identity and the sensitive information should be kept secret to others, has attracted attention from the research area of database management and statistics. The first privacy preservation model called k-anonymity was proposed in [16,18], where generalization and suppression techniques are developed to guarantee the k-anonymity of published data.

In the model of generalization and suppression, it is NP-hard to implement k-anonymity when $k > 2$ [14]. [14] designed an approximation algorithm with approximation ratio $O(k \log k)$ to minimized number of suppression. Recently the ratio is improved to $O(k)$ [2] and $O(\log k)$ [15]. A dynamic programming algorithm, Incognito [9] provides k-anonymity through full-domain recoding. Any metric can be implemented in this framework with generalization property and subset Property, such as t-closeness[11] and l-diversity.

[13] showed the possibility of privacy leakage. If the most or all of the records have the same sensitive value in an equivalence class, adversaries can infer individual sensitive information with high probability. So [13] solve it incompletely by introducing l-diversity. [7,19] proposed several approaches to l-diversity. Anatomy [19] splits the original table into two tables as a table with quasi-identifiers and a table with sensitive attributes and they can be joined according to group information. The work was expanded to transactional data [7]. But l-diversity can not prevent privacy leakage if there are many records leading to the same individual [20]. In l-diversity model, we restricts the knowledge owned by adversaries. While if adversaries have the information on the global distribution of sensitive value, they may obtain some privacy information. Consequently [11] proposed t-closeness model as an extension of l-diversity.

[10] provides algorithms for incorporating a class of target workloads, consisting of classification or regression models, as well as selection predicates, when generating an anonymous data recoding. m-invariance [21,22] addresses both record insertions and deletions in continuous data publish model, which ensures the intersection of sensitive values over all quasi-identifier group of generalized data does not reduce the set of sensitive values compared to each quasi-identifier group. [5] further relaxes the

privacy preservation scenario and assumes that quasi-identifier and sensitive values of a record owner could change in sequential releasing. [5] showed that knowledge of the mechanism or algorithm of anonymization for data publication can also lead to extra information that assists the adversary and jeopardizes individual privacy. In particular, all known mechanisms try to minimize information loss and such an attempt provides a loophole for attacks, which was called minimality attack.

Whether data set has been sanitized for privacy protection or not, the data quality should be a very important measure of published data. We could improve data quality by reducing the number of generalization and suppression, reducing the average number of quasi-identifier [13], and keeping marginal distribution [8]. [4] considered whether the quality of data generated with generalization and suppression of quasi-identifier is better than generated by simply separating quasi-identifiers from sensitive attributes and demonstrated experimentally that even for little privacy requirement, operations on data sets for privacy preservation could result in worse data quality. [4] also proposed semantic privacy which gives a semantic definition of sensitive attribute disclosure. It captures the gain in the adversary's knowledge due to his observations of the sanitized dataset. A methodology for measuring the tradeoff between the loss of privacy and the gain of utility is also proposed in [4]. While the most recent work [12] show that it is inappropriate to directly compare privacy with utility, after analyzing three fundamental characteristics of privacy and utility. Based on these characteristics, they also present a methodology for evaluating privacy-utility tradeoff.

7 Conclusion and Future Work

In this paper, we propose t-closeness with KL-divergence, and study its properties. We also reexamine the semantic privacy (namely δ-disclosure privacy), and discuss the relationship between t-closeness with KL-divergence and δ-disclosure privacy. We also implement an algorithm under the framework of Incognito algorithm to handle δ-disclosure privacy and t-closeness with KL-divergence. Furthermore, we measure the performance of generalization algorithm with different models and verify the quality of anonymized data.

An important question is how to design microdata sanitization algorithms that provide both privacy and utility [4,12]. We will study the relationship between these proposed privacy measures and data quality in the future, for example, the quality of patterns mined from generalized data set.

Acknowledgements

This work is supported by National Natural Science Foundation of China under grant No. 60903014, and National Hi-Tech 863 program under grant No. 2009AA01Z429. We like to thank Dao Tao for the help in implementations.

References

1. http://www.cs.cornell.edu/database/privacy/code/l-diversity/incognito-ldiversity.tgz
2. Aggarwal, G., Feder, T., Kenthapadi, K., Motwani, R., Panigrahy, R., Thomas, D., Zhu, A.: Approximation algorithms for k-anonymity. Journal of Privacy Technology (2005)
3. Bayardo, R., Agrawal, R.: Data privacy through optimal k-anonymization. In: ICDE 2005 (2005)
4. Brickell, J., Shmatikov, V.: The cost of privacy: Destruction of data-mining utility in anonymized data publishing. In: KDD 2008 (2008)
5. Bu, Y., Fu, A., Wong, R., Chen, L., Li, J.: Privacy preserving serial data publishing by role composition. In: VLDB 2008 (2008)
6. Cover, T., Thomas, J.: Elements of Information Theory. Wiley Interscience, Hoboken (1991)
7. Ghinita, G., Karras, P., Kalnis, P., Mamoulis, N.: Fast data anonymization with low information loss. In: VLDB 2007 (2007)
8. Kifer, D., Gehrke, J.: Injecting utility into anonymized datasets. In: SIGMOD (2006)
9. LeFevre, K., DeWitt, D., Ramakrishnan, R.: Incognito: Efficient full-domain k-anonymity. In: SIGMOD (2005)
10. LeFevre, K., DeWitt, D., Ramakrishnan, R.: Workload-aware anonymization. In: KDD 2006 (2006)
11. Li, N., Li, T., Venkatasubramanian, S.: t-closeness: Privacy beyond k-anonymity and l-diversity. In: ICDE 2007 (2007)
12. Li, T., Li, N.: On the tradeoff between privacy and utility in data publishing. In: KDD 2009 (2009)
13. Machanavajjhala, A., Gehrke, J., Kifer, D., Venkitasubramaniam, M.: l-diversity: Privacy beyond k-anonymity. In: ICDE 2006 (2006)
14. Meyerson, A., Williams, R.: On the complexity of optimal k-anonymity. In: PODS 2004 (2004)
15. Park, H., Shim, K.: Approximate algorithms for k-anonymity. In: SIGMOD 2007 (2007)
16. Samarati, P.: Protecting respondents identities in microdata release. IEEE Transactions on Knowledge and Data Engineering 13, 1010–1027 (2001)
17. Sweeney, L.: Achieving k-anonymity privacy protection using generalization and suppression. Int. Journal on Uncertainty, Fuzziness and Knowledge-based Systems 10 (2002)
18. Sweeney, L.: k-anonymity: a model for protecting privacy. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems 10 (2002)
19. Xiao, X., Tao, Y.: Anatomy: Simple and effective privacy preservation. In: VLDB 2006 (2006)
20. Xiao, X., Tao, Y.: Personalized privacy preservation. In: SIGMOD 2006 (2006)
21. Xiao, X., Tao, Y.: m-invariance: Towards privacy preserving re-publication of dynamic datasets. In: SIGMOD 2007 (2007)
22. Xiao, X., Tao, Y.: Dynamic anonymization: Accurate statistical analysis with privacy preservation. In: SIGMOD 2008 (2008)

Competitive Privacy: Secure Analysis on Integrated Sequence Data

Raymond Chi-Wing Wong[1] and Eric Lo[2]

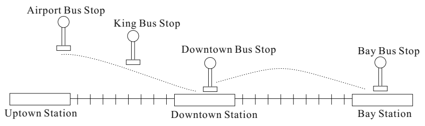
[1] Hong Kong University of Science and Technology
[2] Hong Kong Polytechnic University

Abstract. Sequence data analysis has been extensively studied in the literature. However, most previous work focuses on analyzing sequence data from a single source or party. In many applications such as logistics and network traffic analysis, sequence data comes from more than one source or party. When multiple autonomous organizations collaborate and integrate their sequence data to perform analysis, sensitive business information of individual parties can be easily leaked to the other parties. In this paper, we propose the notion of competitive privacy to model the privacy that should be protected when carrying out data analysis on integrated sequence data. We propose a query restriction algorithm that can reject malicious queries with low auditing overhead. Experimental results show that our proposed method guarantees the protection of competitive privacy with only a significantly small portion of queries being restricted.

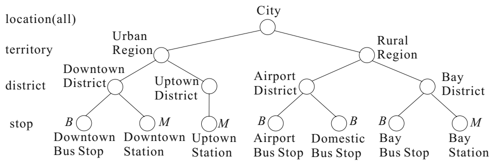
1 Introduction

Sequence data analysis has been studied extensively in the literature [4,6,2]. Most previous work focuses on analyzing sequence data collected from a single source or party. However, in applications such as logistics and network traffic analysis, some autonomous enterprises may want to integrate their sequence data in order to carry out joint data analysis. As a motivating example, consider the collaboration between a bus company B and a metro company M in a city that has implemented RFID-based electronic transportation payment systems (e.g., Washington DC's SmarTrip system). Each passenger has an RFID-card that can be used as a form of e-money for the fare of various transportations. Each transportation company participates in the e-transport network records a huge volume of passenger transactions every day. In this example, we can view each passenger traveling history as a data *sequence*. In Figure 1a, if a passenger traveled from "Airport Bus Stop" to "Downtown Bus Stop" by bus, transferred from "Downtown Bus Stop" to "Downtown Station" (via a transfer terminal in "Downtown") and finally traveled from "Downtown Station" to "Uptown Station" by metro, her traveling history can be represented as a data sequence ("Airport Bus Stop", "Downtown Bus Stop", "Downtown Station", "Uptown Station").

Suppose that B and M collaborate and offer discounts to passengers who traveled from the airport to uptown using a combination of bus and metro (transited at Downtown). One interesting query is to ask the number of passengers who traveled from "Airport Bus Stop" to "Uptown Station" via the transfer terminal in "Downtown". Furthermore, during data analysis, queries are often refined to different abstraction levels



(a) A transportation network



(b) A concept hierarchy

Fig. 1. Motivating Example

by the data analysts interactively. For example, if a concept hierarchy is defined for stations/stops like the one in Figure 1b, then the above query may be "rolled-up" by the user to ask for the number of passengers who traveled from "Airport District" to "Uptown District" via the transfer terminal in "Downtown". All these operations can be handled by sequence analytical systems such as [2] and [4] efficiently. One way to evaluate the analytical queries above is to have bus and metro to *integrate* their passenger data, which are originally *owned* and stored separately. Let D_M be the data owned by M and D_B be the data owned by B. D_M and D_B are integrated to form a new dataset D_I. In practice, however, both M and B actually do not want to disclose their data to their competitors, if possible. For instance, assume that there are two services operated by M and B separately from "Downtown District" to "Bay District". Specifically, M operates a service s_M from "Downtown Station" to "Bay Station" while B operates a service s_B from "Downtown Bus Stop" to "Bay Bus Stop". If passengers want to travel from "Downtown District" to "Bay District", they may choose either s_M or s_B. Thus, these two services s_M and s_B are *competitive*. Suppose that M poses a query and observes that the total number of passengers using service s_B (operated by B) is extremely large compared with its own service s_M. M may then offer discounts to customers who use its service s_M in order to attract the customers originally using service s_B. It is easy to see that, once there are discounts for service s_M, the original service s_B operated by B is definitely affected. Thus, the statistical information about the total number of passengers using service s_B can be regarded as the "competitive privacy" of party B and that should get protected during data analysis.

The objective of this paper is to support data analysis, in particular, OLAP, on an integrated sequence data set without compromising competitive privacy. Informally, let $Q(s_B, f)$ be an aggregate sequence query [2] that specifies an aggregate function f on all the sequences in the integrated data set D_I that match s_B and the data values in s_B are all *owned by* (or *originated from*) party B (formal definitions are given in Section 2), we say that there is a breach of *competitive privacy* if given a real number e, other parties (except B) can infer a value $\widetilde{f}$ such that $|\widetilde{f} - f(s_B)| \leq e$, where $f(s_B)$ denotes the answer of query Q. In this paper, we present a query restriction strategy to support data analysis on an integrated sequence data set without breaching the competitive privacy of any party. The strategy rejects a query Q if its answer can lead to a breach of competitive privacy. Existing query restriction strategies like [1], [5] and [3] focus on the protection of individual privacy or data privacy on a relational data set owned by a single party. The query restriction strategy in this paper focuses on the protection of competitive privacy on a sequence data set integrated from multiple autonomous parties.

2 Preliminary

We are given a set $\mathcal{V}$ of values that are associated with a concept hierarchy. Figure 1b shows a concept hierarchy. Nodes at the leaf level correspond to the values recorded in the data. A node N is said to be *ground* if it is at the leaf level or *non-ground* if it is not.

Each value in $\mathcal{V}$ corresponds to a node in the concept hierarchy. Without ambiguity, in the following, the terms "nodes" and "values" are used interchangeably. Each leaf node N is associated with an *ownership*, denoted by $N.T$. For example, since "Downtown Station" and "Uptown Station" are values originated from the metro company M's data, the *ownership* of these nodes are M. In Figure 1b, the ownership of a leaf node is next to itself. The non-leaf nodes such as "Downtown District" and "Urban Region" are used for data analysis and they do not have any ownership.

Suppose there are m datasets $\mathcal{D} = \{D_1, D_2, ..., D_m\}$ owned by m parties $P_1, P_2, ..., P_m$, respectively. Each data set contains a number of sequences. A *sequence* s is represented in the form of $(N_1, N_2, ..., N_k)$ where N_l is a ground or non-ground value in $\mathcal{V}$ for $l \in [1, k]$. We say that this sequence s is of *length* k. Implicitly, each value N_i in s is associated with a timestamp, denoted by $N_i.ts$, such that if $i < j$, $N_i.ts < N_j.ts$ for any $i, j \in [1, k]$. Each sequence s in dataset D_i is associated with a unique identifier, denoted by $s.id$ (e.g., the card id of an RFID card).

An integrated dataset D_I can be obtained by integrating the set of databases $\mathcal{D}$ according to the timestamp of the values. Specifically, let $\mathcal{C}$ be the set of sequence identifiers in $\mathcal{D}$. For each $x \in \mathcal{C}$, we obtain a set $\mathcal{S}$ of sequences from all datasets in $\mathcal{D}$ such that $\mathcal{S} = \{s \in \mathcal{D} | s.id = x\}$. Let $\mathcal{N}$ be the multi-set containing the values of all sequences in $\mathcal{S}$. We generate a new sequence s' of length $|\mathcal{N}|$ in the form of $(N_1, N_2, ...N_{|\mathcal{N}|})$, such that if $i < j$, $N_i.ts < N_j.ts$ for any $i, j \in [1, |\mathcal{N}|]$. The new sequence s' will be inserted into the integrated dataset D_I.

In this paper, we focus on aggregate sequence queries [2]. If such a query $Q(s, f)$, or simply Q if the context is clear, is posed on a sequence data set D_I, it applies an aggregate function f on all the sequences in D_I that MATCH s, and returns a scalar value, denoted as $f(s)$, to a user. We remark that MATCH can be any pattern matching function. For example, it can be a sub-string matching function (i.e., if s is a sub-string of a sequence s' in D_I, MATCH returns `true`) or a sub-sequence matching function (i.e., if s is a sub-sequence of s' in D_I, MATCH returns `true`). The technique in this paper is applicable to all kinds of aggregate sequence queries discussed in [2]. Nonetheless, for the sake of illustration, the following discussion mainly centers around the `COUNT` aggregation function and the sub-string matching function. Therefore in the following, unless stated otherwise, we assume a query $Q(s, f)$ on D_I means that for each sequence in D_I which contains s as substring (despite the number of occurrences of s in a sequence) increments the value of $f(s)$ by one. A query $Q(s, f)$ is of *length* k if the length of sequence s specified in Q is k. We denote that by $|Q|$. As the data is actually integrated from multiple parties, we assume that all queries are of length at least two.

As in traditional OLAP environments, users may interactively refine their queries. For instance, a user (of party P_i) may first issue a query to obtain the number of customers who traveled from "Airport Bus Stop" to "Uptown Bus Stop" and then refine her query Q by a "pattern roll-up" operation [2] in order to obtain the number of passengers who traveled from "Airport District" to "Uptown District". These concepts can be formalized as follows.

Assume party P_i issues her first query Q_1 at time $t = 1$, second query Q_2 at time $t = 2$ and so on. Let $\mathcal{K}_{P_i}(t)$ be the knowledge of party P_i at time t. Thus, the initial knowledge of party P_i before she issues any query on the integrated data set D_I, denoted as $\mathcal{K}_{P_i}(1^-)$, contains all aggregate values $f(s)$ for all s in D_i. Further, let t^- and t^+ be the time *immediately* before and after time t, i.e., t^- is any time between time $t-1$ and time t, and t^+ be any time between time t and time $t+1$. Thus, for any $t > 1$, $\mathcal{K}_{P_i}(t^-) = \mathcal{K}_{P_i}((t-1)^+)$. After party P_i issues a query Q_t at time t, if Q_t is not rejected, P_i's knowledge $\mathcal{K}_{P_i}(t^+)$ is immediately updated to $\mathcal{K}_{P_i}(t^-) \cup \{f(s)\}$ (where $f(s)$ is the answer of Q_t); otherwise, P_i's knowledge $\mathcal{K}_{P_i}(t^+)$ remains as $\mathcal{K}_{P_i}(t^-)$.

3 Competitive Privacy

In this section, we present the concept of *competitve privacy*, which is the key element that we should consider when supporting data analysis on a sequence data set that is integrated from multiple autonomous parties. We assume that the integrated data set D_I is located at trusted party T and the involved parties send their queries to T. The RFID transport payment company can be regarded as the trusted parties for the motivating example. Although a trusted party is involved, the privacy issue has not been resolved yet. Specifically, in the following, we are going that formalize the concept of competitive privacy and show that if a party P can pose any queries without any restriction, that will breach competitive privacy of some party. Let us begin with the definition of *conflicting node set*. Given a ground node N, the *conflicting node set* of N, denoted by $C(N)$, is a set of ground nodes such that for each node $N' \in C(N)$, $N'.T \neq N.T$. The conflicting node set is *specified* or *given* by the multiple autonomous parties and the trusted party.

In our running example, if the metro company offers a service from "Downtown Station" to "Bay Station" and the bus company offers a service from "Downtown Bus Stop" to "Bay Bus Stop", then the manager of the metro may specify that the conflicting node set of "Downtown Station" as {"Downtown Bus Stop"}. Similarly, the manager is likely to specify that the conflicting node set of "Bay Station" as {"Bay Bus Stop"}. We remark that we assume the notion of conflicting node set is symmetric in this paper. As a result, if C("Downtown Station")={"Downtown Bus Stop"}, then C("Downtown Bus Stop'")={"Downtown Station"}.

Given a sequence s_i in D_i in the form of (N_p, N_q) and another sequence s_j of D_j in the form of (N_r, N_s), s_j is a *competitive sequence* of s_i, if $N_r \in C(N_p)$ and $N_s \in C(N_q)$. The set of competitive sequences of s_i is denoted by $C(s_i)$. For example, let s_M=("Downtown Station", "Bay Station") in D_M and s_B=("Downtown Bus Stop", "Bay Bus Stop") in D_B. Following the example above, as "Downtown Station" $\in C$("Downtown Bus Stop") and "Bay Station" $\in C$("Bay Bus Stop"), s_M is a competitive sequence of s_B (and vice versa because of the symmetric property). Note that instead of asking the managers (which are the target users of OLAP systems) to specify the (query) views that needed to be protected as in [3], we intentionally introduce the notion of conflicting node such that it is more non-technical people friendly. For example, rather than directly specifying s_M and s_B as competitive sequences, it would be more intuitive for those business people, say, the operation manager of party M to directly specify "Downtown Station" and "Downtown Bus Stop" as "conflicting". Nonetheless, of course, it is also possible for users to directly specify competitive sequences as well. Now, we can define the competitive privacy of a party P as follows.

Definition 1 (Competitive Privacy). *The competitive privacy $\mathcal{CP}$ of a party P_i is defined as the statistical information of all competitive sequences in D_i, i.e., $\mathcal{CP} = \{f(s) | \forall s \in D_i$ and there exists $s' \in D_j$ such that $j \neq i$ and $s' \in C(s)\}$.* □

Similar to what we discussed in Section 1, the statistical information of each competitive sequences, namely $\mathcal{CP}$, are regarded as the "competitive privacy" of a party. Thus, without any query restriction, a party can *directly* obtain the statistical information of the competitive sequence of the other party easily and can do something bad to the other party.

We now show that party M can infer a value for $\texttt{COUNT}(s_B)$, through *query inferences*, even though it only obtains the statistical information other than the value of $\texttt{COUNT}(s_B)$.

Example 1 (Query Inferences). In our motivating example, both the bus (party B) and the metro (party M) offer services from Downtown district to Bay district. Assume that each of the parties provide only one service from Downtown district to Bay district.

Initially, $\mathcal{K}_M(1^-) = \{\texttt{COUNT}(s_M)\}$. Suppose at time 1, M issues a query Q_1 $(\hat{s}, \texttt{COUNT})$, where $\hat{s}$=("Downtown District", "Bay District") (where the concept hierarchy is the one in Figure 1b). Without any query restriction, Q_1 can be posed on D_I and thus the knowledge of M can be updated to $\mathcal{K}_M(1^+) = \{\texttt{COUNT}(s_M), \texttt{COUNT}(\hat{s})\}$. Assume $\texttt{COUNT}(s_M) = 10,000$ and $\texttt{COUNT}(\hat{s}) = 90,000$, Party M can infer a value for $f(s_B)$ as $\texttt{COUNT}(\hat{s}) - \texttt{COUNT}(s_M) = 80,000$ □

Definition 2 (Competitive Privacy Breach). *Given two competitive sequences s_i and s_j obtained from D_i and D_j respectively. At time t, we say that there is a* competitive privacy breach *with respect to party P_j by party P_i if, given a real number e, P_i can infer a value $\widetilde{f}(s_j|\mathcal{K}_{P_i}(t^-))$ for $f(s_j)$ such that $|\widetilde{f}(s_j|\mathcal{K}_{P_i}(t^-)) - f(s_j)| \leq e$ based on knowledge $\mathcal{K}_{P_i}(t^-)$.* □

4 Query Restriction

In this section, we will give a high-level description of the proposed algorithm called CCF (conservative competition-free) to avoid any competitive privacy breach. Details of this algorithm can be found in [7]. Intuitively, we reject some queries which may breach competitive privacy. Consider a query Q which has sequence s. We reject query Q if one of the following two conditions holds. *Condition 1:* There exists a competitive sequence which is a sub-sequence of s. *Condition 2:* There exists a *generalized version* of competitive sequence which is a sub-sequence of s. We say that sequence $s_i = (N_1, N_2, ..., N_l)$ is a *generalized version* of another sequence $s_j = (M_1, M_2, ..., M_l)$ if N_x is equal to M_x or is an ancestor node of M_x (in the concept hierarchy) for all $x \in [1, l]$. In [7], we prove that our query restriction algorithm can avoid any competitive privacy breach.

5 Empirical Study

We have conducted extensive experiments on a Pentium IV 2.4GHz PC with 1GB memory, on a Linux platform. The programs were implemented in C++. We evaluated our algorithm, CCF, on both synthetic and real datasets, in terms of four measurements: (1) *average auditing time*, (2) *ratio of restricted queries*, and (3) *storage*. The average auditing time corresponds to the average time to check whether a query is rejected by our proposed algorithm CCF. The ratio of restricted queries is equal to the total number of restricted queries by CCF over the total number of issued random queries. The storage corresponds to the memory usage to hold all competitive sequences of all parties, namely $\mathcal{CS}$. All experiments were conducted 100 times and we took the average for the results. In our experiments, we generate 10,000 batches of queries. Each batch contains 20 queries. We randomly generate a query Q_1 with sequence $s = (N_1, N_2, .., N_{|Q_1|})$ as follows. For each N_i where $i \in [1, |Q_1|]$, we randomly select a value in the concept hierarchy. Then, we refine query Q_1 and generate another new query Q_2. We adopt the refinement operations from [2]: Append, De-tail, Pre-pend, De-head, Pattern-roll-up and Pattern-drill-down. We randomly select one of the operations and generate query Q_2. Similarly, we repeatedly generate query Q_i from Q_{i-1} until $i = 20$.

Synthetic Dataset: The synthetic dataset is generated by a dataset generator. This generator creates sequences with 4 parameters, namely n, p, c and l, where n is the total number of (integrated) sequences, p is the total number of parties, c is the percentage of ground competitive sequences in each party's dataset, and l is the average length of the data sequences. The data sequence is generated as the same way as [2] and we randomly assign $c\%$ of sequences as competitive. We generate a generalization hierarchy of height 3. We partition the ground nodes into different groups such that different ground nodes, say v and v', where $v \in C(v')$ (or $v' \in C(v)$) forms the same group. For each group of ground nodes, we create an internal node N. Finally, we create a root node N' such that the parent of all internal nodes constructed is N'. Thus, the final hierarchy has height equal to 3. The default values of n (num. of tuples), p (num. of parties), c (the ratio of ground competitive sequences) and l (average sequence length) are 500K, 4, 0.05, and 20, respectively. In the experiments, we study the effect of the total number of tuples and the length of the query.

In Figure 2(a), the average audit time remains nearly unchanged when the dataset size changes. The auditing time of our proposed algorithm mainly depends on the size of $\mathcal{CS}$ (Details can be found in [7]). Since the size of $\mathcal{CS}$ is fixed (Figure 2(c)), the change in the dataset size does not affect the auditing time too much. Besides, we can observe that the average auditing time increases with $|Q|$, the length of the query. Figure 2(b) shows that the number of restricted queries is nearly the same with different dataset size. Similarly, since the percentage of competitive sequences remains unchanged when the dataset size changes, the ratio of restricted queries also remains unchanged. When $|Q|$ increases, it is trivial that the ratio increases. Figure 2(c) shows the storage of algorithm CCF keeps unchanged when the dataset size increases. This is because the storage for set $\mathcal{CS}$ is independent of the dataset size.

Real Dataset: The real dataset is obtained from a local transportation organization called MTR in Hong Kong. It consists of passenger transactions of 5-working-day, all

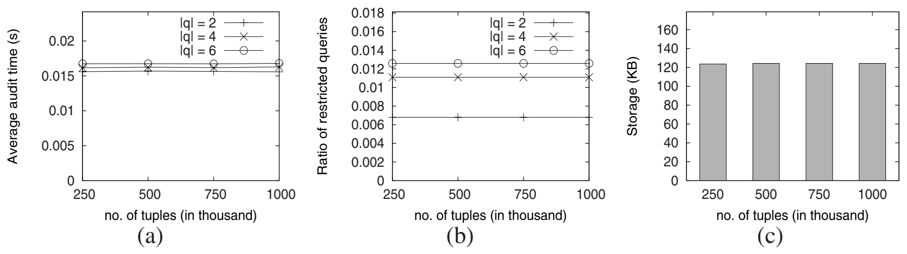


Fig. 2. Effect of the total number of tuples

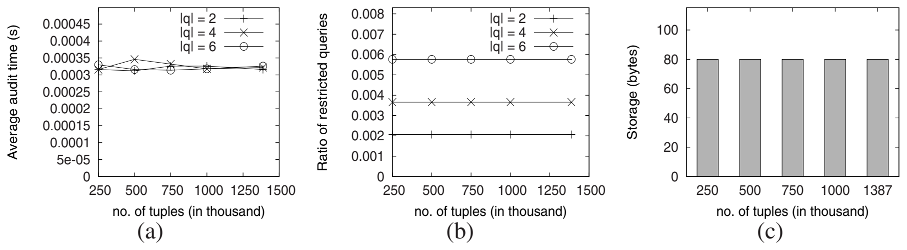


Fig. 3. Effect of the total number of tuples (real dataset)

recorded by an RFID-based electronic payment system. The passenger transactions are consolidated from 4 different in-city railway lanes. Each lane corresponds to a party. There are 63 stations in total and 6 of them are transfer terminals. In particular, 5 transfer terminals allow passengers to switch to another lane, and 1 transfer terminal is a hub that allow passengers to switch to two other lanes. An example record is like (N_1, N_2, N_3, N_4), which denotes that there was a passenger entered the railway network at station N_1, got off at station N_2, transferred to another lane at station N_3 and finally left the railway network at station N_4. All pairs of transfer terminals as defined as conflicting. That is, in this example, $C(N_2) = \{N_3\}$. All together we have 1,387,831 sequence records. The average sequence length of a record is 2.9 stations. According to the locations of stations, we divide the stations into different regions such that there are 63 leaf values and 31 non-leaf values in the concept hierarchy of height 4.

We carry out experiments that are similar to the results for synthetic datasets. Figure 3 shows that the experimental results are similar to those on the synthetic data. In order to conduct the experiments with the variation of the number of tuples, we randomly sample a subset of tuples. The average audit time, the ratio of restricted queries and storage remain nearly unchanged when we vary the total number of tuples.

6 Conclusion

Most previous works focus on privacy issues over data from a single source. This paper formulates a problem called competitive privacy which considers privacy issues when sequence data is integrated from more than one source. Our proposed algorithm CCF rejects queries efficiently and guarantees no competitive privacy breach. In all experiments, the auditing step can be achieved within 0.04s and the ratio of the total number of restricted queries over the total number of queries is also small (within 0.15).

Acknowledgements. The research of Raymond Chi-Wing Wong is supported by HKRGC GRF 621309 and Direct Allocation Grant DAG08/09.EG01. The research of Eric Lo is supported by Hong Kong Research Grants Council GRF grant PolyU 525009E and PolyU internal grant 1-ZV5R.

References

1. Dobkin, D.P., et al.: Secure databases: Protection against user influence. ACM Trans. Database Syst. 4(1), 97–106 (1979)
2. Lo, E., et al.: OLAP on sequence data. In: SIGMOD Conference (2008)
3. Miklau, G., et al.: A formal analysis of information disclosure in data exchange. J. Comput. Syst. Sci. 73(3), 507–534 (2007)
4. Seshadri, P., et al.: Sequence query processing. In: SIGMOD (1994)
5. Motwani, R., et al.: Auditing SQL Queries. In: ICDE (2008)
6. Ramakrishnan, R., et al.: SRQL: Sorted Relational Query Language. In: Rafanelli, M., Svensson, P., Klensin, J.C. (eds.) SSDBM 1988. LNCS, vol. 339. Springer, Heidelberg (1989)
7. Wong, R.C.-W., Lo, E.: Analyzing integrated sequence data in a competitive environment (2009), http://www.cse.ust.hk/~raywong/paper/analyzingIntegratedSequence-tech.pdf

Privacy-Preserving Publishing Data with Full Functional Dependencies

Hui (Wendy) Wang and Ruilin Liu

Stevens Institute of Technology
Hoboken, NJ, USA
{hwang,rliu3}@cs.stevens.edu

Abstract. We study the privacy threat by publishing data that contains full functional dependencies (FFDs). We show that the cross-attribute correlations by FFDs can bring potential vulnerability to privacy. Unfortunately, none of the existing anonymization principles can effectively prevent against the FFD-based privacy attack. In this paper, we formalize the FFD-based privacy attack, define the privacy model (d, ℓ)*-inference* to combat the FFD-based attack, and design robust anonymization algorithm that achieves (d, ℓ)*-inference*. The efficiency and effectiveness of our approach are demonstrated by the empirical study.

1 Introduction

How to publish data that contains sensitive information of individuals has received considerable attention in recent years. It has been shown that simply removing explicit identifiers (IDs), e.g., name and SSN, from the released data is insufficient to protect privacy [9]. The existence of a set of non-ID attributes (called *quasi-identifiers*($\mathcal{QI}$)), e.g., the combination of zipcode, gender and date of birth, that can uniquely identify individuals, can be joined with information obtained from diverse external sources (e.g., public voting registration data) to re-identify the individuals in the released data. This is called the *record linkage attack*.

Table 1. Anonymized microdata before&after FD inference

ID	QI		Sensitive	
Name	Sex	Zip	Phone	Disease
Alice	F	07921	1111111	Ovarian cancer
Bob	M	07920	2222222	Bronchitis
Calvin	M	07902	3333333	Diabetes
Doris	F	07901	1000001	Ovarian cancer
Eve	F	07902	3333333	Bronchitis
Flora	F	07903	2000001	Pneumonia

(a) The Original Microdata

QI		Sensitive	
Sex	Zip	Phone	Disease
*	079**	1111111	Ovarian cancer
*	079**	2222222	Bronchitis
*	079**	3333333	Diabetes
F	0790*	1000001	Ovarian cancer
F	0790*	3333333	Bronchitis
F	0790*	2000001	Pneumonia

(b) The 3-diversity table

QI		Sensitive	
Sex	Zip	Phone	Disease
*	079**	1111111	Ovarian cancer
*	079**	2222222	Bronchitis
*	0790*	3333333	Diabetes
F	0790*	1000001	Ovarian cancer
F	0790*	3333333	Bronchitis
F	0790*	2000001	Pneumonia

(c) The table after FD inference

Various privacy principles have been proposed recently to defend against the record linkage attack (e.g., k-anonymity [9,7] and ℓ-diversity [4]). However, by applying FDs on the released data that has met some aforementioned privacy principles, the attacker may be able to breach privacy. For example, assume the microdata in Table 1 (a) contains the functional dependency $F : Phone \rightarrow Zip$, which states that any two same phone numbers must correspond to the same zipcode. Assume that the attacker possesses the knowledge of F. Then by applying F on the 3-diversity table (i.e., every group that has the same QI-values contains at least three unique sensitive values) in Table 1 (b) , he/she can modify the zipcode value of the third tuple from "079**" to "0790*", since the second group contains the phone number "3333333" with zipcode "0790*". The anonymized table after the FD-based inference is shown in Table 1 (c). The third tuple only satisfies 1-diversity privacy guarantee.

The example shows that using FDs as adversary knowledge may bring privacy breach. Given the fact that it is not difficult for the attacker to obtain these functional dependencies from either the common sense or other sources, it is necessary to develop robust privacy criterion of publishing data when functional dependencies are available and are used as part of the adversary knowledge.

In this paper, we focus on full functional dependencies (FFDs). We have the following contributions.

- Formally define the FFD-based attack.
- Define the (d, ℓ)-*inference* model to defend against the FFD-based attack.
- Propose novel *grouping stradegy* to archive (d, ℓ)-*inference* .
- Design an efficient anonymization algorithm to produce anonymized microdata.
- Demonstrate the efficacy of our algorithm by an extensive set of experiments.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 introduces the preliminaries. Section 4 defines our privacy model. Section 5 proposes the intersection-grouping strategy. Section 6 presents the details of our anonymization algorithm. Section 7 presents the experimental results. We conclude the paper in Section 8.

2 Related Work

Privacy-preserving data publishing has received considerable attention in recent years. The k-anonymity model requires that in the published data, every individual is related with no less than *k* tuples [9,7]. The ℓ-diversity [4] model further requires that every QI-group contains at least ℓ sensitive values that have roughly the same frequency. Other variants of k-anonymity and ℓ-diversity, e.g., t-closeness [3] and (α, k)-anonymity [11], (c, k)-safety [5], are defined to address different privacy requirements. Unfortunately, none of them can defend against our defined FFD-based privacy attack.

Martin et al. [5] and Rastogi et al. [6] are the first to consider the adversary who knows arbitrary correlations between tuples. They show that if such correlations are available, then there exists privacy leakage on the published dataset that is of "meaningful" utility. Both of them focus on tuple correlations but do not consider FDs. Kifer [2] shows that the attacker may induce correlations from the sanitized dataset; such inferred correlations can enable potential vulnerabilities on the sanitized dataset. However, [2] does not provide any solution to defend against the FD-based attack. Tao et al. [10] study the correlation hiding problem of data publishing. They use i-masking operation to ensure the attributes of correlation which needs to hide are independent. Their work presents a different goal from ours.

3 Preliminaries

Functional Dependency. Given two attributes X and Y, a database instance D satisfies the functional dependency FD $F : X \rightarrow Y$ if for every two tuples $t_1, t_2 \in D$, if $t_1.X$ = $t_2.X$, then $t_1.Y$ = $t_2.Y$. We call X the *determinant* attributes and Y the *dependent* attributes, and their values the *determinant* values and *dependent* values. In this paper, *we only consider FFDs*, i.e., the FDs $X \rightarrow Y$ that hold for all values of X and Y.

Anonymization Framework. There are three types of attributes in the microdata: identifiers (ID), quasi-identifiers $\mathcal{QI}$, whose combination can play as the key and uniquely identify any individual, and sensitive attributes $\mathcal{S}$. There may exist multiple sensitive attributes. We can consider them as a super attribute. For simplicity, we consider single sensitive attribute in the remaining of the paper.

In this paper, we consider *generalization* [8,9], a popular anonymization technique. By generalization, numerical QI-values are recoded as an interval (e.g., age 20 is recorded as $[20, 40]$), while the categorical QI-values are replaced with higher level domain values in the taxonomy tree (e.g., city "Hoboken" is replaced with "New Jersey"). In this paper, we only consider numerical QI-values. The purpose of generalization is to hide each individual tuple into a group, which is called the *QI-group* where all tuples inside have the same QI-values after generalization.

It is important to measure the incurred information loss by generalization. In this paper, we consider the generalized loss metric [1], which measures the information loss as a ratio. The definition of the metric is given below.

Definition 1 (Information Loss). *For a data value v, if it is suppressed from the released dataset, its information loss equals $IL_v = 1$. Otherwise if it is generalized to an interval $[L_i, U_i]$, let Min and Max be the minimum and maximum values of the attribute A. The information loss of v is: $IL_v = (U_i - L_i)/(Max - Min)$.*
The information loss IL_t of a tuple t is defined as $IL_t = (\sum_{v_i \in t} IL_{v_i})/n$, where n is the number of non-ID attributes.
The information loss of the microdata D is defined as $IL_D = (\sum_{t \in D} IL_t)/|D|$. □

4 Privacy Model

In this section, first, we define the FFD-based attack. Then we formally define the (d, ℓ)-*inference* model that combats the FFD-based attack.

4.1 FFD-Based Attack

To analyze the impact of FFDs to privacy, we consider a popular privacy model, ℓ-diversity [4]. It requires that each QI-group must consist of at least ℓ "well-represented" distinct values that are of close frequency. We define d-closeness to address the requirement of close frequency. The definition of d-closeness is a simplified version of "well-represented" in [3].

Definition 2 (d-closeness). *Given two sensitive values s_1 and s_2, let f_1 and f_2 be their frequency, then s_1 and s_2 are considered as d*-close *if $|f_1 - f_2| \leq d$. Given a QI-group*

G that consists of a set of distinct sensitive values, G is d-close if $\forall$ sensitive values $s_i, s_j \in G$, s_i and s_j are d-close. □

Based on the definition of d-closeness, we give a simplified version of ℓ-diversity.

Definition 3 (ℓ-diversity). *Given a microdata D, let D^* be its anonymized version. Then D^* is* ℓ-diverse *if $\forall$ sensitive attribute S of D, each QI-group $G \in D^*$ consists of at least ℓ distinct sensitive values on S that are d-close.* □

Next, we explain the details of FFD-based attack. In the anonymized dataset, it is possible that different QI-groups share the same sensitive or QI-values. This enables the possibility of the FFD-based attack. Intuitively, assume both QI-groups G_1 and G_2 include the value a, where a is a determinant value of the FD $F : \mathcal{A} \rightarrow \mathcal{B}$. Let b be the corresponding dependent value of a in the original microdata. Even though b can be generalized to different values b_1^* and b_2^* in G_1 and G_2, due to the presence of FFDs, the attacker still can infer that b_1^* and b_2^* must correspond to the same original value. Thus he/she can "intersect" b_1^* and b_2^*. It is such intersection that enables the FFD-based attack. In the next, first, we formally define $b_1^* \cap b_2^*$, the *intersection* operation on generalized values. To distinguish from the conventional intersection operation $\cap$, we use $\cap^*$ to denote the intersection operation on generalized data values.

Definition 4 (Intersection of Generalized values). *Given two generalized values b_1^* and b_2^*, which are two intervals $[l_1, u_1]$ and $[l_2, u_2]$, if these two intervals overlap, then $b_1^* \cap^* b_2^*$ = $[max(l_1, l_2), min(u_1, u_2)]$, otherwise $b_1^* \cap^* b_2^* = NULL$.* □

For example, given two generalized *Age* values $b_1^* = [20, 40]$ and $b_2^* = [30, 50]$, $b_1^* \cap^* b_2^* = [30, 40]$.

As aforementioned, due to FFDs, the attacker can conclude that generalized dependent values b_1^* and b_2^* in the tuples in G_1 and G_2 that contain the same determinant values indeed correspond to the same original value. Then he/she can replace both b_1^* and b_2^* in these tuples with $b_1^* \cap^* b_2^*$. Such replacement separates the tuples in QI-group G_1 (G_2, resp.) into two sets, the one of the values b_1^* (b_2^*, resp.), and the one of the values $b_1^* \cap^* b_2^*$. We formally define these two sets below. To distinguish from the conventional set intersection/difference($\cap/-$) operation, we use $G_1 \cap_F G_2$ and $G_1 -_F G_2$ to denote the set intersection/difference operations of G_1 and G_2 based on the reasoning of FFD F. We use $t[\mathcal{A}]$ to denote the values of attributes $\mathcal{A}$ of the tuple t.

Definition 5 (FFD-based Intersection/Difference of QI-groups). *Given the FFD $F : \mathcal{A} \rightarrow \mathcal{B}$ of the microdata D and two QI-groups G_1, G_2, let $G_{12} = \{t | t \in G_1, \exists t' \in G_2$ s.t. $t[\mathcal{A}]$=$t'[\mathcal{A}]\}$. Then $G_1 \cap_F G_2$ is a set of tuples J s.t. $\forall t \in G_{12}$, $\exists t' \in J$ s.t.*
(1) $\forall$ attribute $A \in \mathcal{A}$, $t'[A] = t[A]$,
(2) $\forall$ attribute $B \in \mathcal{B}$,

$$t'[B] = \begin{cases} t[B] & \textit{if } t[B] \textit{ is not generalized} \\ G_1[B] \cap^* G_2[B] & \textit{if } t[B] \textit{ is generalized} \end{cases}$$

Furthermore, $G_1 -_F G_2 = G_1 - G_{12}$. □

For example, for the two QI-groups G_1 and G_2 in Table 1 (a) with FFD $F : Phone \rightarrow Zip$, $G_1 \cap_F G_2$ ={*, 0790*, 3333333, Diabetes}, and $G_1 -_F G_2$ returns the first two tuples in G_1.

Now we are ready to define *FFD-based privacy attack*.

Definition 6 (FFD-based Privacy Attack). *Given a microdata D, let D^* be its generalized version that satisfies ℓ-diversity. Then the full functional dependency $F : A \rightarrow B$ ($A, B \subseteq \mathcal{QI} \cup \mathcal{S}$) enables the* FFD-based privacy attack *if there exist two QI-groups $G_1, G_2 \in D^*$ such that at least one of followings is non-empty and does not satisfy ℓ-diversity: (1) $G_1 \cap_F G_2$, (2) $G_2 \cap_F G_1$, (3) $G_1 -_F G_2$, and (4) $G_2 -_F G_1$. Otherwise, we say D^* is* safe at the presence of F. □

We have shown an example of FFD-based privacy attack in Section 1.

Although FFDs may threat privacy, not all FFDs can enable the FFD-based attack. Based on this, we define *safe* and *unsafe* FFDs.

Definition 7 (Safe/Unsafe FFDs). *A functional dependency F is* safe *if for any microdata D that satisfies F, all of its possible generalized versions D^* are safe at the presence of F. Otherwise, we say F is* unsafe. □

Based on the definition, we distinguish the "safe" FFDs from the "unsafe" ones. We have:

Theorem 1 ((Un)safe FFDs). *Given the microdata D that contains the QI attributes $\mathcal{QI}$ and sensitive attributes $\mathcal{S}$, let $F : \mathcal{A} \rightarrow \mathcal{B}$ ($\mathcal{A}, \mathcal{B} \subseteq \mathcal{QI} \cup \mathcal{S}$) be one of its FFDs. then F is* safe *iff $\mathcal{A} \subseteq \mathcal{QI}$. Otherwise, F is* unsafe. □

Due to space limit, we skip the proof. Next, we define the (d, ℓ)*-inference* model to defend against the privacy attack by unsafe FDs.

Definition 8 ((d, ℓ)-inference). *Given microdata D, let D^* be its anonymized version that consists of the QI-groups $\mathcal{G}\{G_1, \ldots, G_n\}$. Let S_i be the set of distinct sensitive values of the QI-group G_i ($1 \leq i \leq n$). Then D^* satisfies* (d, ℓ)-inference *if*
*(1) d-**close**: $\forall G \in \mathcal{G}$, all sensitive value sets in G are d-close,*
*(2) ℓ-**overlapping**: $\forall G_i, \ldots, G_j \in \mathcal{G}$, if $|S_i \cap \cdots \cap S_j| \neq 0$, then $|S_i \cap \cdots \cap S_j| \geq \ell$, i.e., there are at least ℓ shared distinct values in $S_i \cap \cdots \cap S_j$,*
*(3) ℓ-**non-overlapping**: $\forall G_i, G_j \in \mathcal{G}$, $|S_i - S_j| \geq \ell$, i.e., there are at least ℓ non-overlapping distinct values in $S_i - S_j$.* □

Both ℓ-overlapping and ℓ-non-overlapping conditions consider the worst case, i.e., the "smallest" results of intersection and set difference. Thus ℓ-overlapping considers interactions of multiple QI-groups (maybe more than 2), while ℓ-non-overlapping considers the difference of two QI-groups. Note that the (d, ℓ)-inference model is not based on the reasoning of the generalized datasets anymore. Instead, it only reasons on the sensitive values and can be applied on the original microdata directly.

5 Intersection-Grouping (IG)

The key to achieve (d, ℓ)*-inference* is to appropriately group the sensitive values so that all these groups meet the three conditions of (d, ℓ)*-inference*. To address this, we propose the *intersection-grouping* strategy. It puts the sensitive values into groups that intersect (but not contain) in a chain. To avoid considering intersection of arbitrary number of groups, we do not allow the intersection of more than two groups. Furthermore, we require that all overlapped groups construct a chain, i.e., given a set of groups $G_1, \ldots, G_m$, G_i only intersects with G_{i+1} and G_{i-1}, but not the others.

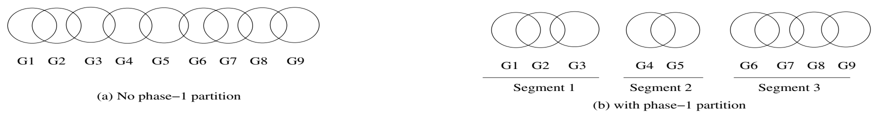


Fig. 1. Without & with Phase-1 Partition

The intersection-grouping approach consists of two steps: (1) *bucket construction* to construct d-close, ℓ-diverse, disjoint buckets and (2)*intersected groups construction* to construct intersected groups from the buckets. Due to the space limit, we omit the details. Instead, we use an example to show how our grouping strategy works.

Example 1. Given the sensitive values $\{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$ of frequency (1, 7, 9, 10, 26, 30, 40, 45, 50), assume $d = 3$ and $\ell = 2$. By Step 1, we construct the buckets $B_1\{s_2, s_3, s_4\}$ of frequency (7, 9, 10), $B_2\{s_5, s_6\}$ of frequency (26, 29), and $B_3\{s_7, s_8, s_9\}$ of frequency (40, 43, 43). There are 1 tuple containing s_1, 1 tuple containing s_6, 2 tuples containing s_8, and 7 tuples containing s_9 that will be removed. In Step 2, we construct the following groups: G_1 $(s_2, s_3, s_4, s_5, s_6)$ of frequency (7, 9, 10, 10, 10), G_2: $(s_5, s_6, s_7, s_8, s_9)$ of frequency (16, 19, 19, 19, 19), and G_3: (s_7, s_8, s_9) of frequency (21, 24, 24). There are 1+ 1 + 2 + 7 = 11 tuples in total that are removed. □

6 Anonymization Algorithm

In this section, we explain the details of the algorithm that constructs the QI-groups for anonymization. The purpose of the QI-group is two-fold: (1) achieve (d, ℓ)-*inference* privacy guarantee, and (2) minimize information loss, including both by tuple suppression and by generalization. Our anonymization algorithm has two phases, phase-1 that minimizes the information loss by tuple suppression, and phase-2 that minimizes the information loss by tuple generalization. In particular, *phase-1* partitions the tuples by their sensitive values, so that each group satisfies (d, ℓ)-*inference*, while the phase-2 constructs QI-groups of minimized information loss by generalization from the constructed phase-1 partitions.

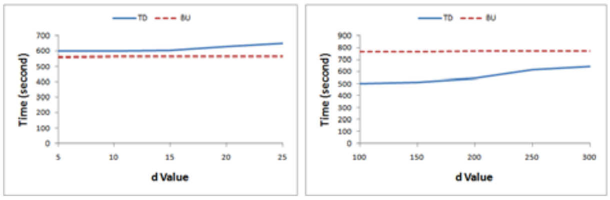
Phase-1 Partition. We split the sensitive values into smaller disjoint segments, and apply IG on these segments. Figure 1 illustrates the effect of the partition. We prove that finding an optimal partition is a NP-hard problem, and propose two heuristic partitioning approaches, namely *top-down* and *bottom-up*. Both heuristics are designed in a greedy fashion. Due to the limit space, we omit the details.

Phase-2 QI-Group Construction. To further reduce the information loss, we split each partition into smaller groups of same sensitive values. Since the partition P satisfies (d, ℓ)-inference, it is straightforward all QI-groups from P must satisfy (d, ℓ)-inference.

7 Experiments

We have done an extensive set of experiments to evaluate both the effectiveness and efficiency of our anonymization algorithm. Due to space limit, in this section, we briefly describe our experiment design and results.

We use a workstation machine of 2.4GHz Intel core and 3GB of RAM. We implement the algorithms in C++. We use both synthetic datasets in the experiments. To measure the impact of frequency distribution of sensitive values to anonymization, we generated two types of synthetic datasets, the one of sensitive values of close-to-uniform, and the one of sensitive values of skewed distribution. We call these two datasets the U*-dis* dataset and S*-dis* dataset. We designed the functional dependency *salary-class* $\rightarrow$ *work-class* for the synthetic dataset.



(a) U-dis set. $\ell = 5$ (b) S-dis set. $\ell = 5$

Fig. 2. Time Performance Comparison (TD: Top-down, BU: Bottom-up)

Time Performance. First, we focus on the top-down and bottom-up approaches and measure the impact of the d value to the performance of anonymization. Figure 2 (a) & (b) show that for both U-dis and S-dis datasets, the performance of the bottom-up approach is relatively stable with changing d value. However, the time performance of top-down gets worse with larger d, as larger d results in more partitions. We also measure the performance of the phase-1 partition of both top-down and bottom-up approaches. The experiment result shows that it is around 0.016 second for both U-dis and S-dis datasets. Thus it is negligible compared with the total time of anonymization (shown in Figure 2 (a) & (b)). Due to space limit, we omit the result.

Information Loss. We use the metric in Section 3 to measure the information loss. Intuitively, the smaller the ratio is, the better is the data utility. From our experiment results, we observe that our information loss is at most 0.31. This proves the effectiveness of our approach.

8 Conclusion

In this paper, we studied the problem of privacy-preserving publishing of data that contains full functional dependencies. We formally defined the privacy model, (d, ℓ)-*inference*, and developed robust and efficient algorithms that anonymize the data with minimized information loss. Our empirical studies using both synthetic and real datasets demonstrated the efficiency and effectiveness of our algorithm.

For the future work, we will consider multiple FFDs. Furthermore, we will move to CFDs, i.e., the FDs $X \rightarrow Y$ that do not hold for all values of X and Y. It turns out that the CFD-based analysis is far more intriguing than FFDs, and the (d, ℓ)-*inference* model fails to effectively defend against its consequent privacy attack. Thus in the future, we plan to strengthen the (d, ℓ)-*inference* model to defend against the CFD-based attack.

References

1. Iyengar, V.S.: Transforming Data to Satisfy Privacy Constraints. In: SIGKDD (2002)
2. Kifer, D.: Attacks on Privacy and deFinetti's Theorem. In: SIGMOD 2009 (2009)
3. Li, N., Li, T.: t-Closeness: Privacy Beyond K-anonymity and l-diversity. In: ICDE 2007. Datasets, SIGMOD 2006 (2007)
4. Machanavajjhala, A., Gehrke, J., Kifer, D., Venkitasubramaniam, M.: l-Diversity: Privacy Beyond k-Anonymity. In: ICDE 2006 (2006)
5. Martin, D.J., Kifer, D., Machanavajjhala, A., Gehrke, J., Halpern, J.Y.: Worst-Case Background Knowledge for Privacy-Preserving Data Publishing. In: ICDE 2007 (2007)
6. Rastogi, V., Suciu, D., Hong, S.: The boundary between privacy and utility in data publishing. In: VLDB 2007 (2007)
7. Samarati, P., Sweeney, L.: Generalizing Data to Provide Anonymity when Disclosing Information. In: PODS (1998)
8. Samarati, P.: Protecting Respondents' Identities in Microdata Release. In: TKDE (2001)
9. Sweeney, L.: K-anonymity: A Model for Protecting Privacy. International Journal on Uncertainty, Fuzziness and Knowledge-based Systems 10(5), 557570 (2002)
10. Tao, Y., Pei, J., Li, J., Xiao, X., Yi, K., Xing, Z.: Hiding Correlation by Independence Masking. In: ICDE 2010 (2010)
11. Wong, R.C., Li, J., Fu, A.W., Wang, K.: (α, k)-Anonymity: An Enhanced k-Anonymity Model for Privacy-Preserving Data Publishing. In: SIGKDD (2006)

Scalable Splitting of Massive Data Streams

Erik Zeitler and Tore Risch

Department of Information Technology
Uppsala University
Sweden
erik.zeitler@it.uu.se, tore.risch@it.uu.se

Abstract. Scalable execution of continuous queries over massive data streams often requires splitting input streams into parallel sub-streams over which query operators are executed in parallel. Automatic stream splitting is in general very difficult, as the optimal parallelization may depend on application semantics. To enable application specific stream splitting, we introduce splitstream functions where the user specifies non-procedural stream partitioning and replication. For high-volume streams, the stream splitting itself becomes a performance bottleneck. A cost model is introduced that estimates the performance of splitstream functions with respect to throughput and CPU usage. We implement parallel splitstream functions, and relate experimental results to cost model estimates. Based on the results, a splitstream function called autosplit is proposed, which scales well for high degrees of parallelism, and is robust for varying proportions of stream partitioning and replication. We show how user defined parallelization using autosplit provides substantially improved scalability (L = 64) over previously published results for the Linear Road Benchmark.

Keywords: Distributed stream systems, parallelization, query optimization.

1 Introduction

Data Stream Management Systems (DSMS) are becoming commonplace for a wide range of scientific and industrial applications, with high-volume data streams and queries that involve complex computations. Scalable execution in such applications requires parallelization. The parallelization of a query is called the *parallelization strategy*. In general, it is very difficult to automate the parallelization strategy, since the optimal parallelization may depend on application semantics. Our approach is to extend the query language with second-order functions to enable the user to specify non-procedural parallelization strategies. These functions split an input stream into large collections of parallel streams over which queries produce collections of result streams. Depending on the application, this collection of result streams can be merged, aggregated or further partitioned.

Splitstream functions partition and/or replicate input streams into a collection of streams. For each tuple in the input stream, splitstream decides whether the tuple should be sent to one specific DSMS node (partitioning) or many DSMS nodes (replication). Partitioning a stream is necessary when executing expensive queries. Replication is required, e.g., when aggregates are computed over data distributed over many local DSMS nodes. A splitstream function is compiled and optimized into a *splitstream plan*. We show how to automatically generate an optimized splitstream plan with high throughput and low CPU cost given a non-procedural splitstream specification. A generic splitstream function *autosplit* is defined that generates an optimized parallel splitstream plan based on a simple decision rule. To investigate the scalability of splitstream functions, we have parallelized an implementation of the Linear Road Benchmark (LRB), which is called *scsq-plr*. We focus on the performance bottleneck in the parallelization strategy of *scsq-plr*, which is splitting the stream of position reports and account balance queries. In summary, we present the following results:

- Splitstream functions are introduced, which enable non-procedural user defined specification of parallelization strategies.
- A cost model is introduced that estimates CPU utilization and throughput of splitstream plans.
- A theoretically optimal tree shaped splitstream plan is devised that has maximum throughput according to the cost model. This plan is compared with other splitstream plans.
- A generic splitstream function *autosplit* automatically generates tree shaped splitstream plans. *Autosplit* is shown to improve the scalability of LRB substantially.

2 Splitstream Functions

A *stream function*, *Q(S, …)* → *So* is a parameterized query that transforms one or more input stream arguments *S* into one or more output streams *So*. A *parallelization function* operates on collections of streams, and is used for specifying parallel executions of stream functions. Fig. 1 illustrates three basic classes of parallelization functions; *splitstream*, *mapstream*, and *mergestream*. *splitstream* splits an input stream into two or more output streams. The number of output streams of a splitsteam is called its *width*. *mapstream* applies a stream function on each stream in a collection of streams, while *mergestream* merges or joins a collection of streams into a single output stream. Examples of *mergestream* functions are stream union and windowed stream join. Although all parallelization functions are used in the final evaluation experiment, the focus of this paper is to optimize splitstream functions since they are shown to be a performance bottleneck.

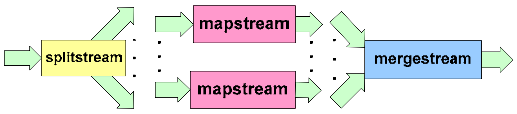


Fig. 1. Splitstream, mapstreams, and mergestream

A splitstream function has the basic signature *splitstream(stream s, integer w, function rfn, function bfn)* → *vector of stream sv*. The input stream *s* is split into *w* output streams in the vector *sv*. The first functional argument *rfn* is the *routing function*, having signature *rfn(object tpl, integer w)* → *integer*, which returns the output stream number (between 0 and *w* – 1) for each tuple that should be routed to a single output stream. The functional argument *bfn(object tpl)* → *boolean* is the *broadcast function*, which returns true for tuples to be broadcasted to all output streams. *bfn* and *rfn* return nil for tuples that should neither be broadcasted nor routed. *rfn* and *bfn* are defined declaratively in the query language by the user.

2.1 Parallelizing LRB

LRB [1] simulates a traffic system of expressways with variable tolling that depends on the utilization of the roads and the presence of accidents. Vehicles undertake journeys in the expressway system consisting of *L* expressways while emitting stream of position reports. An implementation must respond correctly to the continuous and historical queries of the benchmark within the allowed maximum response time (MRT). The number of expressways that an implementation is able to handle is called the *L-rating* of the implementation. An LRB implementation can be seen as a stream function *LR(S)* → *So*. The LRB input stream *S* consists of four kinds of tuples; *P*, *A*, *D*, and *E* (event type 0, 2, 3, and 4, respectively), of which 99% are position reports *P*. The rest of the tuples are account balance queries *A* (0.5%), daily expenditure queries *D* (0.1%), and estimated travel time queries *E* (0.4%). Currently, *E* tuples are ignored [1]. The *D* tuples are computed over historical data, their frequency is very low, and the allowed MRT is 10 sec, so any DBMS can respond to *D* tuples within the required time. Allowed MRT for *P* and *A* tuples are five seconds. Since these tuples are very frequent, they have to be processed efficiently. The input stream rate increases continuously during the 180 minutes of the simulation. The result stream *So* contains toll and accident alerts (event type 0 and 1), and query responses (event type 2 and 3). Some position reports do not result in toll alerts, so the rate of *So* is less than that of *S*.

Our single node LRB implementation *scsq-lr* [17], spent most of its CPU time computing segment statistics. This processing is local to each expressway, i.e., events on one expressway are independent of events on other expressways. Thus, the key to efficient parallelization is to partition the input stream into *L* parallel streams, and execute one instance of *LR()* for each expressway, as is employed in *scsq-plr*. The *A* tuples require account balance information. In *scsq-plr*, a local account table is maintained on each *LR()*, so that vehicles accumulate account balance locally on each expressway. Then, account balance queries must aggregate account data from all expressways. Therefore, all *A* tuples are broadcasted to all DSMS nodes running *LR()*.

Fig. 2 illustrates this parallelization strategy for *L* = 4. The input stream is first split by *splitstream*$_D$, whose routing function *rfnD(e,w)* is defined as:

```
create function rfnD(Event e, Integer w) → Integer as
select i from integer i where
(eventtype(e)<3 and i=0) or (eventtype(e)=3 and i=1);
```

In *splitstream*$_X$, the body of *rfnX(e,w)* is `select expressway(e) where eventtype(e)=0`, while *bfnX(e)* is defined as `select eventtype(e)=2`. Each stream from *splitstream*$_X$ is processed by an *lr* node (executing *LR()*), whose result stream is split by *splitstream*$_O$ using *rfnO(e,w)* defined as `select event-type(e)`. *splitstream*$_O$ and *splitstream*$_D$ do not broadcast, so they have no *bfn*. All

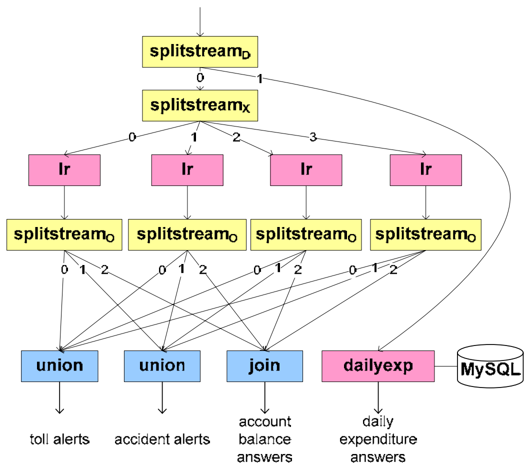


Fig. 2. The parallelization strategy in *scsq-plr*. $L = 4$

toll and accident alert result streams are merged with union-all. Account balance answers from each *splitstream*$_O$ are joined on query id and added together.

2.2 Single Process Splitstream

A splitstream function is naïvely implemented by a single process splitstream operator *fsplit*, its modules being shown in Fig. 3. The input stream S is read and de-marshalled by the *consume* module. In the *process* module, *rfn* and *bfn* are called for each tuple. Each *emit* module marshals and emits tuples to its output stream So_i, i=0…w–1.

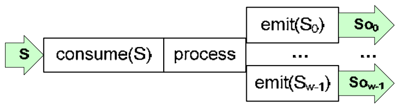


Fig. 3. Modules of *fsplit*

The rate Φ of a stream is defined as the number of tuples per second. The CPU cost C for executing *fsplit* in Fig. 3 is computed as

$$C = \Phi\big(cc + cp(o + r + w \cdot b) + ce(r + w \cdot b)\big). \tag{1}$$

In Equation 1, the consume cost cc measures reading and de-marshalling one input tuple, the process cost cp measures the execution of *rfn* and *bfn* per input tuple, and the emit cost ce measures emitting a tuple. b is the *broadcast percentage*, which is the proportion of tuples in the input stream to be emitted to all w output streams according to *bfn*. Notice that b is multiplied by w. r is the *routing percentage*, i.e. the proportion of tuples to be routed according to *rfn*, while o is the *omit percentage*, which is the proportion of tuples that are not emitted at all. As a tuple is either broadcasted, routed, or omitted, $r + b + o = 1$. Thus, the cost C decreases if o increases because of smaller emit cost. Assuming *rfn* routes each tuple with equal probability for all output streams $So_0 \ldots So_{w-1}$, the rate of each output stream is $\Phi o_i = \Phi \cdot (b + r / w)$ for all i.

For *scsq-plr*, Table 1 shows percentages o, b, and r, widths w, and output stream rates Φ of *splitstream*$_D$, *splitstream*$_X$, and *splitstream*$_O$, respectively. E (0.4%) tuples are dropped by *splitstream*$_D$. P (99%) and A (0.5%) tuples are routed to *splitstream*$_X$, and D (0.1%) tuples are routed to *dailyexp()*. *splitstream*$_X$ broadcasts A tuples to all *lr()* nodes and routes P tuples of expressway $j = 0 \ldots L{-}1$ to the corresponding *lr()*. Thus, *splitstream*$_X$ has $b = 0.5\% / 99.5\% \approx 0.5\%$, and $r = 99\% / 99.5\% \approx 99.5\%$. Each *splitstream*$_O$ routes the low rate result stream Φr_i from one *lr()* node.

According to Equation 1, the cost of *fsplit* increases when w is increasing if $b > 0$. Therefore, the cost of *splitstream*$_X$ increases when scaling L, turning *splitstream*$_X$ into the bottleneck when executing *scsq-plr* with high L. Stream replication is a scalability problem for large w, even if b is very close to zero, as in LRB.

Table 1. Tuple percentages, widths, and output stream rates of splitstream functions in LRB

	o	b	r	w	Φ	
D	0.4%	0%	99.6%	2	$\Phi_X = 99.5\% \cdot \Phi$	$\Phi_D = 0.1\% \cdot \Phi$
X	0%	0.5%	99.5%	L	$\Phi_i = \Phi_X \cdot (0.5\% + 99.5\% / L)$	
O	0%	0%	100%	3	$\Phi r_i < \Phi_i$	

3 Splitstream Trees

To alleviate the bottleneck in *splitstream*$_X$ when scaling w, we propose a hierarchical splitstream plan, called a *splitstream tree*. Each level ℓ in a splitstream tree is numbered, starting from 1 at the root to the depth d. Each node in the tree executes *fsplit*, and the width of the nodes on level ℓ is called the *fanout* f_ℓ of level ℓ. A hierarchical hash function defined in this section enables any user defined *rfn* to be executed in a splitstream tree. Furthermore, we introduce a cost model for splitstream trees. Using this cost model, a splitstream tree with maximum throughput can be generated if r and b are known. We compare its performance to a practical splitstream tree, which does not require knowledge of r and b.

3.1 Multi-level Hash Function

Since each level ℓ in a splitstream tree has fanout f_ℓ, the result of *rfn* on level ℓ must be an integer in range $[0, f_\ell{-}1]$. In addition, a splitstream tree must result in the same set of output streams as that of *fsplit*. To fulfill these requirements, the hierarchical hash function defined in Equation 2 is applied on the result of the routing function *rfn(t)* at each level ℓ and tuple t.

$$h_\ell(rfn(t)) = \mathrm{mod}\left(\left\lfloor \frac{rfn(t)}{\lambda_{\ell-1}} \right\rfloor, f_\ell\right). \qquad (2)$$

The denominator $\lambda_{\ell-1}$ of Equation 2 is the *cumulative fanout* of level $\ell - 1$, i.e., the fanout that the tuple has undergone in the tree levels above level ℓ. The cumulative fanout at the root is $\lambda_0 = f_0 = 1$, and the cumulative fanout λ_ℓ is

$$\lambda_\ell = \prod_{k \le \ell} f_k \,. \tag{3}$$

The output streams of a node at level ℓ are denoted $So^{(\ell)}{}_i$, $i = 0 \ldots f_\ell - 1$. For example, if *splitstream*$_X$ in Fig. 2 is executed as a splitstream tree with $f_1 = 2$ and $f_2 = L/2$, then $h_1(rfn)$ routes position reports of even-numbered expressways to output stream $So^{(1)}{}_0$ and position reports of odd-numbered expressways to $So^{(1)}{}_1$, according to Equation 2. On level 2, $h_2(rfn)$ routes tuples of expressway number x to $So^{(2)}{}_i$, $i = \lfloor x/2 \rfloor$. *bfn* is the same in all nodes, so that one copy of each broadcast tuple arrives at each leaf.

3.2 A Cost Model for Splitstream Trees

In the following discussion, we assume that then omit percentage $o = 0$, as in our *splitstream*$_X$ example. If $b > 0$ (and thus $r < 1$), the routing percentage decreases at each level. This is because the number of tuples to broadcast stay the same in all output streams, whereas the number of tuples to be routed decreases per level. Equation 4 defines the routing and broadcasting percentages r_ℓ and b_ℓ at level ℓ.

$$r_\ell = \frac{r}{r + b \cdot \lambda_{\ell-1}}; \quad b_\ell = \frac{b \cdot \lambda_{\ell-1}}{r + b \cdot \lambda_{\ell-1}}. \tag{4}$$

The rate of one of the output streams at level ℓ is $\Phi o^{(\ell)}$.

$$\Phi o^{(\ell)} = \Phi \cdot \left(b + \frac{r}{\lambda_\ell} \right). \tag{5}$$

The cost C_ℓ of executing a node at level ℓ in a splitstream tree depends on the output stream rate from level $\ell - 1$ according to Equation 5.

$$C_\ell = \Phi o^{(\ell-1)} \cdot \left(cc + (cp + ce) \cdot (r_\ell + f_\ell \cdot b_\ell) \right). \tag{6}$$

The *emit capacity E* of a node executing the *fsplit* operator is defined as its maximum stream rate. The *throughput* $\Phi_{\max}$ of a splitstream tree is limited by E and by the level in the splitstream tree with the highest cost.

$$\Phi_{\max} = \frac{E}{\max_\ell (C_\ell)}. \tag{7}$$

Finally, the total cost of a splitstream tree of depth d can be estimated by adding the splitstream costs for all nodes at each level. The number of nodes at level ℓ is $\lambda_{\ell-1}$.

$$C = \sum_{\ell=1}^{d} \lambda_{\ell-1} \cdot C_\ell \,. \tag{8}$$

3.3 Maxtree and Exptree Splitstream Trees

Assuming that the percentages r and b are known and constant, it is possible to construct an optimal splitstream tree that maximizes the throughput according to the cost model. We call this splitstream tree *maxtree*, which maximizes the throughput while minimizing the total cost. The cost at level $\ell = 1$ is minimized by choosing $f_1 = 2$, so that $C_1 = \Phi \cdot (cc + (r + 2b) \cdot (cp + ce))$. Levels are added until $\lambda_d \geq w$. By choosing a fanout f_ℓ of each level $\ell > 1$ such that $C_\ell \leq C_1$, we ensure that no downstream level in the splitstream tree will be a bottleneck. The total cost in Equation 8 is minimized by minimizing the number of splitstream tree levels. The number of levels are minimized by maximizing f_ℓ on all levels $\ell > 1$, while keeping $C_\ell \leq C_1$. Solving f_ℓ for $C_\ell = C_1$ using Equation 6 obtains the following formula for the optimal fanout f_ℓ at level ℓ (see [17] for details).

$$f_\ell = 2 + \left\lfloor \frac{r}{b}\left(1 + \frac{cc}{cp + ce}\right)\left(1 - \frac{1}{\lambda_{\ell-1}}\right) \right\rfloor . \tag{9}$$

The ratio between the costs $a = cc/(cp+ce)$ depends on the costs of *rfn* and *bfn* and on the properties of the computing and network environments. In general, these parameters are unknown, so the formula in Equation 9 cannot be determined. Therefore, *maxtree* can only be used for comparison in controlled experiments where a is known. We determined $a = 1.08$ for *splitstream*$_X$ in a preliminary experiment. To simplify the theoretical discussion of *maxtree*, a was rounded to 1.

Equation 9 shows that optimal fanout f_ℓ increases quickly for small $\ell > 1$ if $r > 0$. Based on this observation, we introduce a splitstream tree called *exptree*, which increases its fanout for each level with a constant factor. *exptree* was set to generate trees with $f_1 = 2$, and $f_\ell = 2 \cdot f_{\ell-1}$ for all $\ell > 1$. We show that the performance of *exptree* will be almost as good as that of *maxtree*, without the need to know a, r, and b.

3.4 Theoretical Evaluation

Throughput and total CPU cost were estimated for the splitstream plans using Equations 7 and 8, assuming $cc = 1$ and $a = 1$. In a *scale-up* evaluation, w was scaled from 2 to 256 while keeping $b = 0.5\%$, as in *splitstream*$_X$. In a *robustness* evaluation, b was scaled from 0 to 1 while keeping $w = 64$. Fig. 4 shows the estimated performance.

In the scale-up evaluation, the estimated throughput was plotted in Fig. 4 (a) as the percentage of emit capacity E. The estimated total CPU cost was plotted in Fig. 4 (b). As expected, the single-process *fsplit* degrades when w increases. On the other hand, *fsplit* also consumes the least total CPU. The CPU cost of *exptree* increases when a new tree level is added, e.g. when increasing w from 8 to 16. For such small values of $b = 0.5\%$ as in LRB, *maxtree* generates a shallower tree and thus consumes less CPU resources than *exptree*.

When scaling b in the robustness evaluation, Fig. 4 (d) shows that the CPU cost of *maxtree* increases sharply when b increases. If $b \approx 1$ in to Equation 9, $f_\ell = 2$ on all *maxtree* levels, resulting in a binary tree. A splitstream tree with so many nodes consumes a lot of CPU. Fig. 4 (c) shows that all splitstream functions have the same

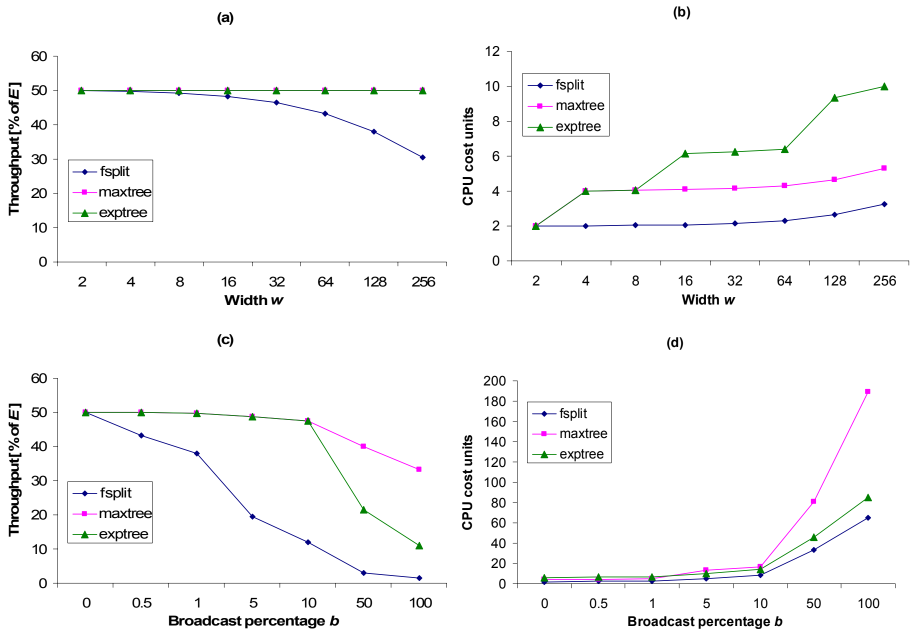


Fig. 4. (a) Estimated throughput and (b) total CPU cost, $b = 0.5\%$. (c) Estimated throughput and (d) total CPU cost, $w = 64$.

throughput for $b = 0$, but the throughput of *fsplit* drops quickly when b increases. For moderate values of b (up to 10%), the estimated throughput of *exptree* is the same as that of *maxtree*. For higher values of b, the estimated throughput of *exptree* is lower, however much better than *fsplit*.

4 Experimental Setup

The splitstream functions were implemented using our prototype DSMS SCSQ [21]. Queries and views are expressed in terms of typed functions in SCSQ's functional query language SCSQL, resulting in one of three collection types *stream*, *bag*, and *vector*. A stream is an object that represents ordered (possibly unbounded) sequences of objects, a bag represents relations, and a vector represents bounded sequences of objects. For example, vectors are used to represent stream windows, and vectors of streams are used to represent ordered collections of streams.

Queries are specified using SCSQL in a client manager. The distributed execution plan of a query forms a directed acyclic graph of stream processes (SPs), each emitting tuples on one or more streams. Continuous query definitions are shipped to a coordinator. Unless otherwise hinted, the coordinator dynamically starts new SPs in a round robin fashion over all its compute nodes, so that the load is balanced across the cluster. The coordinator returns a handle of each newly started SP.

In the SPs, a cost-based query optimizer transforms each query to a local stream query execution plan (SQEP), by utilizing the query optimizer of Amos II [9]. A SQEP reads data from its input streams and delivers data on one or more of its output streams. Stream drivers for several communication protocols are implemented using non-blocking I/O and carefully tuned buffers. A timer flushes the output stream buffers at regular time intervals to ensure that no tuples will remain for too long. The SCSQ kernel is implemented in C, where SQEPs are interpreted. SQEPS may call the Java VM to access DBMSs over JDBC. Thus, an SP may be stateful in that it stores, indexes, and retrieves data using internal main memory tables or external databases. In *scsq-plr*, local main memory tables are used to store account balance data, and MySQL is used to store daily expenditure data.

In our experiments, each SP is a UNIX process on a cluster of compute nodes featuring two quad-core Intel® Xeon® E5430 CPUs @ 2.66GHz and 6144 KB L2 cache. Six such compute nodes (48 cores in total) were available for the experiments. For large splitstream trees, there were fewer CPUs than SPs. Then, some SPs were co-located on the same CPU. For inter-node communication, TCP/IP was used over gigabit Ethernet. Intra node communication used TCP/IP over the loopback interface. Throughput is computed by measuring the execution time of SCSQ over a finite stream. The CPU usage of each SP is determined using a profiler in SCSQ that measures the time spent in each function by interrupt driven sampling.

5 Preliminary Experiments

Two preliminary experiments were performed. The purpose of the first one is twofold: We show that the emit capacity for moderately sized tuples is bound by the CPU and not by the network. We also show that the emit capacity E for an SP, and thus the cost, is the same for moderately sized tuples no matter if streaming inter or intra node. Since the cost is the same, the scheduling of SPs is greatly simplified.

One SP was streaming tuples of specified size to another SP, which counted them. Intra node streaming was performed with the SPs on the same compute node, while they were on different nodes for inter node streaming. The emit capacity is shown in Fig. 5 (a), with less than 3.5% relative standard deviation. For tuples of moderate size, the emit capacity is the same for inter and intra node streaming. LRB input stream tuples have 15 attributes, occupying 83 bytes including header. The network bandwidth consumption is 143 Mbit/s for these tuples, which is significantly less than the capacity of a gigabit Ethernet interface. Streaming moderately sized tuples as in LRB is CPU bound, because of the overhead of marshalling and (de)allocating many small objects. For tuples of size greater than 512 bytes, the intra node throughput is better. Usually however, tuples are smaller.

The purpose of the second preliminary experiment is to measure consume, process, and emit costs (*cc*, *cp*, and *ce*) *splitstream*$_X$ in our environment, as required by *maxtree*. We do that by executing *splitstream*$_X$ as an *fsplit* with $w = 1$. One SP generated a stream of 3 million tuples. A second SP applied *fsplit* with $w = 1$ on the stream from the first SP, using the *rfn* and *bfn* of *splitstream*$_X$. A third SP counted the number of tuples in the single output stream from *fsplit*.

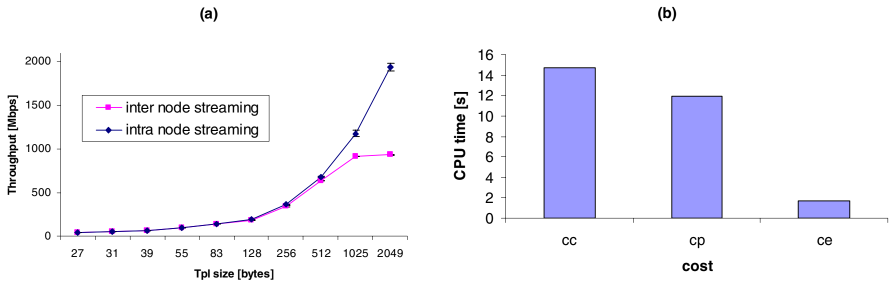


Fig. 5. (a) Inter and intra node emit capacity, (b) CPU time breakdown for *fsplit* with $w = 1$

The CPU times obtained from the *fsplit* SP are shown in Fig. 5 (b). Using these CPU times, $a = cc / (cp + ce) \approx 1.08$, which is used in all *maxtree* experiments. Furthermore, the throughput of this simple splitstream was $\Phi_{max} = 109 \cdot 10^3$ tuples per second (relative standard deviation 0.6%). In LRB, the maximum input stream rate is 1670 tuples per second and expressway, so this throughput corresponds to 65 (109000/1670) expressways in LRB. No splitstream tree can be expected to have higher throughput than *fsplit* with $w = 1$. Thus, no splitstream tree will be able to split the input stream of LRB for $L > 65$.

6 Experimental Evaluation

The goal with the experimental evaluation is to investigate the properties of the splitstream plans in a practical setting. Throughput and total CPU consumption were studied in a *scale-up* experiment and a *robustness* experiment, set up in the same way as in the analytical evaluation. In order to establish statistical significance, each experiment was performed five times and the average is plotted in the graphs.

Fig. 6 shows the throughput and CPU usage of the splitstream trees. Error bars (barely visible) show one standard deviation. All experimental results agree perfectly with the theoretical estimates in Fig. 4 with one exception: The measured throughput of *maxtree* shown in Fig. 6 (c) was significantly lower for large values of b than estimated. This is because the total CPU usage exceeds the CPU resources available for our experiments. If resources were abundant, *maxtree* should have been feasible for splitting streams with a high broadcast percentage. If resources are limited, *exptree* is shown to achieve the same throughput as *maxtree* at a smaller CPU cost. The experiments confirm that our cost model is realistic.

6.1 Autosplit

We observe that *exptree* achieves the same scale-up as *maxtree*. Furthermore, the robustness of *exptree* is the same as that of *maxtree* when resources are not abundant. Based on these results, we implement *autosplit* using the following decision rule: If *bfn* is present, generate an *exptree*. If only *rfn* is present and thus $b = 0$, generate a single *fsplit*, since a single *fsplit* has the same throughput as the splitstream trees for $b = 0$, but consumes less CPU.

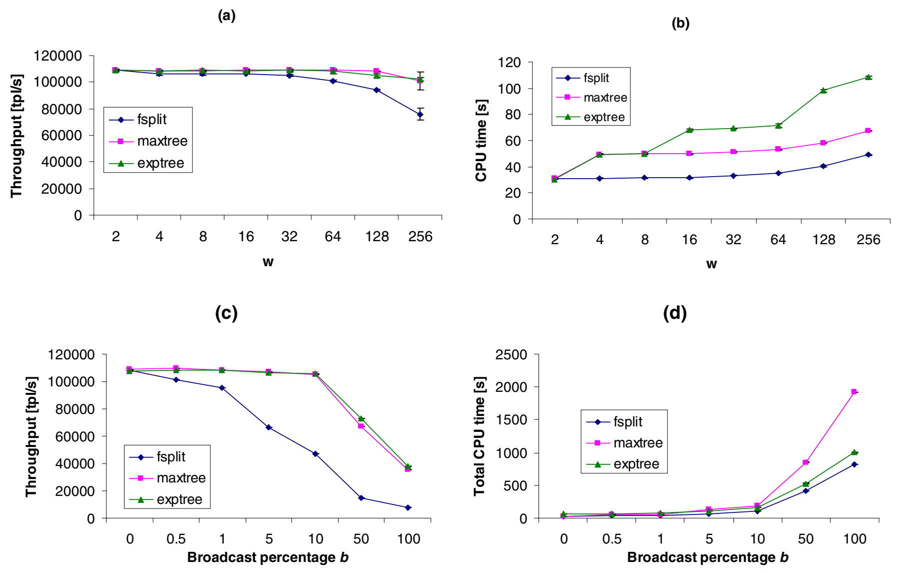


Fig. 6. (a) Measured throughput and (b) measured total CPU cost for $b = 0.5\%$. (c) Measured throughput and (d) measured total CPU cost for $w = 64$.

6.2 LRB Performance

To verify the high scalability of *autosplit*, it was used as the splitstream function in *scsq-plr* as shown in Fig. 2. *autosplit* generated an *exptree* for *splitstream*$_X$ and *fsplit* for *splitstream*$_D$ and *splitstream*$_O$ since they had no *bfn*. The performance of LRB using *autosplit* is compared to LRB using *fsplit* in all splitstream functions. To simplify the experiments, the *dailyexp()* node was disabled since the daily expenditure processing has no bearing on scalability of LRB stream processing in *scsq-plr*.

When using the round robin scheduler of the coordinator described in Section 0, *scsq-plr* with *autosplit* achieved $L = 52$. The limiting factor was that the first node of the plan was not granted enough CPU resources, because too many SPs were assigned to the same multi-core compute node. By adding a hint to the coordinator to limit the number of SPs on the first compute node, the L-rating for *autosplit* improved to $L = 64$, as illustrated by Fig. 7. The *y*-axis is the MRT, and the *x*-axis is the number of minutes into the simulation. *fsplit* keeps up until minute 125, when response time accumulates and exceeds the allowed MRT at 129 minutes. When $b = 0.5\%$ as in *splitstream*$_X$, the maximum throughput of *fsplit* with $w = 64$ is 100000 tpl/sec according to the results in Fig. 6(c). At 125 minutes, the stream rate for $L = 64$ is getting close to 100000 tpl/sec. Thus, *fsplit* is unable to keep up with the increasing input stream rate. Since the maximum throughput of *exptree* is higher, *autosplit* achieves the higher L-rating of $L = 64$. The bumps in the curves are because of cron jobs executing on the compute nodes beyond our control.

In conclusion, we have shown that *fsplit* cannot achieve $L = 64$ in LRB, and that smart scheduling is necessary to take full advantage of *autosplit*. *fsplit* with smart

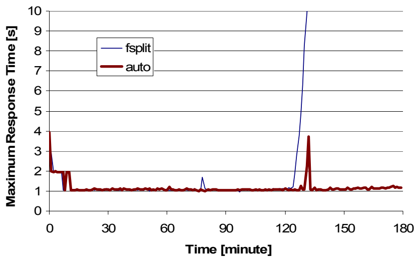


Fig. 7. Maximum response time for $L = 64$

scheduling was measured to achieve $L = 52$. Notice that in standard LRB, the improvement with *autosplit* could not be expected to be very large. However, as indicated theoretically by Fig. 4 (a) and experimentally in Fig. 6 (c), the gain will be bigger if the broadcast percentage b is greater.

7 Related Work

This paper complements other work on parallel DSMS implementations [4, 8, 12, 15, 19], by allowing the user to specify non procedural stream splitting, and by parallelizing the execution of stream splitting. This allows parallel execution of expensive queries over massive data streams.

In previous work [22], we introduced stream processes, allowing the user to manually specify parallel stream processing. The stream splitting proved to be very efficient for online spatio-temporal optimization of trip grouping [7], based on static or dynamic routing decisions. Similarly, GSDM [12] distributed its stream computations by selecting and composing distribution templates from a library, in which some basic templates were defined including both splitting and joining. By contrast, the stream splitting in this paper is specified through declarative second order splitstream functions, allowing optimizable stream splitting insensitive to the percentages of tuples to broadcast or route.

Gigascope [4] was extended with automatic query dependent data partitioning in [14] for queries that monitored network streams. The query execution was automatically parallelized by inferring partitioning sets based on aggregation and join attributes in the queries. The stream splitting was performed in special hardware, which provided high throughput. By contrast, we have developed a method to split streams involving both routing and broadcasting by generating efficient hierarchical splitstream plans executing on standard PCs. Furthermore, splitstream functions allow the user to declaratively specify splitstream strategies, which allows parallelization of queries that cannot be parallelized automatically.

Efficient locking techniques were developed in [5] to parallelize aggregation operators using threads. Since SCSQ uses processes instead of threads for parallelization, locking is not an issue.

Partitioning a query plan by statically distributing the execution of its operators proved to be a bottleneck in [13]. In [2], query plans were partitioned by dynamically migrating operators between processors. However, expensive operators are still bottlenecks. In our work, the bottleneck was overcome by splitting the input stream into several parallel streams, and further reduced by parallelizing the stream splitting itself. Furthermore, allowing both routing and broadcasting provide a powerful method to parallelize queries, as shown by *scsq-plr*.

The Flux operator [18] dynamically repartitions stateful operators in running streams by adaptively splitting the input stream based on changes in load. By contrast, we have studied user defined stream splitting. Dynamic scheduling of distributed operators in continuous queries has been studied in [19] and [23]. A dynamic distributed scheduling is introduced in [19] based on knowledge about anti-correlations in load between different independent operators in a plan. In [23], stream operators are dynamically migrated between compute nodes based on the current load of the nodes. By contrast, this paper concentrates stream splitting for parallel processing downstream. However, scheduling proved to be important, and future work should investigate the effectiveness of these approaches when used with parallelization functions.

Dryad [10] generalizes Map-Reduce [6] by implementing an explicit process graph building language where edges represent communication channels between vertices representing processes. By contrast, SCSQ users specify parallelization strategies over streams on a higher level using declarative second order parallelization functions. These parallelization functions are automatically translated into parallel execution plans (process graphs) depending on the arguments to the parallelization functions.

SCOPE [3] and Map-Reduce-Merge [20] are more specialized than Dryad, providing an SQL-like query language over large distributed files. The queries are optimized into parallel execution plans. Dryad, Map-Reduce-Merge, and SCOPE operate on sets rather than streams. None of these provide parallelization functions.

Out of the existing implementations of LRB, IBM's Stream Processing Core (SPC) is the only attempt to parallelize the execution [13]. The SPC implementation of LRB was partitioned into 15 building blocks, each of which performed a part of the implementation. One processing element computed all segment statistics on a single CPU, which proved to be a bottleneck. With the SCSQ implementation and *autosplit*, we achieved over 25 times the L-rating of the SPC implementation by user defined parallelization. The performance difference between SCSQ and SPC illustrates (i) the importance of how the execution is parallelized; and (ii) the usefulness of splitstream functions where the user provides application knowledge for the parallelization declaratively by specifying *rfn* and *bfn*.

For streams, *rfn* and *bfn* are analogous to fragmentation and replication schemes for distributed databases [16]. However, for distributed databases the emphasis is mainly on distributing data without skew. In our case, there are orders of magnitude higher response time demands on stream splitting and replication than on disk data fragmentation and replication. Therefore, the performance of stream splitting is critical.

8 Conclusions and Future Work

We investigated the performance of *splitstream functions,* which are parallelization functions that provide both partitioning and replication of an input stream into a collection of streams. A splitstream function is compiled into a *splitstream plan*. We first defined a theoretical cost model to estimate the resource utilization of different splitstream plans, and then investigated the performance of these splitstream plans experimentally using the SCSQ DSMS. Based on both theoretical and experimental evaluations, we devised the splitstream function *autosplit*, which splits an input stream, given the degree of parallelism, and two functions specifying how to distribute and partition the input stream. The *routing function* returns the output stream number for each input tuple that should be routed to a single output stream. The *broadcast function* selects the tuples that should be broadcasted to all output streams. *autosplit* was shown to generate a robust and scalable execution plan with performance close to what is theoretically optimal for a tree shaped execution plan. *autosplit* was used to parallelize the Linear Road DSMS Benchmark (LRB), and shown to achieve an order of magnitude higher L-rating than other published implementations.

A simple scheduler was used in the experiments, which balanced the load evenly between the compute nodes for all splitstream plans. This scheduler achieved $L = 52$ in the LRB experiment. By hinting the scheduler not to overload the first node of the execution plan, the L-rating improved to 64, which is close to the theoretically maximum throughput for *scsq-plr* in our cluster environment.

As splitstream has shown to be sensitive to the cost of *rfn* and *bfn*, future work includes optimizing splitstream for a wider class of *rfn* and *bfn*. By devising a cost model like in [24], the scheduling of SPs can be further improved. The robustness of dynamic re-scheduling and SP migration should be investigated. It should be investigated whether other, non-tree shaped splitstream plans can improve performance further. Furthermore, other application scenarios are being studied within the iStreams project [11].

Acknowledgments. This project is supported by VINNOVA in the iStreams project. The experiments were performed using UPPMAX resources. We greatly appreciate the insightful comments by the anonymous reviewers.

References

1. Arasu, A., et al.: Linear Road: A Stream Data Management Benchmark. In: VLDB (2004)
2. Balazinska, M., Balakrishnan, H., Stonebraker, M.: Contract-Based Load Management in Federated Distributed Systems. In: NSDI (2004)
3. Chaiken, R., et al.: SCOPE: Easy and Efficient Parallel Processing of Massive Data Sets. In: VLDB (2008)
4. Cranor, C., Johnson, T., Spataschek, O., Shkapenyuk, V.: Gigascope: A Stream Database for Network Applications. In: SIGMOD (2003)
5. Das, S., Antony, S., Agrawal, D., El Abbadi, A.: Thread Cooperation in Multicore Architectures for Frequency Counting over Multiple Data Streams. In: VLDB (2009)
6. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI (2004)
7. Gidofalvi, G., Pedersen, T.B., Risch, T., Zeitler, E.: Highly scalable trip grouping for large-scale collective transportation systems. In: EDBT (2008)
8. Girod, L., Mei, Y., Newton, R., Rost, S., Thiagarajan, A., Balakrishnan, H., Madden, S.: XStream: A Signal-Oriented Data Stream Management System. In: ICDE (2008)

9. Risch, T., Josifovski, V., Katchaounov, T.: Functional Data Integration in a Distributed Mediator System. In: Gray, P.M.D., Kerschberg, L., King, P.J.H., Poulovassilis, A. (eds.) The Functional Approach to Data Management (2004)
10. Isard, M., et al.: Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. ACM SIGOPS Operating Systems Review 41, 59–72 (2007)
11. iStreams homepage, `http://www.it.uu.se/resnearch/group/udbl/html/iStreams.html`
12. Ivanova, M., Risch, T.: Customizable Parallel Execution of Scientific Stream Queries. In: VLDB (2005)
13. Jain, N., et al.: Design, Implementation, and Evaluation of the Linear Road Benchmark on the Stream Processing Core. In: SIGMOD (2006)
14. Johnson, S., Muthukrishnan, Shkapenyuk, V., Spatscheck, O.: Query-Aware Partitioning for Monitoring Massive Network Data Streams. In: SIGMOD (2008)
15. Liu, B., Zhu, Y., Rundensteiner, E.A.: Run-Time Operator State Spilling for Memory Intensive Long-Running Queries. In: SIGMOD (2006)
16. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 2nd edn. Prentice-Hall, Englewood Cliffs (1999)
17. SCSQ-LR homepage, `http://user.it.uu.se/~udbl/lr.html`
18. Shah, M.A., Hellerstein, J.M., Chandrasekaran, S., Franklin, M.J.: Flux: An Adaptive Partitioning Operator for Continuous Query Systems. In: ICDE (2002)
19. Xing, Y., Zdonik, S., Hwang, J.-H.: Dynamic Load Distribution in the Borealis Stream Processor. In: ICDE (2005)
20. Yang, H., Dasdan, A., Hsiao, R.-L., Parker, D.S.: Map-reduce-merge: simplified relational data processing on large clusters. In: SIGMOD (2007)
21. Zeitler, E., Risch, T.: Processing high-volume stream queries on a supercomputer. In: ICDE Workshops (2006)
22. Zeitler, E., Risch, T.: Using stream queries to measure communication performance of a parallel computing environment. In: ICDCS Workshops (2007)
23. Zhou, Y., Ooi, B.C., Tan, K.-L.: Efficient Dynamic Operator Placement in a Locally Distributed Continuous Query System. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 54–71. Springer, Heidelberg (2006)
24. Zhou, Y., Aberer, K., Tan, K.-L.: Toward massive query optimization in large-scale distributed stream systems. In: Middleware (2009)

Constraint-Aware Complex Event Pattern Detection over Streams[⋆]

Ming Li[1,3], Murali Mani[1], Elke A. Rundensteiner[1], and Tao Lin[2]

[1] C.S. Dept., Worcester Polytechnic Inst., Worcester, Massachusetts USA
[2] Research and Development, Amitive Inc., Redwood City, California USA
[3] Silicon Valley Laboratory, IBM Corp., San Jose, California USA
mingli3@us.ibm.com, {mmani,rundenst}@cs.wpi.edu, taolin@amitive.com

Abstract. In this paper, we propose a framework for constraint-aware pattern detection over event streams. Given the constraint of the input streams, our proposed framework on the fly checks the query satisfiability / unsatisfiability using a lightweight reasoning mechanism. Based on the constraint specified in the input stream, we are able to adjust the processing strategy dynamically, by producing early feedbacks, releasing unnecessary system resources and terminating corresponding pattern monitor, thus effectively decreasing the resource consumption and expediting the system response on certain situations. Our experimental study illustrates the significant performance improvement achieved by the constraint-aware pattern detection framework with little overhead.

1 Introduction

Event stream processing (ESP) [17][3][9][14] technologies enable enterprise applications such as algorithmic trading, RFID data processing, fraud detection and location-based services in telecommunications. The key applications of the ESP technologies rely on the detection of certain event patterns (usually corresponding to the exceptional cases in the application domain). Alerts will be raised after the target pattern has been detected in the form of system notifications or triggers. Such functionality is sometimes referred to as the *situation alert*, which corresponds to many key tasks in enterprises computing.

In many practical cases business events are generated based on pre-defined business logics, shown by the following two scenarios:

Supply Chain Management. Business Activity Monitoring (BAM) has been described by Gartner [1] as a technology that "allows business users real-time access to, and analysis of, important business indicators". One major BAM application is in supply chain management (SCM). The business events corresponding to the stream-line logistics flow in SCM follow a pre-defined procedure.

Network Anomalies Detection. Assume a firewall server monitoring the network packets between inside and outside machines. The server can maintain the

[⋆] This work has been partially supported by NSF under Grant No. NSF IIS-0414567.

statistics of all the network traffic flows. Anomalies are detected from statistical data sent as event streams [3], which are generated by workflow engines or simply customized programs following pre-defined schema.

In real-life event-based systems, constraints such as workflows often hold among the event data. For pattern detection over such event data streams, reasoning using the constraints enables us to (1) identify queries which are guaranteed to not lead to successful matches at the earliest, thereby helping us to terminate these long running pattern detection processes and release the corresponding CPU and buffer resources; (2) identify queries which can be guaranteed to surely lead to a future alert at the earliest (even though the matched result has not yet happened), thereby helping us to get prepared for upcoming situations. The above two are referred to as *detection of query unsatisfiability* and *detection of query satisfiability* separately.

Consider the following event query [17][3] in SCM, which monitors whether an item has passed several process steps of certain location in a certain order:

SEQ(SUPPLIER_WAREHOUSE, LABEL_CTR, SHELTER)

Without given constraint knowledge of the input events, the earliest we can say that the expected pattern cannot be matched over the event trace is after the whole event trace has been completely received and still no match has been found. Similarly, the earliest a situation alert could be triggered is after a match of the expected pattern corresponding to the alert has been fully received. Assume we are given the event constraint as the product transportation workflow shown in *Figure* 1. By such semantics of the input stream, if the item's arrival at the *logistics center* is notified, we can guarantee that no match can be found for the expected pattern in a future, since no *shelter* (which is a required pattern in the query) could appear in the coming trace. Thus the pattern monitor can be terminated at this moment. Similarly, if the item's arrival at the *retail warehouse* is notified, we can guarantee that the current event trace can surely lead to a future match for the expected pattern, since a coming *label center* followed by a *shelter* is indicated by the workflow. Thus an early alert can be triggered for helping the corresponding party get prepared for upcoming situations.

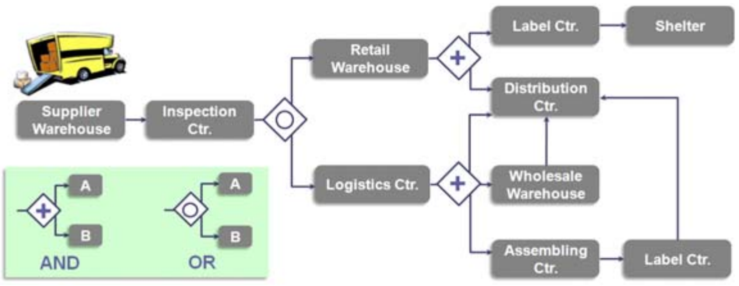


Fig. 1. Example Workflow in SCM

We propose a framework for constraint-aware pattern detection over event streams. and we have implemented our proposed framework in a prototype system called E-Tec (constraint-aware query **E**ngine for pattern de**Tec**tion over event streams) [13]. Given the constraint of the input event stream, E-Tec on the fly checks the query satisfiability / unsatisfiability using a lightweight reasoning framework. Based on such runtime constraint, E-Tec can adjust the processing strategy dynamically, by producing early feedbacks, releasing unnecessary resources (CPU and buffer) and terminating corresponding pattern monitor, thus effectively decreasing the resource consumption and expediting the system response on certain situation alerts. Our contributions include:

1. **Lightweight Constraint Checking.** Given the constraint of the input event stream at compile time, the query satisfiability / unsatisfiability is efficiently observed on the fly by E-Tec's *constraint engine* using our proposed checking algorithm. The process is made to be lightweight through decreasing the cost of runtime checking using our proposed automaton encoding algorithm for pre-computation. (Section 3)
2. **Execution Strategy.** We propose a query execution strategy following the *Event-Condition-Action* (ECA) rule-based framework. Real-time streaming event data input serves as the *events*. The constraint engine described earlier uses the checking algorithm to determine whether a set of specific *conditions* are satisfied at runtime. Based on the checking results, corresponding *actions* are taken on the fly such as monitor termination, buffer releasing and early situation alerts. (Section 4)
3. **System Integration and Experimental Studies.** Our constraint-aware pattern detection framework can be easily integrated with an automaton-based ESP engine by combing automaton applied for constraint checking with the automaton applied for pattern detection. Our prototype system E-Tec is implemented following such design patter. Based on E-Tec, we conduct experimental studies to demonstrate that our proposed techniques bring significant performance gains in memory and CPU usage. (Section 5)

State-of-the-Art. Recently the emergence of stream data processing had been extended to complex event processing on event streams [17][3][8]. Wu [17] proposes an expressive yet easy-to-understand language to support pattern detection over event streams, but constraint knowledge is not within the consideration of its query evaluation. In [3], a plan-based technique is used to perform streaming complex event detection across distributed sources. Its focus is mainly on handling pattern detection over event streams in a distributed environment. A rule-based ESP solution is provided in [8]. However, it only considers a limited number of rules instead of utilizing the complete input event constraint.

Roadmap. In Section 2 we give the preliminary for this paper. Section 3 provides our query satisfiability / unsatisfiability checking algorithms. Execution strategy is studied in Section 4. Our experimental results are presented in Section 5, followed by related work in Section 6 and conclusion in Section 7.

2 Preliminary

2.1 Event Data Model

Event Instance. An *event instance* is an occurrence of interest in a system, which can be either primitive or composite as further introduced below.

Event Type. Similar event instances can be grouped into an *event type*. That is, each event type E corresponds to a class of event instances which share set of common attributes. We use capitalized letters for event types such as E. and we use lower-case letters such as e to represent instances of event type E.

Event Stream and Event Trace. An *event stream* is heterogeneously populated with event instances of different event types. In most event processing scenarios, it is assumed that the input to the system is a potentially infinite stream which contains all events that might be of interest [17]. Such real-time input is referred to as an *event trace* (usually denoted as h), which evolves on the fly by receiving new instances as input. For an event trace h and an event type E, $E[h]$ denotes the set of all the event instances of E in h.

Temporal Aspects of Events. An event is associated with a unique timestamp, indicating the discrete ordering in the time domain. An event instance that happens instantaneously at a time point is called a *point event*. An event instance that occurs over a time interval is called a *interval event*. As a general representation for both the point and interval temporal semantics, for any event instance e_i, we use $e_i.ts$ and $e_i.te$ to denote the start and the end timestamp of the event instance e_i, respectively. The start and the end timestamps of a point event e are the same, which is simplified as $e_i.t$ (i.e., $e_i.ts = e_i.te = e_i.t$). For an event instance e, we use a pair of numbers as $e_{t1|t2}$ adjacent to it to represent the timestamp of e (denoting both the start and end time). For the point-based events, the representation is simplified as e_t, where the t adjacent to e denotes the time point when e happens. For easier understanding, we assume events to be point-based (thus event e's timestamp will be represented as $e.t$) in the rest of the paper. However our proposed constraint-aware pattern detection framework works with general interval event streams as well.

2.2 Event Query Model

Complex event pattern detection languages are studied in a number of existing works [3][17][7]. In this work we adopt the query language defined in [17] as follows to specify an *event pattern query*:

```
<Query>::= EVENT <pattern expression>
         [WHERE <equality conditions>]
```

The **EVENT** clause specifies temporal and logical relationships among events. As its algebraic translation, we use a standard set of event composite operators [17], which are AND, OR, SEQ and NEGATION. The sequence operator SEQ is used by the examples of this paper thus its definition is given below.

[**Sequence Operator**]. The sequence operator SEQ specifies a particular order in which the event instances of interest should occur and these event instances form a composite event instance. It takes a list of n ($n > 1$) event types as its parameters and outputs composite events $e' = <e1, e2, ..., en>$ defined as:

$$SEQ(E1, E2, ..., En)[h] = \{ < e1, e2, ..., en > \mid (e1.t < e2.t... < en.t) \wedge (< e1, ..., en > \in E1[h] \times ... \times En[h]) \}. \quad (1)$$

Example 1. The following example illustrates the computation of SEQ(A, B) given the event trace $h = \{ a_1, b_2, e_5, a_6, e_7 \}$. Remind that the small number adjacent to an event instance denotes the timestamp of the event. We have A[h]=$\{ a_1, a_6 \}$, B[h] = $\{ b_2 \}$ and A[h] $\times$ B[h] = $\{ < a_1, b_2 >, <a_6, b_2> \}$. The sequence result $<a_1, b_2>$ satisfies the condition $a_1.t < b_2.t$ (i.e., $1 < 2$). However, $<a_6, b_2>$ is not a correct result for $a_6.t > b_2.t$ (i.e., $6 \geq 2$).

Most applications require real-time filtering, where users are interested in complex event patterns that impose additional constraints on the event instances [3]. Such parameterized constraints between event attributes and specific values can be specified in the optional **WHERE** clause [17].

Window-based processing is widely applied in data stream processing systems [4]. Our proposed framework could be extended with window-based functionalities thus to support event pattern queries with sliding windows. Due to space limitation, we skip the discussion on this subject in the paper.

3 Query Satisfiability and Unsatisfiability

3.1 Event Constraint

As discussed earlier, in many practical cases events are generated based on predefined constraint. In this work, we consider an event constraint C which can be expressed using a regular expression. For instance, C can be given as the event workflow of the input stream. $L(C)$ denotes the language defined by C.

For any event trace h, if h is the prefix of a sequence $k \in L(C)$, we call h being consistent with C. *Trace*(C) denotes the set which contains all the event traces that are consistent with C. Thus given a trace h, $h \in$ *Trace*(C) iff $\exists$ sequence k: $hk \in L(C)$. Obviously, $L(C) \subseteq Trace(C)$.

Example 2. Regular expression $A^+K^*B^+KC^+$ represents a given event constraint C_{exp2}, where consecutive A event instances, B event instances and C event instances are divided by a K event instances but the K's between the A's and B's is optional. Event trace $h_1 = \{ a_1, k_2, b_4, b_7, k_8, c_9 \}$ and $h_2 = \{ a_1, b_4, b_7, k_8, c_9 \}$ are both in *Trace*(C_{exp2}).

3.2 Satisfiability and Unsatisfiability Checking

An event trace is said to match event query Q if the evaluation of Q over h produces at least one matched pattern. For a pattern query Q, $L(Q)$ denotes the set which contains all the event traces that match Q.

Example 3. Consider event trace h_1, h_2 given in *Example 2* and event pattern query Q_{exp3} = *SEQ*(A, K, K, C), which looks for event patterns with at least two K event instances appearing between an A instance and a B instance. Trace h_1 matches Q_{exp3} since in the trace there exist a complex event pattern instance $<a_1$, k_2, k_8 $c_9>$ matching the target pattern $<a$, k, k, $c>$. However, trace h_2 does not match Q_{exp3} since no instance of the target pattern could be found.

Given a pattern query Q, an event constraint C and an event trace $h \in$ *Trace*(C), we want to determine whether a query match may exist for h while h keeps evolving at runtime. This problem is regarding the checking of the *query satisfiability / unsatisfiability*, of which we give the formal definitions below.

Query Satisfiability. Given Q, C and $h \in$ *Trace*(C), Q is **satisfiable** iff $\forall$ k: $hk{\in}L(C) \rightarrow hk{\in}L(Q)$. This is denoted as *Satisfiable*(Q, C, h) = *true.*

Query Unsatisfiability. Given Q, C and $h \in$ *Trace*(C), Q is **unsatisfiable** iff $\nexists$ k: $hk{\in}L(C) \wedge hk{\in}L(Q)$. This is denoted as *Unsatisfiable*(Q, C, h) = *true.*

The key functionalities of event stream processing applications rely on the detection of certain event patterns (usually corresponding to the exceptional cases in the application domain). Alerts (such as system notification or triggers) are usually raised after the target pattern has been detected. Under the context of such *situation alerts*, checking the query satisfiability / unsatisfiability is equivalent to determining whether situation alerts will be raised in the future while a given event trace evolves on the fly. Given Q, C and $h \in$ *Trace*(C), Q either could be determined as satisfiable (i.e., *Satisfiable*(Q, C, h) holds true) or unsatisfiable (i.e., *Unsatisfiable*(Q, C, h) holds true), or could not yet be determined (i.e., both *Satisfiable*(Q, C, h) and *Unsatisfiable*(Q, C, h) are false, which means that whether a matched pattern may exist for Q while h evolves could not be decided yet).

Example 4. Consider Q_{exp3}, C_{exp2} given earlier and event trace h_3 = { a_1, k_2 }, h_4 = { a_1, b_4 } and h_5 = { a_1, a_2 }. We have *satisfiable*(Q_{exp3}, C_{exp2}, h_3) as true. This is because C_{exp2} guarantees one or more K instances will appear (i.e., the K's appearing before and after the consecutive B's) and then C events will appear after that. Thus, no matter how h_3 evolves, matched result(s) will surely appear in the future. For instance, a match is formed after h_3 evolving to h_1. Similarly, we knows that *unsatisfiable*(Q_{exp3}, C_{exp2}, h_4) is true, since C_{exp2} indicates that only one K instance (i.e., the K appearing after the consecutive B's) will appear. We could also see that both *satisfiable*(Q_{exp3}, C_{exp2}, h_5) and *unsatisfiable*(Q_{exp3}, C_{exp2}, h_5) are false, since whether a match may exist could not yet be decided at the moment. How to determine satisfiability / unsatisfiability using algorithms for this example will be given later in *Example 6.*

A query could be statically determined as satisfiable / unsatisfiable before receiving any event input, i.e., the event trace h being an empty sequence. These two extreme cases are referred to as *static query satisfiability / unsatisfiability.*

Example 5. Consider C_{exp2} given earlier and Q_{exp5-a} = *SEQ*(A, K, C), Q_{exp5-b} = *SEQ*(A, K, D). Obviously we can guarantee the static query satisfiability of

Q_{exp5-a} because C_{exp2} indicates the existence of instances such as $<a, k, c>$. Similarly, the static unsatisfiability of Q_{exp5-b} is guaranteed.

For an event constraint C, we let τ_C denote the minimized DFA for the language $L(C)$. Similarly, for a pattern query Q, we let τ_Q denote the minimized DFA for the language $L(Q)$. Construction of τ_C and τ_Q is described in [12]. For a given DFA τ, We use $\mathring{s}\tau$ to represent τ's start state. The state transition function of τ used for processing a sequence input is denoted as $\hat{\delta}\tau$. $\hat{\delta}\tau(s, \mathit{seq})$ denotes the state reached after finishing the transition of *seq*, beginning from a given state s in τ. $\hat{\delta}\tau(s, \mathit{seq}) = \emptyset$ if the transition falls out of τ. We use $D\tau(s)$ to denote the derivative of a state s in τ, which is equivalent to the language accepted by τ starting from state s.

We give two theorems as below before showing our algorithm on query satisfiability / unsatisfiability checking, given query Q, constriant C and trace h. We have DFA τ_C, τ_Q defined earlier and we use $\tau_\cap$ to denote the minimized DFA equivalent to $\tau_C \cap \tau_Q$. For efficient algorithms of constructing $\tau_\cap$ and computing automaton derivatives, please refer to [12].

Theorem 1. *Unsatisfiable*(Q, C, h) holds true iff $\hat{\delta}\tau_\cap(\mathring{s}\tau_\cap, h) = \emptyset$.

Theorem 2. Assuming $\hat{\delta}\tau_\cap(\mathring{s}\tau_\cap, h) = p$ and $\hat{\delta}\tau_C(\mathring{s}\tau_C, h) = q$, *Satisfiable*$(Q, C, t)$ holds true iff $D\tau_\cap(p)$ is equivalent to $D\tau_C(q)$.

Due to space limitation, proofs are skipped in this paper. Based on the theorems, we design our query satisfiability / unsatisfiability checking algorithm shown in *Algorithm 1*. *Line 6* to *13* give the static checking process. Q can be statically guaranteed as unsatisfiable if $\tau_\cap$ accepting only empty language (by *Theorem 1*) and Q can be statically guaranteed as satisfiable if $\tau_\cap$ is equivalent to τ_C (by *Theorem 2*). *Line 17* to *47* give the runtime checking process. Once an input event from the evolving trace *seq* triggers a state change in either $\tau_\cap$ or τ_C, derivative of the current state (pre-computed in *Line 17*) of $\tau_\cap$ and τ_C will be compared. If they are equivalent, the satisfiability of Q can be guaranteed (by *Theorem 2*). If the transition falls out of $\tau_\cap$ (note that it could never fall out of τ_C because the input sequence is consistent with C), the unsatisfiability of Q can be guaranteed (by *Theorem 1*).

Example 6. Consider Q_{exp3} and C_{exp2} given earlier. *Figure 2* shows three automaton respectively: (1) $\tau_{C_{exp2}}$, (2) an equivalent NFA of $\tau_{Q_{exp3}}$ (instead of showing $\tau_{Q_{exp3}}$ for easier understanding) and (3) the minimized DFA $\tau_\cap$ equivalent to $\tau_{C_{exp2}} \cap \tau_{Q_{exp3}}$. Let us first look at trace h_3 (given in *Example 4*). When the first event a_1 is processed, both $\tau_{C_{exp2}}$ and $\tau_\cap$ transit to state *s1* and the derivatives for these two states are $A^*K^*B^+KC^+$ and $A^*K^+B^+KC^+$ respectively, which are obviously not equivalent. When the second event k_2 is processed, both $\tau_{C_{exp2}}$ and $\tau_\cap$ transit to state *s2* and the derivatives turn to be equivalent as $K^*B^+KC^+$. Thus we can guarantee the satisfiability of Q_{exp3}. We then look at trace h_4 (given in *Example 4*) which is an example of query unsatisfiability. The transition falls out of $\tau_\cap$ when the second event b_4 is processed and unsatisfiability of Q_{exp3} can be guaranteed at this moment.

Algorithm 1. Query Satisfiability / Unsatisfiability Checking

```
 1: Procedure: $SatUnsatChecking$
 2: Input: (1) constraint $C$, (2) query $Q$, (3) real-time evolving sequence $seq$ (must be consistent
    with $C$) as "$e_1$ $e_2$, $e_3$..." received incrementally, with the End of Stream (EOS) message
    arriving at the very end if input termination is indicated
 3: Output: static or runtime notification of query satisfiability / unsatisfiability
 4:
 5: ——————————————— Static Checking ———————————————
 6: construct $\tau_C$, $\tau_\cap$ and pre-compute $D\tau_\cap(\mathring{s}\tau_\cap)$ and $D\tau_C(\mathring{s}\tau_C)$
 7: if $\tau_\cap$'s accepted language $L(\tau_\cap) = \emptyset$ (i.e., $\tau_\cap$ being an empty automaton) then
 8:     notify unsatisfiable and return
 9: else
10:     if $\tau_C$'s accepted language $L(\tau_C)$ is equivalent to $L(\tau_\cap)$ (i.e., $D\tau_\cap(\mathring{s}\tau_\cap) = D\tau_C(\mathring{s}\tau_C)$)
        then
11:         notify satisfiable and return
12:     end if
13: end if
14: ————————————————————————————————————————
15:
16: ——————————————— Runtime Checking ———————————————
17: calculate the derivatives for all the states $\tau_C$ and $\tau_\cap$ except $\mathring{s}\tau_\cap$ and $\mathring{s}\tau_C$
18: var $p \leftarrow \mathring{s}\tau_\cap$
19: var $q \leftarrow \mathring{s}\tau_C$
20: var $p'$, $q'$
21: var $checkFlag \leftarrow$ false
22: var $e \leftarrow poll(seq)$
23: while $e$ != EOS do
24:     $p' \leftarrow \hat{\delta}\tau_\cap(p, e)$
25:     $q' \leftarrow \hat{\delta}\tau_C(q, e)$
26:     $checkFlag \leftarrow$ false
27:     if $p' = \emptyset$ then
28:         notify unsatisfiable and return
29:     else
30:         if $p$ != $p'$ then
31:             if $D\tau_\cap(p')$ is equivalent to $D\tau_C(q')$ then
32:                 notify satisfiable and return
33:             end if
34:             $checkFlag \leftarrow$ true
35:             $p \leftarrow p'$
36:         end if
37:         if $q$ != $q'$ then
38:             if !$checkFlag$ then
39:                 if $D\tau_\cap(p')$ is equivalent to $D\tau_C(q')$ then
40:                     notify satisfiable and return
41:                 end if
42:             end if
43:             $q \leftarrow q'$
44:         end if
45:     end if
46:     $e \leftarrow poll(seq)$
47: end while
48: ————————————————————————————————————————
```

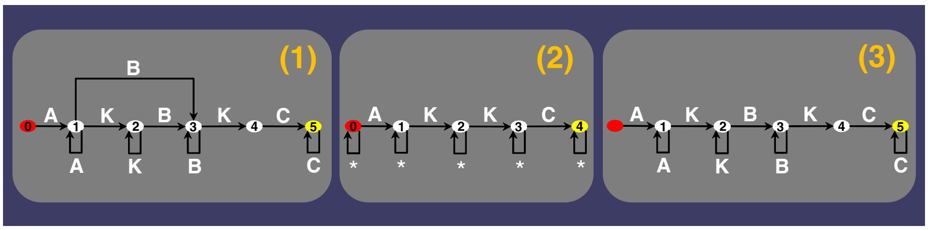


Fig. 2. Example Automaton

3.3 Lightweight Constraint Checking

Checking the derivative equivalency between the states of $\tau_\cap$ and τ_C (polynomial time complexity) introduces runtime costs in *Algorithm 1*. It is conducted every time an input event instance triggers state transition(s) on either $\tau_\cap$ or τ_C, which could bring in big overhead for such runtime process. Besides that, *Algorithm 1* requires two automaton state look-up at runtime for each input event, i.e., the state lookup at $\tau_\cap$ and τ_C.

Below we introduce an optimized query satisfiability / unsatisfiability checking algorithm to decrease the runtime cost in *Algorithm 1*. Let us first look at a theorem given as following, where $\tau_\cap$'s state set is denoted as $S\tau_\cap$ and τ_C's state set is denoted as $S\tau_C$.

Theorem 3. For any p in $S\tau_\cap$, there exists q in $S\tau_C$: for any sequence *seq*, $\hat{\delta}\tau_\cap(\mathring{s}\tau_\cap,\ seq) = p \rightarrow \hat{\delta}\tau_C(\mathring{s}\tau_C,\ seq) = q$.

The proof for *Theorem 3* is skipped due to space limitation. From the theorem we can see that each state in $\tau_\cap$ has one and only one mapping state in τ_C. This fact guarantees the correctness of a runtime constraint checking mechanism given in the following *Algorithm 2*.

Algorithm 2. Lightweight Query Satisfiability / Unsatisfiability Checking

```
1: Procedure: LightweightSatUnsatChecking
2: Input / Output: Same as Algorithm 1 (Line 2 to 3)
3:
4: ———————————————— Static Checking ————————————————
5: Same as Algorithm 1 (Line 6 to 13)
6: ——————————————————————————————————————————————————
7:
8: ———————————————— Runtime Checking ———————————————
9: perform pre-computation by running Algorithm 3 as AutomatonEncoding($\tau_\cap$, $\tau_C$)
10: var $p \leftarrow \mathring{s}\tau_\cap$
11: var $p'$
12: var $e \leftarrow poll(seq)$
13: while $e$ != EOS do
14:     $p' \leftarrow \hat{\delta}\tau_\cap(p,\ e)$
15:     if $p' = \emptyset$ then
16:         notify unsatisfiable and return
17:     else
18:         if $p$ != $p'$ then
19:             if $p.encoding$ = DER_EQUIVALENT then
20:                 notify satisfiable and return
21:             end if
22:             $p \leftarrow p'$
23:         end if
24:     end if
25:     $e \leftarrow poll(seq)$
26: end while
27: ——————————————————————————————————————————————————
```

Different from *Algorithm 1*, *Algorithm 2* achieves lightweightness in constraint checking by applying the *automaton encoding* before the runtime checking process. For each state p in $\tau_\cap$, its mapping state q in τ_C is found and derivative equivalency of p and q is checked. Then the corresponding checking result is encoded with p (*Line 9* in *Algorithm 2*). *Algorithm 3* depicts such automaton encoding process. The process has two components: the *traverser* and the *applier*. Each state in $\tau_\cap$ is associated with a variable *encoding* which is used to record the encoded information. By default the *encoding* value is set to be N/A for each state. The *traverser* traverses $\tau_\cap$ and directs the *applier* to each of its states. For a given state p, the *applier* calculates p's mapping state q in τ_C and performs the derivative comparison between p and q. If these two are equivalent, $p.encoding$ will be encoded as DER_EQUIVALENT. By using such encoded result on $\tau_\cap$, runtime cost in the runtime process is greatly decreased. Runtime checking of the derivative equivalency is completely replaced by a simple checking on the encoding value of the reached state in $\tau_\cap$ (*Line 20* in *Algorithm* 2). The query can be determined to be satisfiable while the encoding value is DER_EQUIVALENT. Besides that, running τ_C alongside with $\tau_\cap$ is no longer needed thus only one automaton look up at $\tau_\cap$ is required for each input event.

Example 7. Consider the same scenario as in *Example 6* but applying the lightweight constraint checking algorithm. The state *s2* to *s5* of $\tau_\cap$ are all encoded as DER_EQUIVALENT after applying *Algorithm* 3. For trace h_3, when the second event instance k_2 is processed, $\tau_\cap$ transits to state *s2*. Thus we are guaranteed the satisfiability of Q_{exp3} at this moment.

Algorithm 3. Automaton Encoding

```
1: Procedure: AutomatonEncoding
2: Input / Output: (1) DFA $\tau_\cap$, (2) DFA $\tau_C$
3: Output: $\tau_\cap$ with encoded information on derivative equivalency checking
4:
5: calculate the derivatives for all the states $\tau_C$ and $\tau_\cap$ except $\mathring{s}\tau_\cap$ and $\mathring{s}\tau_C$
6: for all $p \in S\tau_\cap$ except $\mathring{s}\tau_\cap$ do
7:     find $p$'s mapping state $q$ in $\tau_C$
8:     if $D\tau_\cap(p)$ is equivalent to $D\tau_C(q)$ then
9:         $p.encoding \leftarrow$ DER_EQUIVALENT
10:    end if
11: end for
```

3.4 Handling Predicate-Based Filtering

As earlier discussion, most applications require real-time filtering, where users are interested in complex event patterns that impose additional constraints on the event instances. Our constraint-aware pattern detection framework supports predicate-based filtering on event streams using the same automaton-based mechanism introduced earlier. We skip the formal description in this paper due to space limitation. Instead we simply show an example as following.

Example 8. Consider constraint $C_{exp8} = A^+B^+A^+C^+$, and query $Q_{exp8} =$ *SEQ*(A, B, C) *WHERE* $A.id =$ "3". In order to fit into the automaton-based framework, we rewrite C_{exp8} into $C'_{exp8} = (A[id! = 3]|A[id == 3])^+B^+(A[id! = 3]|A[id == 3])^+C^+$. Minimized DFA $\tau_{C'_{exp8}}$ and $\tau'_\cap$ (equivalent to $\tau_{C'_{exp8}} \cap \tau_{Q_{exp8}}$) are given in *Figure* 3 (1) and (2) respectively. Take trace $h_6 = \{\ a_1[id = 2], b_3\ \}$ and $h_7 = \{\ a_1[id = 3], b_3\ \}$ as example. For h_6, unsatisfiability of Q_{exp8} can be guaranteed. For h_7, satisfiability of Q_{exp8} can be guaranteed instead.

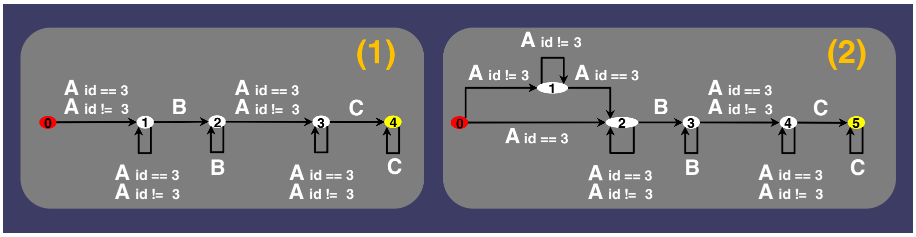


Fig. 3. Automaton for Example Query with Predicate-Based Filtering

4 Query Execution

Pattern monitoring is a long running process for event pattern detection. In an execution strategy without considering constraint knowledge, the monitoring process could be stopped only when the event trace is terminated. Corresponding CPU and buffer resources could not be released earlier. During the monitoring process, situation alert will be raised while target event patterns has been detected. *Algorithm* 4 given below sketches such *basic execution strategy*.

Algorithm 4. Basic Execution Strategy

```
1: Procedure: BasicExecution
2: Input: real-time evolving sequence seq as "$e_1$ $e_2$, $e_3$...", with the End of Stream (EOS)
   message arriving at the very end if input termination is indicated
3: Output: situation alerts and matched result patterns
4:
5: var $e \leftarrow poll(seq)$
6: while $e$ != EOS do
7:     process $e$:
8:     perform necessary data buffering, raise situation alerts and produce results if possible
9:     $e \leftarrow poll(seq)$
10: end while
11: terminate the pattern monitor for the current event trace
```

As earlier discussion, observation of the query satisfiability / unsatisfiability could be utilized in two aspects. First, it enables us to identify queries which are guaranteed to not lead to successful matches at the earliest, thereby helping us to terminate such long running pattern detection processes and release the corresponding CPU and buffer resources earlier. All the buffer taken for this trace can be released and no more CPU and memory footprint is required in the future on this trace. We call this process *early monitor termination*. Second, it enables us to identify queries which can be guaranteed to surely lead to a future alert at the earliest (even though the matched result has not yet happened), thereby helping us to get prepared for upcoming situations. We call this process *early situation alert*.

We thus propose a *constraint-aware execution strategy* for complex event pattern detection over streams in *Algorithm* 5, by which the query satisfiability / unsatisfiability will be notified at the earliest possible moment during the execution to achieve both early monitor termination and early situation alert. The execution strategy follows the *Event Condition Action* (ECA) rule-based framework.

It applies a constraint checker M using the checking algorithm (*Algorithm* 1 or 2) to notify the query satisfiability / unsatisfiability on the fly. Through the ECA framework, the real-time streaming event input serves as the **events**. The checking results from M serve as the **conditions** and the corresponding steps taken based on the checking result are seen as the **actions**.

To be specific, following benefits could be obtained through the early monitor termination and early situation alert by our proposed execution strategy:

1. **Early Buffer Release.** By the early monitor termination, buffered elements can be released earlier.
2. **Further Buffer Avoidance.** By the early monitor termination, no further event buffering is required for the current event trace.
3. **Further Monitor Avoidance.** By the early monitor termination, no further pattern detection process is needed for the trace.
4. **Taking Precaution Action for Upcoming Situations.** By the early situation alert, we can get prepared for upcoming situations at the earliest.

Algorithm 5. Constraint-Aware Execution Strategy

```
1: Procedure: ConstraintAwareExecution
2: Input: (1) real-time evolving sequence seq as "$e_1$ $e_2$, $e_3$...", with the End of Stream (EOS)
   message arriving at the very end if input termination is indicated, (2) procedure $M$ for
   lightweight satisfiability / unsatisfiability monitor given in Algorithm 1 or 2
3: Output: situation alerts, matched result patterns and early situation alerts, with the early
   monitor termination functionality
4:
5: invoke $M$'s static checking process
6: if $M$ raises unsatisfiable then
7:     terminate the pattern monitor determination for the current event trace
8:     return
9: else
10:    if $M$ raises satisfiable then
11:        raise early situation alert
12:    end if
13: end if
14: invoke $M$'s runtime checking process
15: var $e \leftarrow poll(seq)$
16: while $e$ != EOS do
17:    pass $e$ to $M$
18:    if $M$ raises unsatisfiable then
19:        release buffer, perform early monitor determination for the current event trace
20:        pass EOS to $M$
21:        return
22:    else
23:        if $M$ raises satisfiable then
24:            raise early situation alert
25:        end if
26:    end if
27:    process $e$:
28:    perform necessary data buffering, raise situation alerts and produce results if possible
29:    $e \leftarrow poll(seq)$
30: end while
31: terminate the pattern monitor for the current event trace
32: pass EOS to $M$
```

Example 9. Consider the same scenario as in *Example 6*. Let us first look at trace h_3. When the second event k_2 is processed, the constraint checker raises **satisfiable**. Thus, an early situation alert will be thrown at this moment for helping the corresponding parties to get prepared for upcoming situations at the earliest, even though the whole $<a, k, k, c>$ pattern has not yet been formed. For trace h_4, the constraint checker raises **unsatisfiable** when the transition falls out of $\tau_\cap$ at the second event b_4. Thus a_1 (the first event in h_4) which was received and buffered earlier can be purged and no further buffering is required under this trace. Also, the pattern monitor can be terminated at this point in order to release corresponding CPU resources. Consider h_4 evolving to h_2 (given in *Example 2*). Pattern detection and buffering for extracting and keeping k_8 can be avoided through the early monitor termination.

5 Performance Evaluation

5.1 System Implementation

E-Tec is implemented using Java 5. *Figure 4* shows the system architecture. The **Query Plan Generator** parses and translates a given event query into an execution plan, which includes a pre-computed encoding. The **Query Executor** takes in events from input streams and constructs results on the fly. The **Constraint Engine** utilizes automaton-based technique to perform runtime constraint monitoring. The **Execution Controller** receives feedbacks from the constraint engine and triggers the query executor to perform corresponding runtime actions.

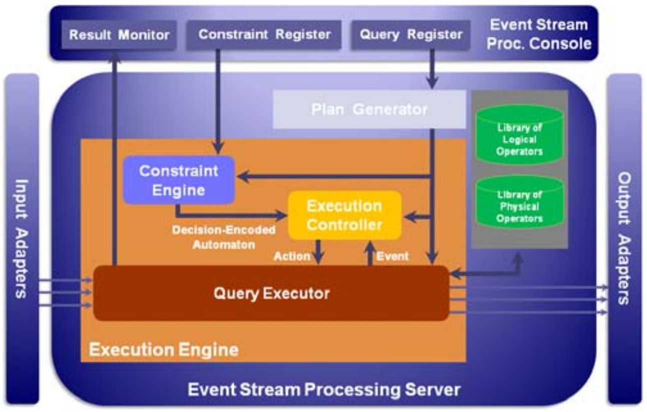


Fig. 4. E-Tec System Architecture

The automaton-based model is commonly used by the state-of-the-art ESP engines. Our provided query satisfiability / unsatisfiability checking framework can be easily integrated with the automaton-based ESP engines by combing the monitoring automaton ($\tau_\cap$) with the automaton applied for pattern detection.

5.2 Experimental Setting

Experiments are run on two Pentium 4 3.0GHz machines, both with 1.98G of RAM. One machine sends the event stream to the second machine, i.e., the query engine. In Section 5.3 and 5.4 we will report the performance of our constraint-aware techniques on a 5G data input based the supply chain data model given in [10], which contains multiple real life use cases on SCM. From its workflow, we can see that the data can be highly irregular, with 60% of the event types that can be optional or exclusive choices (used for constroling query selectivity).

We run two sets of experiments. One is on event pattern queries with only pattern-based filtering, where we vary the pattern-based selectivity, which controls the percentage of patterns being filtered out through the query structure-related factors (Q_{exp3} as an example) from zero to 100% through changing the query complexity (state number of the query automaton in our case). The other set of experiments is on queries with only predicate-based filtering (Q_{exp8} as an example). In this test the pattern-based selectivity is 100%. However we vary the predicate-based selectivity from zero to 100% through changing the predicate type and predicate position. As our future work, we plan to perform experiments on an on-line auction data, which conforms to the schema used in XMark [15].

5.3 Queries with Only Pattern-Based Filtering

Memory Consumption. Our constraint-based pattern detection technique should be able to minimize the amount of data that is buffered: with a smaller selectivity (less results being produced), more unnecessary data buffering could be avoided. The results shown in the right chart of *Figure* 5 provides the verification. X axis shows 6 groups of queries categorized by their pattern-based selectivities. Y axis shows the accumulative memory consumption for each query. We can see that the basic constraint checking (*Algorithm 1*) has the same buffer performance as the lightweight constraint checking (*Algorithm 2*) since they have the same effect on cutting memory consumption.

CPU Performance. The left chart of *Figure* 5 shows the query execution time. We can see that in most cases constraint-aware approaches outperform the

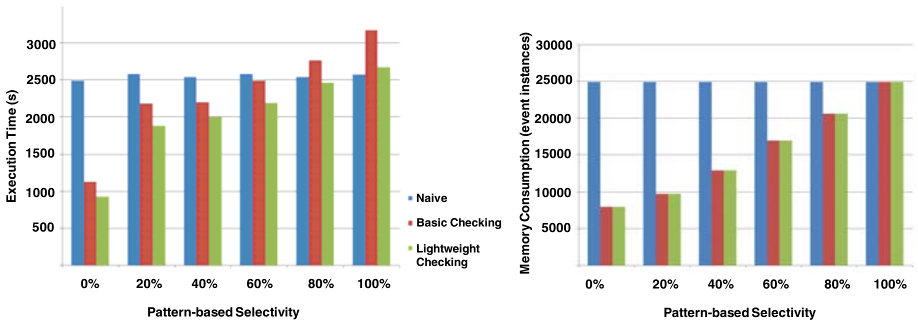


Fig. 5. Exp. Results for Queries with Only Pattern-Based Filtering

naive approach without considering constraints: with a smaller selectivity, more unnecessary CPU computation could be avoided. However, when the selectivity is very high, constraint-aware approaches have poor performance because their overheads on runtime constraint checking become higher than the CPU saving through early monitor termination. Y axis here shows the execution time for each query. In the best case (i.e., the query for which selectivity is 0%), plans optimized with constraint-based processing reduce the execution time of the original plan by 76%. We can also observe that the basic constraint checking does not perform as well as the lightweight constraint checking since its costly runtime process introduces higher overhead.

5.4 Queries with Only Predicate-Based Filtering

Experiments on memory and CPU consumption are also run for queries with only predicate-based filtering. Eexperimental results with similar characteristics as in Section 5.3 are reported, which are shown in *Figure* 6. Due to space limitation, we skip further description for this set of experiments in the paper.

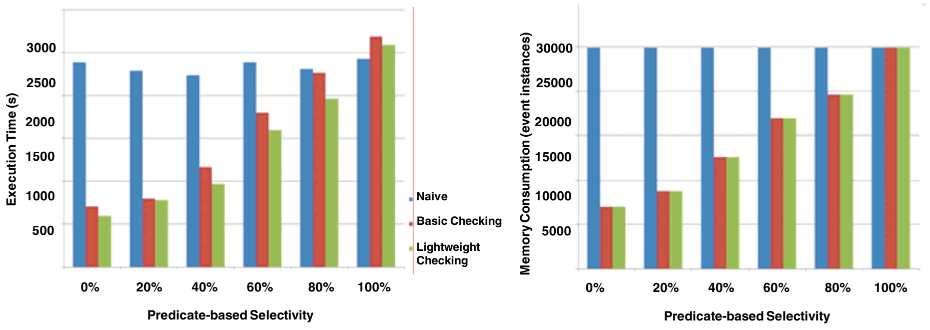


Fig. 6. Exp. Results for Queries with Only Predicate-Based Filtering

5.5 Conclusion of the Experimental Study

Above experimental results reveal that the proposed constraint-aware pattern detection framework is practical in two senses: (1) the technique can surely reduce the system memory consumption; (2) savings on CPU performance brought by the technique can be significant in most cases.

6 Related Work

The constraint-aware query processing has been studied extensively in traditional databases, which does not meet the requirement of event stream processing application because they do not provide real-time solution for event processing. Some work on XML stream processing engines [16][11]have looked at the schema-based optimization opportunity focusing on reducing CPU and memory footprint in XML data processing. Such techniques for handling of semi-structured data cannot be applied in event stream processing which is handling high volume of real-time stream input in the format of heterogenous events. Event-specific ESP technology, which has an event-specific system design and evaluation mechanism, is shown to be superior to generic stream processing solutions [5][2][6] because it is being specifically designed for handling sequence queries over streaming event. An expressive yet easy-to-understand language is proposed in [17] to support pattern queries on such sequential streams and proposes customized algebra operators for the efficient processing of such sequence queries with sliding windows. Constraint knowledge is not within the consideration of its query evaluation. A plan-based technique to perform streaming complex event detection across distributed sources is discussed in [3]. Its focus is mainly on handling pattern detection over event streams in a distributed environment. A constraint-aware ESP solution is provided in [8]. However, it only considers a limited number of event constraint types instead of completely utilizing the whole input constraint. Even though a compile time pre-computation mechanism is given to improve the runtime constraint inferencing, this process still requires multiple state checking for every input event. Besides that, the abductive inference which is required at their compile time pre-computation is NP-complete.

7 Conclusion

In many practical cases business events are generated based on pre-defined business logics. Hence, constraints often hold among event data. For pattern detection over event streams, reasoning using such known constraints enables us to identify the unsatisfiability and the satisfiability for a query at the earliest possible moment, thereby helping us to get prepared for upcoming situations at the earliest. thus helping us to effectively decrease the resource consumption and expedite the system response on certain situation alerts. We propose a framework for constraint-aware pattern detection over event streams. Given the constraint of the input event stream at compile time, the query satisfiability / unsatisfiability is efficiently monitored on the fly using our lightweight runtime checking algorithm. Following an ECA-based query execution strategy, we are able to adjust the processing strategy dynamically, by producing early feedbacks, releasing unnecessary resources and terminating corresponding pattern monitor. We have implemented our proposed framework in the E-Tec prototype system, which is efficiently augmented by our optimization module to use finite automaton for runtime pattern detection and constraint checking. Our experimental studies illustrate that the proposed techniques bring significant performance improvement in both memory and CPU usage with little overhead.

References

1. Gartner Inc., http://www.gartner.com
2. Abadi, D., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.: Aurora: A new model and architecture for data stream management. VLDB Journal 12(2), 120–139 (2003)
3. Akdere, M., Cetintemel, U., Tatbul, N.: Plan-based complex event detection across distributed sources. PVLDB 1(1), 66–77 (2008)
4. Babcock, B., Babu, S., Motwani, R., Widom, J.: Models and issues in data streams. In: PODS, June 2002, pp. 1–16 (2002)
5. Babu, S., Widom, J.: Continuous queries over data streams. In: ACM SIGMOD (September 2001)
6. Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M., Hellerstein, J., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M.: TelegraphCQ: Continuous dataflow processing for an uncertain world. In: CIDR, pp. 269–280 (2003)
7. Demers, A.J., Gehrke, J., Panda, B., Riedewald, M., Sharma, V., White, W.M.: Cayuga: A general purpose event monitoring system. In: CIDR, pp. 412–422 (2007)
8. Ding, L., Chen, S., Rundensteiner, E.A., Tatemura, J., Hsiung, W.-P., Candan, K.S.: Runtime semantic query optimization for event stream processing. In: ICDE, pp. 676–685 (2008)
9. Etzion, O.: Semantic approach to event processing. In: DEBS, p. 139 (2007)
10. Harris, C., Gass, S.: Encyclopedia of MS/OR (2000)
11. Koch, C., Scherzinger, S., Schweikardt, N., Stegmaier, B.: Schema-based scheduling of event processors and buffer minimization for queries on structured data streams. In: VLDB, pp. 228–239 (2004)
12. Kozen, D.: Automata and computability. W. H. Freeman and Company, New York (2003)
13. Li, M., Mani, M., Rundensteiner, E.A., Lin, T.: E-tec: A constraint-aware query engine for pattern detection over event streams. In: ICSC, pp. 565–566 (2009)
14. Liu, M., Li, M., Golovnya, D., Rundensteiner, E.A., Claypool, K.T.: Sequence pattern query processing over out-of-order event streams. In: ICDE, pp. 784–795 (2009)
15. Schmidt, A., Wass, F.: XMark: a benchmark for XML data management. In: VLDB, pp. 974–985 (2002)
16. Su, H., Rundensteiner, E.A., Mani, M.: Semantic Query Optimization for XQuery over XML Streams. In: VLDB, pp. 1293–1296 (2005)
17. Wu, E., Diao, Y., Rizvi, S.: High-performance complex event processing over streams. In: SIGMOD, pp. 407–418 (2006)

Attribute Outlier Detection over Data Streams

Hui Cao[1], Yongluan Zhou[2], Lidan Shou[1], and Gang Chen[1]

[1] College of Computer Science, Zhejiang University, China
[2] Department of Mathematics and Computer science,
University of Southern Denmark, Denmark
chaserzju@gmail.com, zhou@imada.sdu.dk, {should,cg}@cs.zju.edu.cn

Abstract. Outlier detection is widely used in many data stream application, such as network intrusion detection, fraud detection, etc. However, most existing algorithms focused on detecting class outliers and there is little work on detecting attribute outliers, which considers the correlation or relevance among the data items. In this paper we study the problem of detecting attribute outliers within the sliding windows over data streams. An efficient algorithm is proposed to perform exact outlier detection. The algorithm relies on an efficient data structure, which stores only the necessary information and can perform updates incurred by data arrival and expiration with minimum cost. To address the problem of limited memory, we also present an approximate algorithm, which selectively drops data within the current window and at the same time maintains a maximum error bound. Extensive experiments are conducted and the results show that our algorithms are efficient and effective.

Keywords: attribute outlier, date stream.

1 Introduction

Outlier detection has been widely studied in the literature. Hawkins [6] gave a well-known definition to outlier as "an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism". On one hand, outlier detection may be useful for removing noises from massive datasets. On the other hand, outliers detection can also be used in applications looking for abnormal signals, such as network security threats, bank frauds etc.

In general, outliers could be classified into two types, namely *class outliers* and *attribute outliers* [12,16]. Class outliers are those data items that deviate significantly from the rest with a user-given metric, such as distance and density. Attribute outliers are the data items that have some abnormal *attributes* compared to other "similar" tuples. Intuitively, we can firstly partition the whole dataset into several subsets with respect to the similarity of their attributes. Then, the class outliers of each subset can be regarded as attribute outliers. We will illustrate the differences between class outliers and attribute outliers using an example.

Table 1 shows a snippet of two-week stock trading history starting from July 1 2008. According to the definition of conventional class outliers, tuples with ID 2,3,9,10 are obviously class outliers, as they deviate considerably from other tuples in both open/close

Table 1. Trading Information of Stock in Two Weeks

Tuple ID	Ticker	Open($)	Close($)	Volume	Data
1	ACS	53.25	52.84	8992	2008-6-30
2$*$	ZMH	70.66	69.93	16075	2008-6-30
3$*$	XRAY	26.32	26.1	14110	2008-6-30
4	ACS	52.77	53.28	8573	2008-7-1
5	ACS	53.39	53.60	7976	2008-7-2
6	ACS	53.54	53.84	7680	2008-7-3
7	ACS	53.79	53.91	7758	2008-7-4
8	ACS	53.90	54.02	9021	2008-7-7
9$*$	ZMH	72.34	72.18	17893	2008-7-7
10$*$	XRAY	25.12	24.87	13549	2008-7-7
11	ACS	54.02	53.94	9820	2008-7-8
12$\star$	ACS	53.99	54.35	**17028**	2008-7-9
13	ACS	54.40	54.48	9726	2008-7-10
14$\star$	ACS	54.43	54.99	**18920**	2008-7-11

prices and trading volumes. But, in fact, the values of these tuples are normal. For example, tuple 2 and 9 represent the trading information of the stock whose ticker name is ’ZMH’. Although these are the only two ’ZMH’ tuples in the table, their rarity does not mean any abnormality, as it is meaningless to compare trading information across different ticker names. Therefore, tuple 2 and 9 should not be regarded as outliers. The same observation can be made for tuple 3 and 10. In such cases, computing class outliers fails to find the true abnormality, and attribute outlier could be a more useful alternative. In Table 1, tuple 12 and 14, both marked by a $\star$, are the actual abnormal records which may be of interest to the users. The reason is that all tuples in the table with the trading records of ‘ACS’ have similar open/close price and trading volume except tuple 12 and 14. These two tuples are characterized by very high trading volumes, which are nearly twice the average number of the other ’ACS’ tuples. Therefore, tuple 12 and 14 are the true (attribute) outliers in the table.

Many applications require outlier detection over data streams, for example video surveillance, network intrusion detection and so on. Due to the dynamic and unbounded characteristics of data streams, outlier detection over streams is a challenging problem and hence has attracted much attention in recent years [2,13]. However, almost all these studies focused on finding class outliers. The problem of detecting attribute outlier over data streams has not been studied. In comparison with the class outlier detection, attribute outlier detection over data streams has its unique challenges and a new non-trivial solution is needed.

First, one may intuitively propose a two-phase approach, which uses stream clustering algorithms to continuously maintain clusterings on sliding windows and then performs outlier detections within each cluster in each window. However, such an approach is inefficient as the outlier detection has to be executed once for each slide of the window. A more efficient approach should continuously maintain a data structure that stores the necessary information for outlier detection within each cluster. As memory is limited and data is of high arrival rate in typical data stream applications, both the size and the update cost of such a data structure should be kept minimum. Furthermore, the clustering will evolve over time and hence clusters may have to be merged from time to time. The data structure should also support efficient outlier re-calculation incurred by cluster merging.

Second, to keep up with fast data rate, the algorithm should selectively drop some tuples in order to maintain the memory constraints. Such approximation adds extra complexity to the outlier re-calculation incurred by cluster merging. As some tuples may have been dropped, useful neighborhood information may be lost, and the outlier re-calculation becomes less accurate in comparing to the case without tuple shedding. An approximate estimation of the lost information has to be developed.

To address these challenges, we propose a new approach, called AOMA (Attribute Outlier Monitoring Algorithm), to support continuous detection of attribute outliers over data streams. In summary, we have made the following major contributions in this paper:

– We define the problem of attribute outlier detection over data streams and to the best of our knowledge, this is the first paper to study this problem. Different from traditional class outlier detection, we focus on finding outliers by considering the correlation among the data items.
– We propose a new approach to efficiently monitor attribute outliers over data streams. In this approach, data within a window are partitioned into clusters based on their similarity and outliers are detected on each individual cluster. The algorithm continuously maintains a data structure, namely OSQ (Outlier Searching Queue), which stores the necessary information for updating the outliers when window is slided. Efficient updating operations over OSQ are developed.
– We also propose an approximate algorithm, which strikes a balance between the accuracy of the outlier detection and the computing/memory costs. The algorithm selectively drops some "less important" tuples and hence saves the storage cost and computing cost. The problem of precision losing in re-calculation after cluster merging is addressed by using an approximate estimation approach as a partial remedy. We prove that this approach can provide a guarantee on the maximum error bound for the output.
– Finally, we have conducted extensive experimental studies on the proposed algorithms with both real and synthetic datasets. The results verify the efficiency and accuracy of our proposed algorithms.

The rest of the paper is organized as follows. Section 2 reviews existing work on outlier detection and clusterings. Section 3 introduces the background and presents the problem statement. Section 4 and 5 describe the details of the exact and approximate outlier detection algorithms in AOMA. The experimental results and analysis are presented in Section 6. Section 7 concludes this paper.

2 Related Works

The problem of outlier detection has been extensively studied. Below we will review four major categories of outlier.

Traditional statistical methods try to build some best-fit distribution models for the given dataset, and then those objects that do not fit the model well are considered **distribution-based outlier** [3]. However, in many applications, it is hard to find a proper distribution model especially for high dimensional data.

Clustering-based outlier [8] could be regarded as a by-product of clustering. The clustering-based approach partitions the dataset into several clusters, and regards the objects in the tiny clusters as outliers. Although it also involves a clustering procedure, it still detect outliers over the whole dataset, which is different from our attribute outliers. Furthermore, the proposed approach is essentially a clustering algorithm over static dataset and hence cannot address our challenges that are mentioned above.

The method proposed by Breunig et al. [4] first used *Local Outlier Factor* to measure the deviation of objects from their local neighbors and then returned the **density-based outliers** according to its neighborhood information. Although this approach could find the hidden local outliers, it needs to maintain the K-NN neighborhood for each object, which is especially time-consuming in data stream with large amount of update.

Distance-based outlier is first introduced by E.M.Knorr and R.T.Ng [9,10]. An object o in dataset S is a distance-based outlier if at least a fraction p of the objects in S have a distance from object o greater than r. This definition gives a straight-forward and intuitive criterion to find outliers from a dataset. One problem of this approach is that the quality of results depends on suitable parameters p and r. In this paper, we focus on detecting distance-based outliers.

Most existing algorithms mainly focus on detecting outliers over static data, but data stream processing has recently attracted more and more attention. Angiulli and Fassetti [2] proposed a continuous outlier detection algorithm over sliding windows, which can efficiently return the distance-based outliers. To reduce the memory cost, they also introduced an approximate algorithm which can maintains a provable error bound.

Koh et al. [12,11] is perhaps the first one to study attribute outliers detections. Their approach can find outliers in correlated subsets where objects have the same value in several attributes. However, this approach is not suitable for applications where data are not naturally partitioned by exact value matches. In contrast, our method detects attribute outliers with respect to the similarity of attributes instead of exact matches, which could be required by some applications. Moreover, our algorithm is capable of handling streaming data and can produce approximate results when the memory space is limited.

As AOMA has to perform clusterings within each sliding window, our work is also related to data stream clustering [7,14]. In particular, Aggarwal et al. [1] introduce the temporal property into cluster and propose the algorithm CluStream to address stream clustering. In this paper, we extend this approach to handle sliding windows, which is not considered in the original paper [1].

3 Preliminary

In this section, we first introduce the background and then present the formal statement of the problem of monitoring attribute outliers over data streams.

3.1 Background

In this paper, we focus on distance-based outliers. We refer to such outliers as *DB*-outlier (Distance-Based outlier).

Definition 1. *$DB(k,R,D)$-outlier. For the given positive parameters: k and R, an object o in data set D is a $DB(k,R,D)$-outlier if the number of objects which lie within the distance of R from o is less than k.*

In this above definition, the distance between a pair of data tuples is calculated by a user-given function *dist*. Those objects lie within R distance from object o are called neighbors of o.

In our algorithms, before finding attribute outliers, we need partition the data into several clusters. Zhang et.al. [14] introduce the concept of *Cluster Feature Vector*, which uses some statistical information to represent each cluster. We adopt their definitions as follows.

Definition 2. *$CF(\overline{CF1}, CF2, n)$. Given a cluster in a dataset D with d dimensions that contains data tuples $p_1 \cdots p_n$, it can be represented as a $(d+2)$ vector, $CF(\overline{CF1}, CF2, n)$, where n is the number of the data tuples in the cluster, and $\overline{CF1}$ is the linear sum of the n tuples, i.e., $\overline{CF1} = \sum_{i=1}^{n} \overline{p_i}$, and $CF2$ is the square sum of the n tuples, i.e., $CF2 = \sum_{i=1}^{n} (\overline{p_i} \cdot \overline{p_i})$.*

As stated in [14], the cluster feature CF's additivity can help incrementally maintain the synopses of the cluster and merge two different clusters. Consider two cluster with CF, $CF_a(\overline{CF1_a}, CF2_a, n_a)$ and $CF_b(\overline{CF1_b}, CF2_b, n_b)$, then $CF_a + CF_b = (\overline{CF1_a} + \overline{CF1_b}, CF2_a + CF2_b, n_a + n_b)$. Based on the property of CF, the center of the cluster p_c and the radius r are both easily computed from the CF vector. We refer the interested readers to [14] for details.

Clustering over data stream is quite different from static dataset, it is assumed that each data tuple in a data stream is associated with a timestamp which defines its life span. The cluster would also need a temporal property to determine its life-cycle, so that, when all the information maintained in the cluster become obsolete, this cluster and its synopsis CF vector should also be ignored. A temporal extension of CF, called *Temporal Cluster Feature* (TCF) is introduced in [1,15] to address the problem.

Definition 3. *$TCF(\overline{CF1}, CF2, t, n)$. For a cluster C with d- dimensional data tuples $p_1...p_n$ associated with timestamps $t_1...t_n$, TCF is a $(d+3)$-dimensional vector which brings the time factor into consider, $(\overline{CF1}, CF2, t, n)$, t is timestamp of the most recent data tuples in the cluster from data stream, i.e., $t = t_i$ (p_i is the last tuple came into the cluster). The other values of TCF are the same as CF.*

3.2 Problem Statement

Our algorithms aim to continuously monitor attribute outliers over sliding window. Different from [12], which only addresses the problem over static dataset and needs to examine all the subspaces of the dataset which is intolerable in data stream environment, we propose a new cluster-based definition for attribute outliers over data streams, which is **C**luster-based **A**ttribute **O**utlier (CAO).

Definition 4. *$CAO(k,R,C,t)$. Given sliding window with tuples $p_1...p_W$, which are clustered into m clusters $C_1...C_m$, each containing some tuples similar to each other, and a tuple p_i with timestamp t_i that falls into the current window and belongs to the cluster C_j, then p_i is called a $CAO(k,R,C_j,t_i)$ outlier if there are no more than k tuples in C_j lie within distance R from p_i.*

Given the definition of the cluster-based attribute outlier, now we can present the description of the problem addressed in this paper.

Definition 5. *Continuous query of attribute outliers over data stream. Given a sliding window with size W over data stream and the user-defined parameters k and R, continuously return all the $CAO(k,R,C,t)$ outliers in the latest window.*

4 The AOMA Algorithm

4.1 Data Structure

To support data expiration, we propose a data structure, called *Linear Histogram of Cluster Feature* (LHCF). Each *LHCF* for a cluster contains several buckets, and each bucket contains a TCF vector. The *TCF* vectors in each *LHCF* are organized into a queue, which are ordered by their latest timestamp *t*. We assign the fixed size *b* for each bucket, so that each TCF vector will involve the information of *b* data tuples at most. The *LHCF* help keep track of the evolving data stream. If a *TCF*'s timestamp *t* falls outside of the *W* records of sliding window, then this bucket can be removed from the queue. Furthermore, the newly arrived data of the cluster will be added to the last bucket of the queue and the timestamp of the *TCF* will be updated with the timestamp of the new data. Suppose a cluster has *B* buckets, due to the additivity of *CF*, the center and the radius of the cluster can be simply calculated from the *LHCF* vector, i.e., the center $\overline{p_c} = \frac{\sum_{i=1}^{B} \overline{CF1_i}}{\sum_{i=1}^{B} n_i}$ and the radius $r = \sqrt{\frac{\sum_{i=1}^{B} CF2_i}{\sum_{i=1}^{B} n_i} - \overline{p_c} \cdot \overline{p_c}}$; and the *LHCF* structure contains *B* buckets which are $TCF(\overline{CF1_1}, CF2_1, t_1, n_1) \cdots TCF(\overline{CF1_B}, CF2_B, t_B, n_B)$.

Moreover, as described before, to find the attribute outliers, we need to do the outlier detection over the clusters which are online partitioned from the current sliding window. To efficiently maintain the neighborhood information of each data tuple in its cluster, an OSQ (Outlier Search Queue) node will be created for each tuple. An OSQ node consists of the following items: *data*, which denotes the data tuple in the cluster; *time*, the timestamp of the data tuple; *preNL*, an sorted array of the timestamps of the most recent *k* neighbors that precedes the current tuple; *sucNC*, the count of the succeeding neighbors of the current tuple.

Note that, *preNL* only need to keep the timestamps of the most recent *k* preceding neighbors. That is because an outlier may have at most *k* neighbors. Furthermore, this array need not be updated when a preceding tuple is expired. Instead, one can easily count the number of unexpired preceding neighbors by looking up the location of the oldest timestamp that is still within the scope of the current window by using a binary search. For the succeeding neighbor, we only need to maintain a count. That is because the current tuple will not expire before the tuple. If *sucNC* of this tuple exceeds the number k, the data tuple will never become an outlier, in other words, this tuple is a *safe inlier*.

Definition 6. Safe Inlier. *If the sum of the numbers of preceding and succeding neighbors of a tuple exceeds k, then this tuple is an inlier. If its succeeding neighbors exceeds k, then it is a* safe inlier.

Each cluster is associated with a queue of OSQ nodes, one for each tuple in the cluster, which are sorted by their timestamps. This OSQ queue should be continuously updated as the window slides. When a new data tuple p comes into cluster c, a new OSQ node q is created for the tuple. A neighbor search is necessary to create the entry of *prevNL*. Given the user-defined parameter R, the neighbor search query returns all the neighbor nodes which lie within R distance from p. We denote the set of the returned nodes as $q.prev$. The most recent k preceding neighbors are chosen from $q.prev$ and their timestamps are stored in the array *p.preNL* in sorted order. Meanwhile, for each node in $q.prev$, the count *sucNC* is incremented. Moreover, along with the window slides, we will simply remove the expired node from the OSQ queue.

4.2 Cluster Maintenance Module

There are two tasks in this module. First, we have to maintain the clustering information, i.e. $LHCF$, in the current sliding window, so that the clustering could be updated according to the new tuple arrival and old tuple expiration. Necessary re-clustering has to be performed. Second, we have to maintain the neighboring information, i.e. the OSQ queue, within the clusters to facilitate the outlier detection module.

Assume that the numbers of the clusters that should be built in the window is m. In general, m is far smaller than the size of sliding window W. The clusters could be denoted as $C_1...C_m$. Each cluster is associated with a $LHCF$ vector and an OSQ (Outlier Search Queue), which is a list of all the points in the cluster with additional information about neighborhood. OSQ will be useful in outlier query module.

Initially, with the W tuples in the current window, we could use the k-means algorithm to create m clusters, and then instantiate the $LHCF$ vector and generate the OSQ queue of each cluster.

Whenever a new data tuple p arrives at time t, we should update clusters for the effect contributed by the new tuple. Every new tuple need to be assigned to a cluster. First we should find a nearest cluster C to absorb this tuple. The distance between cluster C and p, denoted as $dist(p, C)$, can be calculated as the distance between the data tuple p and the centroid of C. As described in the last section, both the center and radius can be easily calculated from $LHCF$ of the cluster.

As mentioned in [1], p does not naturally belong to the closest cluster. p may be still too far away from C in related to the radius r of C. So each cluster has a maximum boundary which can be denoted as $\lambda \cdot r (\lambda >= 1)$. The maximum boundary of a cluster containing only one tuple is defined as the distance from that tuple to the another cluster that is closest to it. We use the maximum boundary to decide whether the cluster should absorbed the tuple. If $dist(p, C) < \lambda \cdot r$, p will be assigned to C. Otherwise, a new cluster containing only p will to be created.

But when there are already m clusters, we should reduce the number of the clusters to avoid exceeding the limited total number of clusters. Then two closest clusters are chosen to merge. With the additivity of $LHCF$, the merging of two $LHCF$ is easy to perform. Furthermore, we have to merge the two OSQ queues, which will be handled as follows. First, the two OSQ queues has to be merged to maintain its sorted order on timestamps. Then for each node, we need to find the those neighbors that are in the other merging cluster in order to update the array *preNL* and the count *sucNC*. Moreover, as the number of neighbors of a node after the merge can only become lager, a safe inlier will still be safe in the new merged queue. Therefore, the neighborhood of safe inliners is not updated to save the merging cost.

At last the algorithm also should eliminate the expired data tuples. At time t, all the OSQ nodes with timestamps older than $t - W + 1$ and the TCF older than $t - W + 1$ from the cluster should be discarded. And when there is no more TCF in the $LHCF$ of the cluster, this cluster should also be removed.

4.3 Outlier Query Module

As the above module maintains the necessary information, the actual outlier detection procedure is very simple and induce minimum cost. More specifically, the outlier query module uses the OSQ queue of each cluster to find the outliers. The OSQ queue of each cluster is scanned to get the count of the neighbors of each data tuple in the cluster. For each OSQ node, a simple binary search within $O(\log k)$ time over the array *preNL* can return the oldest non-expired preceding neighbor in the current sliding window. Since our array stores the OSQ nodes' timestamps in sorted order, the count of non-expired preceding neighbors can be derived by the array index of the oldest non-expired preceding neighbor. We denote this count as *count_pre*. Then we could get the sum of neighbors by adding up the two count *sucNC* and *count_pre*. If the sum is less than k, then we can report it as an attribute outlier to the user.

Note that there may exist some tiny clusters, which have fewer than k data tuples. Such tiny clusters are not considered in outlier detection. Otherwise, all their tuples would be outliers, which is meaningless in practice.

5 The Approximate AOMA Algorithm

The algorithm introduced above needs to store all the data tuples in the current sliding window. However, in a streaming environment, the memory space is often limited in comparing to the fast data rate and storing all the data tuples might be prohibited. In this section, we will improve the algorithm and give the approximate AOMA that can monitor the attribute outliers under limited memory space with a provable error bound.

To reduce the memory cost of the algorithm, it is necessary to shed some data tuples to keep the memory consumption under the system restriction. The central problem of data shedding is to decide which data to drop. There are three kinds of tuples in each cluster: outliers, safe inliers and unsafe inliers. Outliers surely cannot be dropped since they are the results to return. For the unsafe inliers, they may become outliers when some of their preceding neighbors expire when the window slides. Therefore, both outliers and unsafe inliers are preferred to be kept in the memory to make sure that the algorithm can get a high recall of the outliers. In another word, safe inliers are preferred to be shed. As mentioned in the last section, safe inliers will never be outliers until they expire and hence we can be sure that dropping them will not drop the real outliers. Another more important reason to drop safe inliers is that, as shown in the previous studies [1], they are usually the majority of the whole data set in the current window. Hence, we focus on the shedding of safe inliners in this paper, which is the most critical problem.

Although the safe inliers can never be outliers, the algorithm cannot drop all of them. This is because they may be neighbors of the other data tuples and dropping too many safe inliers would introduce inaccuracy to the calculation of the neighborhood of the incoming data. Given a cluster c with n data tuples, we start to drop safe inliers when their number exceeds $\mu n (1 > \mu > 0)$, where μ is determined by the available memory size and the number of data tuples in a window. Those dropped safe inliers are chosen randomly within all the safe inliners.

With the dropping of tuples, the attribute *preNL* of an OSQ node can no longer accurately estimate the number of preceding neighbors, as some of them may have been dropped before *preNL* is generated. Furthermore, *preNL* is memory consuming. Therefore, we replace the attribute *preNL* with two new attributes that have smaller sizes. The first one is *pid*, which is an sequential id of the node. The *pid* of a new OSQ node is assigned as the *pid* of its previous node incremented by 1. In addition, we maintain an attribute of the whole OSQ queue, namely pid_{exp}, which is the *pid* of latest expired node. The second new attribute is *preRatio*, which is the ratio between the number of the node's preceding neighbors that are safe inliers and the total amount of safe inliers in the current OSQ queue. This ratio is calculated when the node is created.

With the two new attributes, the number of preceding neighbors can be easily estimated as follows. For a cluster with n data tuples and a given node q in this cluster, by assuming that the neighbors of each node are uniformly distributed over their arrival time, we can estimate the number of preceding neighbors of node q as:

$$N_p \approx N/n \cdot (q.pid - pid_{exp})$$

where N is the number of neighbors that are not dropped when this node first arrives. As the neighbors of the node is uniformly distributed, N/n is an estimation of density of the node's neighbors in the queue, and $q.pid - pid_{exp}$ is the number of the preceding nodes of q in the current window.

As most data nodes are safe inliers, we can use the number of safe inliers to approximate to the number of data nodes. With the definition of the attribute *preRatio* of an OSQ node, we can estimate N/n as follows:

$$N/n \approx q.preRatio$$

Hence, the number of preceding neighbors of q at time t can be calculated solely from its own attributes:

$$N_p \approx q.preRatio \cdot (q.pid - pid_{exp})$$

Algorithm 1. The Approximate AOMA Algorithm

```
Require: m initial clusters created by k-means algorithm, $C_1...C_m$. Current time t. User-given
    parameter R,k, $\mu$. The number of data tuples in cluster c is $n_c$.
Ensure: All the attribute outliers in the sliding window at time $t$.
 1: $safe\_inlies = 0, pid\_inc = 0$.
 2: for each newcomer data tuple$p$ do
 3:     Call cluster maintenance module update cluster info.
 4:     $safe\_pre = 0$.
 5:     Create new OSQ node $q$ for $p$ and append to the queue of its cluster.
 6:     Scan OSQ and return all nodes lie within $R$ from $q$.
 7:     for each returned node $q_{pre}$ do
 8:         if $++q_{pre}.sucNC > k$ then
 9:             $safe\_inlies++, safe\_pre++$.
10:             Mark $q_{pre}$ as safe inlier.
11:             if $safe\_inliers > \mu n_c$ then
12:                 Randomly choose a safe inlier to remove .
13:     $q.preRatio = safe\_pre/safe\_inlies$
14: for each cluster $c$ in current window do
15:     for each node $q$ in the OSQ queue of $c$ do
16:         if $q.sucNC < k$ then
17:             $count\_pre = q.preRatio \cdot (q.pid - pid_{exp})$
18:             if $count\_pre + q.sucNC <= k$ then
19:                 return the node $q$ as a CAO regard to $c$.
```

Given the estimated number of preceding neighbors, the total number of neighbors can be computed by adding up T_t and $q.sucNC$. Algorithm 1 shows the approximate AOMA algorithm.

As mentioned earlier, two clusters may need to be merged when the number of clusters exceeds the given value. Unlike the exact algorithm, which can re-calculate the various attributes of an OSQ node accurately, the approximate algorithm has to approximately estimate the attributes values of an OSQ node, such as the values of *sucNC* and *preRatio*.

The merging process is illustrated by the following example. Suppose two clusters, c_1 and c_2, have to be merged and we need to create a new OSQ node q in the new cluster for a particular node from c_1, it is necessary to calculate the preceding and succeeding neighbors of this node in both two cluster c_1 and c_2. Assume that this node's preceding and succeeding neighbors in cluster c_1 and c_2 are N_{p1}, N_{p2}, $N_{s1}, andN_{s2}$ respectively. Then, according to the definition of the OSQ node, N_{p1} and N_{s1} could be derived as:

$$N_{p1} \approx q.preRatio \cdot (q.pid - c_1.pid_{exp})$$
$$N_{s1} = q.sucNC$$

To compute the neighbors of q in cluster c_2 need to retrieve the OSQ queue of c_2. The first step is to decide which position q could stay in the queue, it can be simply obtained since the OSQ queue is is maintained by temporal order. Next we will examine the distance between q and all the nodes in the queue. Then the count of preceding neighbors and succeeding ones of q in c_2 could be evaluated, which are denoted as M_p

and M_s. However, these two count can not be recognized as precise count of neighbors of the node q, since the approximate approach may throw many nodes in the queue. Hence, we need to re-estimate the two counts approximately from existing information. Suppose in OSQ queue of c_2 the temporal preceding OSQ node next to q is q_p, and the succeeding one is q_s. We also use q_l to represent the latest node in c_2, the count of all temporal preceding OSQ node of q is denoted by n_p. At last, we will get following equation:

$$N_{p2} \approx \frac{M_p}{n_p} \cdot (q_p.pid - c_2.pid_{exp})$$
$$N_{s2} \approx \frac{M_s}{c_2.n - n_p} \cdot (q_l.pid - q_s.pid)$$

Finally, we get the count of the neighbors of the node q in both two clusters. For the new merged cluster c', a new OSQ node q' will be created for q, *preRatio* and *sucNC* will be estimated as follows

$$q'.preRatio = \frac{N_{p1} + N_{p2}}{q'.pid - c'.pid_{exp}}$$
$$q'.sucNC = N_{s1} + N_{s2}$$

While the approximate algorithm can significantly reduce the memory cost, it introduces inaccuracy into the estimated number of preceding neighbors of a particular tuple. Fortunately, we can prove that such an error has a statistical upper bound as given by the following theorem.

Theorem 1. *For a given tuple that is in a cluster with size n, suppose the estimated number and the actual number of its preceding neighbors are N' and N respectively. The probability that $|N'-N| \leq \epsilon$ is larger than or equal to δ ($0 < \delta < 1$), i.e.* $\Pr[|N'-N| \leqslant \epsilon] \geqslant \delta$, *where*

$$\epsilon = n \cdot \sqrt{\frac{1}{4\mu n}\left(\phi^{-1}\left(\frac{1+\delta}{2}\right)\right)^2}$$

and $\phi(x)$ is the cumulative probability function of the standard normal distribution $N(0,1)$, *i.e.* $\phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{x^2}{2}} dx (-\infty < x < \infty)$.

The proof of the above theorem is omitted due to space limit. It is mainly based to the De Moivre-Laplace theorem and the detail proof is available in the full version of this paper[5].

6 Experimental Evaluation

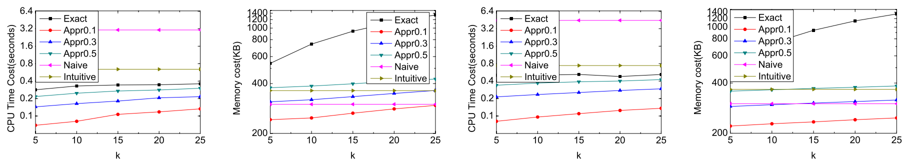
6.1 Experiment Environment and Data Sets

In the experiments we use both synthetic and real-world datasets to generate data streams. The synthetic datasets generated to fit in 10 different clusters, and the points in each cluster follow the Gaussion distribution. Besides, we also experiment on the KDD-CUP'98 Charitable Donation dataset, which has been used to evaluate many data mining algorithms.

All experiments are conducted on a Laptop with Intel Core Duo 2.2 GHz and 2 GB memory, running Microsoft Windows 7. For comparison, we propose two simpler algorithms. Since there have not been any existing approaches to solve our problem, these two methods will be used as the base case of the efficiency and accuracy. The first algorithm is implemented by a nest loop method to find outliers in each cluster of a window, called **N**aive **A**ttribute **O**utlise **M**onitoring algorithm(NAOM). The second algorithm improves the naive one by incrementally maintaining the count of neighbors of the data tuples. When a new tuple arrives or old tuple expires, we update the neighbor counts of the relevant tuples. Such approach avoids the recalculation of the whole neighborhood among all the data. We call it as **I**ntuitive **A**ttribute **O**utlise **M**onitoring algorithm(IAOM). All the algorithms are implemented with C.

6.2 Experiment Analysis

In the experiments, the following parameters are set based on our experiments to ensure a suitable number of outliers are returned and their values do not affect the conclusion of the performance study: the default number of clusters is fixed to be 10, which results in clusters with moderate sizes; the neighbor distance R for the real and synthetic data set are set to 75 and 100 respectively; the value of the absorb coefficient λ is fixed to 2. Furthermore, we employ the time-based sliding window model and, at each slide of the window, there are 500 new data arrive and the same number of data expire. In the following, we refer to NAOM algorithm as "Naive", IAOM algorithm as "Intuitive", the exact AOMA algorithm as "Exact", and the approximate AOMA algorithm with parameter $\mu = 0.1$ as "Appr0.1", and so on.



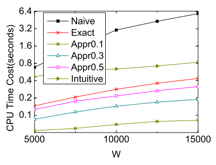
(a) CPU time with KDD-CUP'98 dataset
(b) Memory cost with KDD-CUP'98 dataset
(c) CPU time with Synthetic dataset
(d) Memory cost with Synthetic dataset

Fig. 1. Performance with outlier judge parameter k

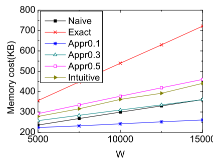
Sensitivity to the outlier threshold k. Within the AOMA algorithms, the value of k decides the number of neighbors need to find for each tuple in order to see if it is an outlier. Hence, the query performance would be affected by k. In this experiment, we fix the size of window to contain 10000 tuples.

Figure 1(a) and 1(c) report the average CPU time spend to update a new window with different k values with both two datasets. While both Naive and Intuitive perform much worse than the AOMA algorithms, it is insensitive to the value of k. That is because they will always check the neighborhood among all the data in the cluster and hence their performance is independent on the value of k. Furthermore, as k increases, the CPU costs of both the exact and approximate algorithms increases slowly. As k should generally be very small, the AOMA algorithms can significantly reduce the CPU consumption in comparing to Naive and Intuitive.

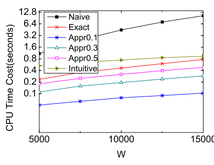
Figure 1(b) and 1(d) illustrate the memory consumption with various k values. We can see that Naive and Intuitive have lower memory consumption than the exact AOMA algorithm and hence they might be suitable for the situation with abundant CPU resources and moderate memory space. On the contrary the exact AOMA is suitable for the case with moderate CPU resources and abundant memory space. Finally, the approximate AOMA algorithm can address both problem of limited CPU and memory space with a suitable μ parameter.



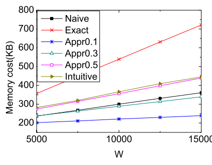
(a) CPU time with KDD-CUP'98 dataset



(b) Memory cost with KDD-CUP'98 dataset



(c) CPU time with Synthetic dataset



(d) Memory cost with Synthetic dataset

Fig. 2. Performance with sliding window size W

Sensitivity to window size W. In this experiment, the examine the scalability of the algorithms, we vary the number of tuples in each sliding window size by varying the timespan of the sliding window. Figures 2(a) and 2(d) show the CPU cost of different algorithms. Naive algorithm performs the worst and its cost grows geometrically as the W increases. On the contrary, the costs of all the other algorithms only grow linear growth with window sizes. Furthermore, the AOMA algorithm perform more than 2 times faster than Intuitive. Finally, the approximate AOMA algorithm can further reduce the CPU cost. For example Appr0.1, it is nearly 4 times faster than the exact algorithm.

Figures 2(b) and 2(d) depict the results about the memory cost. As we did not include the space occupied by the data points into consider, the naive algorithm use little memory since it does not rely on additional data structure. The exact algorithm grows linearly as W increases and approximate approach grows much slower. Similar to the above experiment, the AOMA algorithm, especially with approximation, can be scaled to different window size according to the available memory and CPU resources.

Sensitivity to the number dimension d. In this experiment, we present the CPU cost with different numbers of dimensions. The number of dimensions of the synthetic dataset is varied form 2 to 10 dimensions. Figure 3 shows the experimental result. It can be observed that the AOMA algorithms are much faster than Naive and Intuive. For example, for the dataset with 5 dimensions, Naive spends 4 sec to perform an update of

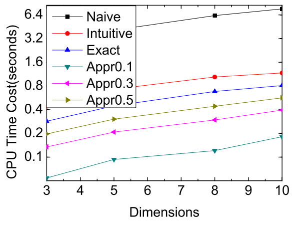


Fig. 3. CPU time with dimensionality

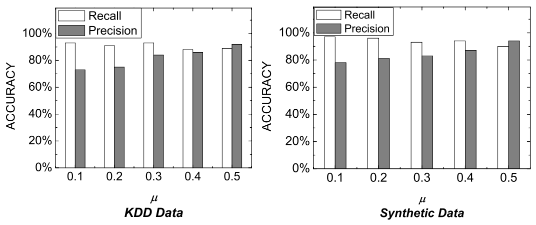


Fig. 4. The accuracy of approximate AOMA

a new window while AOMA only spends 0.44 sec, almost 10 times faster. The approximate algorithm can further reduces the processing time, which is only 0.09 sec when $\mu = 0.1$.

Accuracy analysis. We report the accuracy experiment on both the KDD and synthetic datasets. For comparison, the results reported by the exact AOMA algorithm will be regarded as the true outliers. In this experiment, the window size W and outlier threshold k are set as follows: $W = 10000, k = 5$. We evaluate the approximate algorithm with regard to the exact algorithm and we use two widely used metrics, namely *precision* and *recall*, to measure the accuracy of the approximate results. Suppose the number of the true positive points, false positive points and false negative points in the results returned by the approximate algorithm are tp, fp and fn respectively. Then *precision* and *recall* can be calculated as: $precision = \frac{tp}{tp + fp}, recall = \frac{tp}{tp + fn}$.

Figure 4 presents the results. We can find that both experiments have similar results. While it is obvious that a lower μ value will produce a lower precision, it keeps a high value (70%) even when $\mu = 0.1$. This means the approximate approach only reports a small number of false positive results. From Figure 4 we can also observe that the recall is high, which exceeds 85% for all the tested μ values. This means the approximate algorithm can return most of the outliers.

7 Conclusion

In this paper, we introduce the problem of finding attribute outliers over data streams. Attribute outlier detection differs from traditional outlier detection in a way that the targeted mining set is the data subsets that are generated by partitioning the whole dataset based on the data similarity. We propose a cluster-based algorithm to perform continuous window-based attribute outliers detection, called AOMA. AOMA partitions the current sliding window into several clusters, then uses distance-based approach to find those attribute outliers. Furthermore, to address the problem with limited memory space, an approximate technique is employed to improve the AOMA algorithm. By dropping the safe outliers, it significantly reduces memory consumption and largely speeds up the query processing. We prove that the approximate algorithm can ensure a statistical error bound on the query results. Finally, extensive experimental results show that the AOMA algorithm is scalable and robust to various parameter settings and produces results with high precision and recall.

Acknowledgement

This work was supported in part by the National Science Foundation of China (NSFC Grant No. 60803003, 60970124) and by Chang-Jiang Scholars and Innovative Research Grant (IRT0652) at Zhejiang University. The second author is partially supported by the Danish Council for Independent Research for Natural Sciences.

References

1. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: VLDB 2003: Proceedings of the 29th international conference on Very large data bases, pp. 81–92. VLDB Endowment (2003)
2. Angiulli, F., Fassetti, F.: Detecting distance-based outliers in streams of data. In: CIKM 2007: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, pp. 811–820. ACM, New York (2007)
3. Barnett, V., Lewis, T.: Outliers in statistical data (1984)
4. Breunig, M.M., Kriegel, H.-P., Ng, R.T., Sander, J.: Lof: identifying density-based local outliers. SIGMOD Rec. 29(2), 93–104 (2000)
5. Cao, H., Zhou, Y., Shou, L., Chen, G.: Attribute outlier detection over data streams, 9 (2009), http://db.zju.edu.cn/wiki/index.php/Hui_Cao
6. Hawkins, D.: Identification of outliers. Chapman and Hall, Reading (1980)
7. Guha, S., Meyerson, A., Mishra, N., Motwani, R., O'Callaghan, L.: Clustering data streams: Theory and practice. IEEE Trans. Knowl. Data Eng. 15(3), 515–528 (2003)
8. Jiang, M.-F., Tseng, S.-S., Su, C.-M.: Two-phase clustering process for outliers detection. Pattern Recognition Letters 22(6/7), 691–700 (2001)
9. Knorr, E.M., Ng, R.T.: A unified notion of outliers: Properties and computation. In: KDD, pp. 219–222 (1997)
10. Knorr, E.M., Ng, R.T.: Algorithms for mining distance-based outliers in large datasets. In: VLDB 1998: Proceedings of the 24th International Conference on Very Large Data Bases, pp. 392–403. Morgan Kaufmann Publishers Inc., San Francisco (1998)
11. Koh, J.L.Y., Lee, M.-L., Hsu, W., Ang, W.T.: Correlation-based attribute outlier detection in XML. In: ICDE 2008: Proceedings of the 24th International Conference on Data Engineering, pp. 1522–1524 (2008)
12. Koh, J.L.Y., Lee, M.-L., Hsu, W., Lam, K.-T.: Correlation-based detection of attribute outliers. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 164–175. Springer, Heidelberg (2007)
13. Zhang, J., Gao, Q., Wang, H.: Spot: A system for detecting projected outliers from high-dimensional data streams. In: ICDE 2008: Proceedings of the 24th International Conference on Data Engineering, pp. 1628–1631. IEEE, Los Alamitos (2008)
14. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: an efficient data clustering method for very large databases. In: SIGMOD 1996: Proceedings of the 1996 ACM SIGMOD international conference on Management of data, pp. 103–114. ACM, New York (1996)
15. Zhou, A., Cao, F., Qian, W., Jin, C.: Tracking clusters in evolving data streams over sliding windows. Knowl. Inf. Syst. 15(2), 181–214 (2008)
16. Zhu, X., Wu, X.: Class noise vs. attribute noise: A quantitative study. Artif. Intell. Rev. 22(3), 177–210 (2004)

ISIS: A New Approach for Efficient Similarity Search in Sparse Databases

Bin Cui[1], Jiakui Zhao[2], and Gao Cong[3]

[1] Department of Computer Science & Key Laboratory of High Confidence Software Technologies (Ministry of Education), Peking University
bin.cui@pku.edu.cn
[2] China Electric Power Research Institute, China
jkzhao@pku.edu.cn
[3] Aalborg University, Denmark
gaocong@cs.aau.dk

Abstract. High-dimensional sparse data is prevalent in many real-life applications. In this paper, we propose a novel index structure for accelerating similarity search in high-dimensional sparse databases, named ISIS, which stands for *I*ndexing *S*parse databases using *I*nverted file*S*. ISIS clusters a dataset and converts the original high-dimensional space into a new space where each dimension represents a cluster; furthermore, the key values in the new space are used by Inverted-files indexes. We also propose an extension of ISIS, named ISIS$^+$, which partitions the data space into lower dimensional subspaces and clusters the data within each subspace. Extensive experimental study demonstrates the superiority of our approaches in high-dimensional sparse databases.

1 Introduction

Recently, sparse data with high-dimensionality arises in a variety of database applications, such as e-commerce [2,4] and bioinformatics. In these applications, the sparse dataset has two main characteristics: 1) High dimensionality. The dimensionality of feature vectors can be very high, i.e. the number of all possible attributes is huge (up to thousands). For example, in some e-commerce databases, each participant may declare their own idiosyncratic attributes for products and work orders, which can result in datasets that have thousands of attributes [2]. 2) Sparsity. Each data object may have only a small subset of the attributes, called *active dimensions*. In addition, different entities may have different active dimensions. For example, some e-commerce datasets may have thousands of dimensions, but most of them are null and only a few of them apply to a particular product. The high-dimensional sparse data poses significant challenges to existing high-dimensional indexing techniques [5], and a direct application of any existing technique may suffer from heavy computational and disk I/O cost due to the aforementioned characteristics.

In this paper, we propose a novel index structure to support efficient similarity search over high-dimensional sparse datasets which contain at least hundreds of dimensions. The index structure is named *ISIS*, which stands for *I*ndexing *S*parse databases using *I*nverted file*S*. ISIS is motivated by the following observations. First, due to dimensionality curse, index schemes may need to access the whole dataset to answer a query, and thus reducing the index size is essential for improving the query performance. Second, efficient filter-and-refine approach is preferred for reducing the computational cost, as it can avoid the computing of the distance over the whole dataset. Third, since each data object may have only a small number of active dimensions, we can apply the inverted-files[11]: the first layer is a B^+-tree which is very efficient in terms of space; moreover, it can avoid expensive distance computation during the filtering stage.

The basic idea of ISIS is to convert the data into a new space using clustering techniques. Each cluster corresponds to a dimension in the new space, and the cluster which contains the data is the active dimension of certain object in the transformed space. We embed distance information into the indexing keys of inverted-files. Therefore, we only access inverted-files to locate the clusters containing the query point to generate answer candidates, and refine the results by examining the candidates. Note that while the inverted-files structure can be applied directly by indexing the active dimensions in the original space, it may not be efficient since all the objects that share any active dimension with the query object need to be searched, which may result in expensive disk I/O and computational cost.

To achieve better performance by exploiting the property of sparse data, we propose $ISIS^+$, an extension of ISIS. In $ISIS^+$, a new mechanism is designed to split the data space into multiple low dimensional subspaces. We then vertically partition the dataset and cluster the data in each subspace, where each cluster in a subspace corresponds a dimension in the new space. All the clusters contained the object, i.e. the active dimensions in the new space, are used as indexable terms for $ISIS^+$. With this enhancement, we can represent the original data more precisely and provide better query performance.

Distinguished from previous approaches, our new indexing scheme has the some advantages: 1) Compactness. ISIS adopts a code representation instead of storing the actual feature values, and thus can significantly reduce the storage cost and effectively decrease the number of disk accesses. 2) Effectiveness. In ISIS, similar objects are clustered and share the same key in inverted-files. In the filter stage, we are able to only access the objects which have very high probability of belonging to the result, thus improve the pruning effectiveness. 3) Efficiency. ISIS gets the candidate list based on code comparison instead of expensive distance computation.

We conducted extensive experiments to evaluate our proposed indexes, and compare them against some existing indexes. The results show that our proposed approaches can handle similarity search, such as KNN queries, more efficient over sparse datasets. The ISIS scheme provides a substantial performance improvement for similarity queries in high-dimensional sparse databases.

The remainder of this paper is organized as follows. In the next section, we review related work. In Section 3, we introduce our ISIS schemes, including index structure construction and query algorithms. Section 4 reports the experimental results, followed by our conclusion in Section 5.

2 Related Work

Many indexing techniques have been proposed for accelerating similarity search in high-dimensional databases [5,12,3]. However, with the increase of dimensionality, the indexing techniques fail to outperform sequential scan [14] due to the well known "dimensionality curse" problem. To tackle this phenomenon, recent proposals adopt one of the three approaches, i.e. dimensionality reduction, one dimensional transformation, and data approximation. The dimensionality reduction approaches [6,8] map the original high-dimensional space into a low-dimensional space which can be indexed efficiently using existing indexing techniques. However, dimensionality reduction incurs information loss, and may not be effective to find important dimensions in high-dimensional space [1]. The one dimensional transformation approaches, e.g. the iDistance [15] suffer, however, from the fact that any meaningful search operation involves assessing distances between the full high-dimensional representation of the data points; thus, pruning during search becomes problematic as the dimensionality increases. Additionally, mapping to single-dimensional space results in high information loss. Data approximation methods, such as VA-file [14], represent the original data points by much smaller vectors. It can reduce disk accesses compared with sequential scan. However, it will not reduce computational cost as it need to access approximation of each record and compute the approximate distance respectively.

An alternative way to deal with dimensionality curse is to conduct approximate similarity search for applications where trading a small percentage of recall for faster response is acceptable [10,9]. In [10], the authors presented a new similarity-search method, named Clindex. In Clindex, the dataset is first partitioned into "similar" clusters. To improve the I/O performance, each cluster is then stored in a sequential file, and a mapping table is built for indexing the clusters. The Clindex splits the whole data space into cells, and uses a bottom-up approach that groups objects adjacent in the space into a cluster. When the dimensionality is high, it incurs a huge number of cells, which make the use of mapping table and clustering impractical. More recently, the LDC method [9] was proposed, which exploits bit representation for each dimension. Pruning during LDC KNN search is performed by dynamically selecting a subset of the bitmap based on which subsequent comparisons are performed. In [13], an R-tree based structure, the xS-tree, was proposed for similarity search in very high-dimensional sparse databases. Unlike the R-tree which uses rectangles as bounding regions, the xS-tree uses xSquares. An xSquare is a cross product of high-dimensional squares. In order to guarantee a reasonable minimum fan-out, lossy compression is applied to xSquares.

3 The New Approach: ISIS

In this section, we present a new index structure, named ISIS, to facilitate efficient similarity search over high-dimensional sparse datasets. The ISIS structure deploys the inverted-files structure [11] for similarity search in sparse databases. The inverted-files structure applies the multiple layer architecture. The first tier of the index is a B$^+$-tree which is used to index the dimension $NO.$, i.e. each dimension represents a key. In the leaf level, each dimension has an inverted list. Within the list, each item has two fields: one is the identifier (ID) of the object whose active dimensions include the dimension; the other is the key value which is used to calculate the similarity between the query object and the data object.

The similarity queries on the inverted-files are more like ranked queries than boolean queries, in that similarity score is calculated between the query object and each data object. Processing a similarity query then consists of the following steps. First, an array of accumulators, one per object, is initialized to zero. Then, for each active dimension of the query object, the corresponding inverted list is fetched and processed. Processing a list consists of stepping through it, and, for each object ID, retrieving the value of the active dimension, and adding this similarity contribution to the appropriate accumulator. Finally, when all the active dimensions have been processed, the accumulator values are sorted and the top-K objects are returned.

The method of directly adapting inverted-files to support similarity queries in sparse databases suffers from two limitations. First, unlike simple boolean queries, to support similarity search, distance computation is essential. Second, while keeping the array for the whole dataset is not necessary, the size for array of accumulators could be huge as we have to store the objects which even have only one same active dimension with the query object.

3.1 Dimension Transformation

In this section, we introduce our solution to address above two limitations. To deal with the first limitation, we reserve distance information in the indexed items, and use simple comparison operation to get the candidate answers for filtering. The basic principle behind it is dimension transformation which converts the raw sparse dataset into a new space, where the same active dimension of two objects can represent certain similarity between them. Even with this enhancement, the second limitation remains: the inverted-files structure has to examine a long array. Thus, our solution also tries to reduce the number of active dimensions for sparse data objects.

Our approach for dimension transformation is to apply clustering, i.e. organize the individual objects into clusters. Clearly, the objects in the same cluster have higher potential to be close to each other. We employ the K-means clustering scheme [7] to generate k clusters for the objects in the original high-dimensional space. We generate new data space of k dimensions using the k clusters as follows: (a) let the number of dimensions be the number of clusters, and cluster i corresponds to dimension i, (b) the ith dimension for an object is active only if the object falls into cluster i.

Table 1. Example of sparse dataset

	D1	D2	D3	D4	D5	D6	D7	D8
Object 1	3	1	⊥	⊥	⊥	⊥	⊥	⊥
Object 2	1	3	⊥	⊥	⊥	⊥	⊥	⊥
Object 3	3	1	2	⊥	⊥	⊥	⊥	⊥
Object 4	1	1	⊥	3	⊥	⊥	⊥	⊥
Object 5	⊥	⊥	1	4	⊥	⊥	⊥	⊥
Object 6	⊥	1	⊥	4	⊥	⊥	⊥	⊥
Object 7	⊥	⊥	⊥	⊥	2	2	⊥	⊥
Object 8	⊥	⊥	⊥	⊥	1	3	⊥	⊥
Object 9	⊥	⊥	⊥	⊥	⊥	⊥	1	1
Object 10	⊥	⊥	⊥	⊥	⊥	⊥	2	3

A simple example is given to show the effect of dimension transformation in Table 1. The dataset contains 10 8-dimensional objects. Suppose after clustering, three clusters are generated: cluster 1 contains objects (1, 2, 3), cluster 2 contains objects (4, 5, 6), and cluster 3 contains objects (7, 8, 9, 10). We will have a 3-dimensional new space, where object 1 can be represented by [1, 0, 0] since it falls into cluster 1. For a query point [1, 2, 0, 0, 0, 0, 0, 0], we also convert it into the new space, and its new representation is [1,0,0] (it can have more than 1 active dimension). Therefore, we only need to check the objects (1, 2, 3) which are contained by the same cluster. Compared with the original inverted-files scheme, we have to check 5 objects, i.e. 1, 2, 3, 4 and 6, as they all have the same active dimensions with the query object.

The number of the clusters is predefined as an optimization parameter. Each cluster will correspond a dimension in the new space. The larger is this number, the larger the required storage. On the other hand, a small number will not provide sufficient discrimination. Note that the new space is used for indexing and query processing.

3.2 Algorithms for ISIS

We first present the algorithm for generating the ISIS index structure in the new space generated by clustering, and then present the algorithm for query processing. Fig. 1 shows the algorithm for constructing the ISIS index. We first generate k clusters for the dataset and keep the information of each cluster, i.e. $(ID_i, Center_i, radius_i)$ (Line 1). The original space is converted into a new space: if the object is nearest to the center of a certain cluster, the cluster ID is the active dimension for the object in the new space (Lines 3-4). Finally, index based on the inverted-file structure is constructed (Line 5).

```
Algorithm Construction()
Input: The sparse dataset, the number of clusters $k$
Output: The ISIS structure
1. generate $k$ clusters for the dataset, and record $(ID_i, Center_i, radius_i)$ for each cluster $i$;
2. for each object
3.     find the cluster whose center is nearest to the object;
4.     assign the cluster ID to the object;
5. generate Inverted-files for the new dataset;
```

Fig. 1. The algorithm for ISIS construction

The ISIS index reduces the dimensions of the original dataset while reserving distance information between data objects. Given a query, we apply ISIS index to filter the distant objects, and focus on promising candidates to refine query results. To accomplish this process, the query is first converted into the new space where the ith dimension is set active if the distance between the query and $Center_i$ is smaller than $radius_i$. The new key values are then used for searching in the inverted-files. Any data objects which are far away from the query object can be eliminated and the remainder is returned as the candidates which will be checked.

The similarity search includes both range queries and KNN queries. Since similarity range search can be treated as a window search with a fixed radius and is simpler than KNN search in terms of algorithm complexity, in this study, we concentrate on the KNN

```
Algorithm Search()
Input: Query $q$, the ISIS structure
Output: KNN answer
1. convert the query $q$ into the new space;
2. for each active dimension of $q$
3.        get a list;
4.        merge with the existing list;
5.        sort the list;
6. access the dataset for exact distance evaluation;
7. return the answers;
```

Fig. 2. The search algorithm of ISIS

queries only. To facilitate KNN search, we employ the filter-and-refine strategy in the ISIS scheme. The algorithm is outlined in Fig. 2. First, the query is transformed into the new space: 1) for each cluster ID_i we check the distance between the query q and $Center_i$; 2) if the distance is smaller than $radius_i$, we set the corresponding dimension i active. Note that, in index construction, one object has a single active dimension, while the query object can have multiple active dimensions as clusters may overlap. This is to find all the clusters containing the query object to reduce false negatives. If no cluster contains the query object, we select the nearest cluster. After the transformation, we access the inverted-files for each active dimension of the query object (lines 2-5). After obtaining a list of candidates, they will be merged with the existing lists which are from other previously examined active dimensions. In addition, the lists are sorted according to the number of active dimensions meeting the query. Note that, one same active dimension means that the query and data fall into the same cluster in the original space. Therefore, the object that shares a large number of active dimensions has a higher probability to be near to the query object. This helps us to generate candidates of high quality. Finally, we access the raw dataset and compute the exact distance to refine the query answers (lines 6-7).

3.3 An Enhanced Approach

ISIS index may have the following two problems: First, its effectiveness depends on how well a dataset is globally clustered, i.e., effective clusters can be generated as a transformed space. For real datasets that are typically not globally clustered, more clusters may have to be searched. This can lead to more expensive query processing. Second, in high-dimensional space, a query object may belong to multiple clusters. In the worst case, the complete data space has to be examined for a query.

On the other hand, in such high-dimensional feature spaces, most clustering algorithms would break down in terms of efficiency and accuracy because usually many features are irrelevant or correlated. In addition, different subgroups of features may be irrelevant or correlated according to varying subgroups of data objects. Thus, global dimensionality reduction techniques such as PCA cannot be applied to such datasets to improve the effectiveness of clustering, because they cannot account for local trends in the datasets.

Since the high-dimensional sparse data has a special property, i.e. each data instance may have only a small subset of attributes (active dimensions), this motivates us to cluster the sparse data differently in varying lower dimensional subspaces.

The Subspace Determination. A sparse object typically has only tens of active dimensions, and similar objects tend to share active dimensions. For example, in recommendation systems and target marketing, it is important to find homogeneous groups of users with similar ratings in subsets of the attributes. In addition, it is interesting to find groups of users with correlated affinities, which can help companies to predict customer behavior and thus develop future marketing plans. Therefore, a better way is to find the subspaces which are shared by similar data groups, and generate the clusters within the subspaces, and the approach is named ISIS$^+$. Compared with ISIS, ISIS$^+$ has two advantages. First, it generates more precise clusters in lower dimensional space to represent sparse data. Second, there are less overlapping between clusters, which may lead efficient filtering effects.

In ISIS$^+$, we find correlated dimensions and divide the original space into non-overlapping subspaces. We propose an efficient and effective method to find correlated dimensions, which represent the subspaces in sparse databases. Suppose that the sparse data is D-dimensional, and has N tuples, we generate a D*D table to represent the relation of inter-dimensions of the raw dataset.

Definition 1. **Relation table**: The relation table represents the correlation of dimensions in a sparse database, which is a D*D table. An entry R[i,j], where i and j are in [1,...,D], counts for the times that dimensions i and j appear as active dimension concurrently in the sparse dataset.

Table 2. The relation table of sample dataset

	D1	D2	D3	D4	D5	D6	D7	D8
D1	4	4	1	1	0	0	0	0
D2	4	5	1	2	0	0	0	0
D3	1	1	2	1	0	0	0	0
D4	1	2	1	3	0	0	0	0
D5	0	0	0	0	2	2	0	0
D6	0	0	0	0	2	2	0	0
D7	0	0	0	0	0	0	2	2
D8	0	0	0	0	0	0	2	2

Given the dataset in Table 1, we can generate the relation table as shown in Table 2. For example, R[1][1] equals to 4, which means that dimension 1 is active in four tuples; R[1][3] equals to 1, which means that dimensions 1 and 3 appear as active dimensions concurrently once. To generate the relation table of the sparse dataset, we first initialize the table R, where each entry is set to 0 from the beginning. After that, the sparse dataset is scanned and each object can be processed one by one. For each tuple, we convert it into an array of length D. In this array, the value of a certain entry is 1 if the corresponding dimension is active; otherwise, the value is 0. With this array, we can accumulate the information into the relation table R. For each active dimension i, we access the row i of relation table, scan the array, and increase R[i][j] by 1 if dimensions i and j are both active. The algorithm is very efficient since the dataset is scanned only once and no distance computation is involved.

According to the information presented in this table, the full high-dimensional sparse data space can be partitioned into several subspaces. A perfect subspace partition should enjoy two properties, i.e. all the dimensions are correlated intra-subspace, while unrelated inter-subspaces. However, due to the distribution of sparse data and the restriction on the number of subspaces, perfect subspaces typically do not exist. Given the number of clusters, our algorithm is to find the near-optimal subspaces in an efficient way for sparse databases, where the dimensions within a subspace are more correlated. The correlation is defined as follows.

Definition 2. **Correlated degree** CD: the CD defines the correlation between two dimensions in a sparse database, $CD[i,j] = \frac{R[i,j]}{max(R[i,i],R[j,j])}$.

Algorithm Gen_subspace()

Input: Relation table, number of subspace K

Output: Subspaces

```
1. while there exists any unclassified dimension
2.      find a dimension $d$ with the highest value;
3.      generate a new subspace $s$;
4.      for each correlated unclassified dimension $j$
5.          if $CD[j,d]$ is large than $CD[j,d']$;
        /* where $d'$ is any unrelated unclassified dimension.*/
6.              include $j$ into the subspace;
7. while the number of subspaces is less than K;
8.      find a subspace $S'$ with the highest dimensionality;
9.      find a dimension $d''$ is least correlated with $k\_d$;
10.     generate a new subspace $S''$;
11.     for each dimension in $S'$
12.         if it is more correlated to $k\_d$ of $S''$ than $S'$
13.             move to $S''$;
14. while the number of subspaces is less than K;
15.     check the relation between all $k\_d$s;
16.     merge the most correlated subspaces;
```

Fig. 3. The algorithm to generate subspaces

The algorithm for subspace generation is shown in Fig. 3. If there are dimensions unclassified, i.e. not included in any subspace, we pick the dimension d with the highest value, which is the most active dimension left in the raw dataset. When there is a tie, we select the dimension with the smallest order. We generate a new subspace s, and set d as the key dimension k_d of s. In lines 4-6, we examine all correlated unclassified dimensions. If a dimension j is more correlated to d than any unrelated unclassified dimension, j is added into subspace s. If the number of subspaces equals to the user defined parameter K, the subspace generation is done; otherwise, we need to either split subspaces (lines 7-13) or merge the subspaces (lines 14-16). To split subspaces, a subspace with highest dimensionality is found firstly, because the clustering in a high-dimensional space is inefficient and ineffective. By examining the CDs with the key dimension d', the dimension d'' with the lowest correlated degree is picked up, and a new subspace S'' with key dimension d'' is generated. For all the rest dimensions in S', if it is more correlated to d'' than d', we move it to the new subspace S''. To merge the subspaces, we find the most correlated subspaces, and combine two subspaces into a single space. If there is a tie, we merge the subspaces with the lowest dimensions.

Given the dataset as shown in Table 1, we are able to partition the original 8-dimensional space into multiple subspaces. At the beginning, we select D2 as the key dimension of subspace 1, because D2 has the maximum value 5, and add D1, D3, and D4 into the subspace. After that we can use the same strategy to generate another two subspaces, i.e. [D5, D6] and [D7, D8]. Thus in the first stage of the algorithm, we generate 3 subspaces. If the user sets the number of the subspaces as 4, we need to select a subspace to split. Subspace 1 is picked, and D3 is selected as the key dimension for subspace 4. Now we can compare the difference between full dimensionality and subspaces. For simplicity, we do not further cluster the data in each subspace. As in section 3.1, we partition the data into 3 subspaces, i.e. the transformed new space is 3-dimensional, where for example object 1 can be represented by [1, 0, 0]. If the query point is [0, 0, 0, 0, 1, 2, 0, 0], the representation is [0, 1, 0] in transformed space. Therefore, we only need to retrieve objects 7 and 8 as candidates for distance computation. Compared with the global clustering scheme, this query has to access 4 objects, i.e. 7, 8, 9 and 10. It is worth to note that since the sparse data typically are very high-dimensional, and only a small percentage of dimensions are active, the pruning effect can be more significant. Additionally, the clustering within the lower dimensional subspaces can further discriminate similar/nonsimilar objects efficiently.

Algorithms on ISIS$^+$. Now, we can present the ISIS$^+$ algorithms with minor revision from those of ISIS. To construct an ISIS$^+$ index structure, we first split the whole space into multiple subspaces. The subspace generation mechanism is as described in section 3.3. After that, the data is partitioned into clusters in each subspace. The sum of all clusters is the dimensionality of the new space. Thus each cluster is defined by (ID, subspace, cluster center, radius), and data is converted from the original space into new space. The query algorithm of ISIS$^+$ follows that of ISIS except that we consider subspace information.

4 Experimental Study

In this section, we present an experimental study to evaluate the proposed ISIS schemes for similarity query in sparse databases. The experiments have been conducted on a computer with P4 CPU (2.5GHz), 512 MB RAM, and running Windows XP Professional Operating System. To study performance characteristics over a wide range of search function, we generate the similar dataset using the synthetic data generator [2]. This allows us to vary the following parameters: the dimensionality of sparse database, the cardinality, percentage of active dimensions (degree of sparsity), and the number of distinct values in each dimension. Table 3 shows the parameters of data and experiments, and the values in the bracket are the default values.

The experimental study includes two parts. The first experimental study investigates the index behavior under tuning. We test the performance on two variants for ISIS scheme: ISIS globally clusters the raw sparse data; and ISIS$^+$ partitions the original space into multiple subspaces and generates clusters afterward. Second, we compare them with some existing techniques for KNN search. We use four referenced techniques for the comparison: VA-file [14], inverted-files, LDC [9], and xS-tree [13].

Table 3. Experiment Parameters

Degree of Sparsity	1% - 10% (5%)
Datasize	100K - 500K (100K)
Dimensionality	500 - 1000 (1000)
Number of clusters	5 - 100 (50)
Number of subspaces	1 - 50 (30)
Disk page size	4K bytes

The performance of schemes is measured by efficiency and approximation quality. In terms of efficiency, we examine the average disk access and response time for 10-NN search over 100 different queries. Another performance metric is approximation quality, as ISIS approach may introduce false dismissals, e.g. an exact KNN may not fall in any same cluster with the query point. Computing the number of false dismissals, is enough to capture the traditional error metric, which we will refer to as the $ratio\ of\ false\ dismissals$ (RFD). Let NN_i, where $i \in [1, K]$, be the ith nearest neighbor (NN) in the accurate result set, ANN_i be the ith NN in the approximate result set, Q be the query point, and $Distance\_K$ be the Kth nearest neighbor distance of accurate result. The $ratio\ of\ false\ dismissals$ can be defined as follows:

$$RFD = \frac{1}{K}\sum_{1}^{K}\begin{cases}1\ distance(ANN_i, Q) > \text{Distance\_K}\\ 0\ otherwise\end{cases} \tag{1}$$

Furthermore, we also employ a metric which takes into account the quality of the answers with respect to closeness to the query. We refer to it as the $ratio\ of\ distance\ errors$ (RDE) which can be defined as follows:

$$RDE = 1 - \frac{\sum_1^K distance(NN_i, Q)}{\sum_1^K distance(ANN_i, Q)} \tag{2}$$

4.1 Parameter Tuning

We conduct an extensive study to tune the proposed schemes for optimality, specifically on two main parameters, i.e. number of subspaces (for ISIS$^+$) and clusters (for both ISIS variants). We set the rest parameters as default values in the experiments.

Effect on Subspace. In the first experiment, we vary the number of subspace from 1 to 50. Note that, the ISIS$^+$ is exactly same as ISIS when the number of subspaces equals to 1. Fig. 4 shows the KNN search performance of our new structures for different numbers of subspaces. When the number of subspaces is small (< 20), the dimensionality of subspaces is high, the effectiveness of cluster in each subspace is worse. Because the expected distance between any two objects in high-dimensional space is large, and the cluster radius could be very large as well, thus residing in same cluster does not mean two objects are neighbors. As a result, more clusters will have to be accessed. We observe that as the number of subspaces increases, the number of disk I/O decreases. The performance becomes optimal when number of subspaces is around 30. However, as the

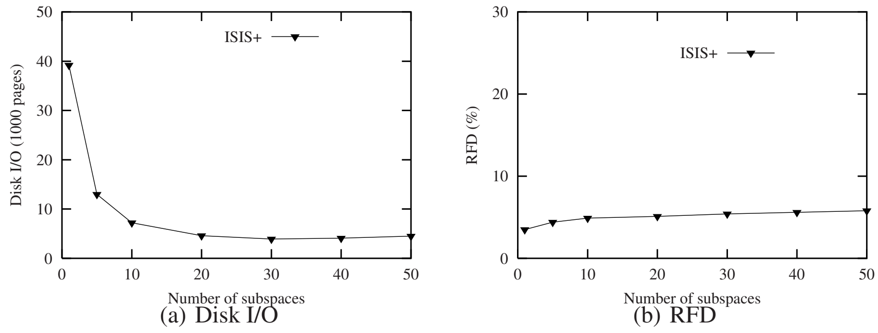


Fig. 4. Performance on different numbers of subspaces

number of subspaces reaches beyond a certain point (> 30), the performance starts to degenerate gradually again. This is because too large a number of subspaces results in that more objects may split into multiple subspaces, i.e. we have to access more entries in the ISIS$^+$. Therefore, the total disk I/O increases accordingly. The optimal number of subspaces (around 30) is a compromise of these factors.

For the approximation quality, we find that the RFD remains quite stable in all the cases, which only increases from 4% to 6% with respect to the number of subspaces from 1 to 50. Although it incurs much more disk accesses for a small number of subspaces, the improvement on approximation quality is not significant, less than 2%. As we mentioned previously, the clustering is not effective for high-dimensional data due to a small number of subspaces. Overall, the large number of subspaces is preferred as it needs much less disk I/O, but provides a satisfactory approximation quality.

Effect on Clustering. In this experiment, we vary the numbers of clusters from 5 to 100. Fig. 5 shows the KNN search performance of our new structures for different numbers of clusters. The performance of Sequential Scan is shown as a baseline.

We observe that as the number of clusters increases, the number of disk accesses decreases. When the number of clusters is small, the cluster radius can be large. Although the query point may only fall in fewer clusters, the space covered by the clusters could be large. In other words, since each cluster contains more data points, the number of disk access is large as the ISIS scheme has to examine these clusters. On the contrary, when the number of clusters is large, the covered space of a single cluster is smaller.

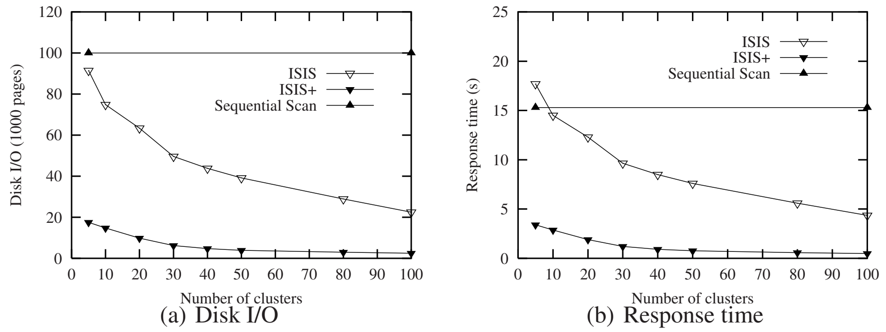


Fig. 5. Performance on different numbers of clusters

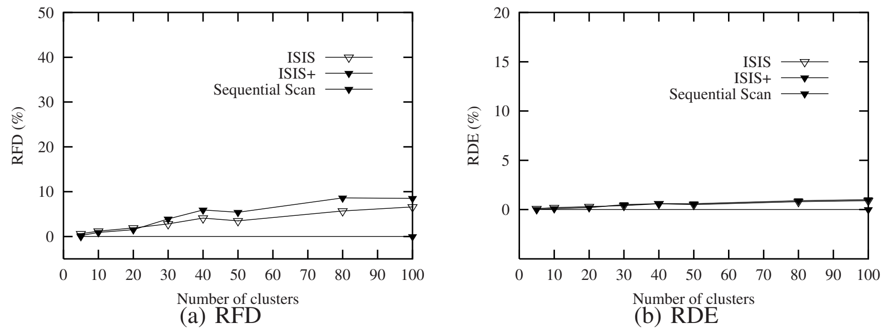


Fig. 6. Performance on approximation quality

Although the query point can be contained in more clusters, the overall space is less. Take 50 as the number of clusters, the dimensionalities of transformed space are 50 and 1500 for ISIS and ISIS$^+$ respectively, as we select 30 subspaces for ISIS$^+$ approach. Note that the dimensionality here represents the number of clusters generated. Compared with ISIS, the ISIS$^+$ is more promising as it requires much fewer disk accesses. Due to the high dimensionality of sparse data, the global clusters of ISIS cannot capture the local information of near neighbors effectively.

Fig. 6 shows the approximation quality by varying the numbers of clusters, i.e. RFD and RDE. For Sequential Scan, both metrics equal to 0 as it can return all exact KNN. The approximation quality of the ISIS and ISIS$^+$ degrades as the number of clusters increases, e.g. the RFD increases from 0.5% to 8%, and RDE from 0.1% to 1%. The reason is that when the query point is near the margin of a certain cluster, some near points may not be in any cluster which contains the query point. The ISIS shows a bit better performance than ISIS$^+$ with five times less disk and time cost. Overall, the enhanced ISIS$^+$ scheme is more optimal, as the query cost is much smaller than ISIS. Additionally, the approximation quality is almost as good as the ISIS, with RFD less than 5% and RDE less than 0.4% when the number of clusters equals to 50.

Since ISIS$^+$ performs better than ISIS, we shall restrict our discussion to ISIS$^+$ in the following experiments, and use the optimal parameters determined above, i.e. we set the number of subspaces as 30 and the number of clusters as 50.

4.2 Comparison with Other Methods

In this section, we compare the ISIS$^+$ with some existing methods over different datasets, such as the VA-file [14], inverted-files, xS-tree [13] and LDC [9]. In the inverted-files, we index all the active dimensions of objects in the original data space. To simplify the comparison, we set default parameter for other approaches, e.g. in VA-file, each dimension is uniformly represented by 5 bits.

Effect on Degree of Sparsity. In many applications, sparse data comes often with various densities. In the first comparison, we evaluate the performance over datasets with various sparse degrees from 1% to 10%, where the degree represents the percentage of active dimensions, and the results are shown in Fig. 7.

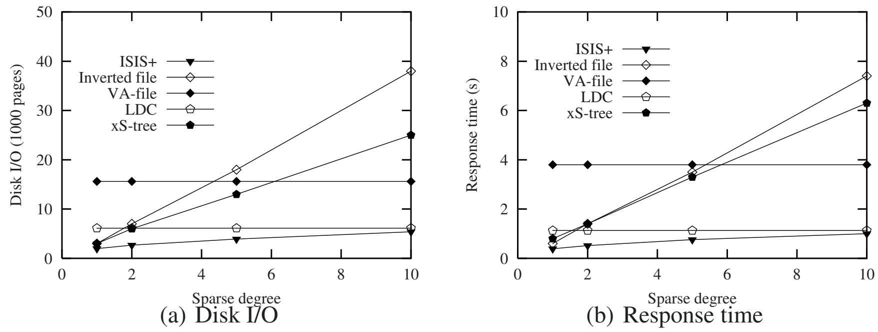


Fig. 7. Performance on different degrees of sparsity

With the increase of degree of sparsity, the VA-file and LDC schemes yield similar performance on Disk I/O. The VA-file fixes the bit representation, which is 5 bits for each dimension. The LDC uses 1 bit for each dimension and retrieves top 3% candidates for exact distance evaluation. The inverted-files yields worst performance and incurs more disk accesses for large degree of sparsity, because it has to retrieve all the objects which has same active dimension with the query point. The xS-tree performs better than the inverted-files, but worse than other approaches. Because it is R-tree based, and uses MBRs in the index nodes which significantly reduces the fan-out of the tree and introduces heavy overlaps due to the high dimensionality. Clearly the proposed ISIS$^+$ provides the best performance for all the datasets. We partition the original high-dimensional space into multiple lower dimensional subspace, thus the clusters within the subspace can capture the local information of the similar objects efficiently. By only accessing the objects in the clusters which contain the query object, the ISIS$^+$ can prune most far-away objects, and effectively reduce the disk accesses.

The results about response time in Fig. 7 (b) show similar performance change tendency compared with disk access, because the main cost of query processing is disk access and computational cost, and typically we need to calculate the similarity on majority of the data retrieved from disk. We found two main differences compared with Fig. 7 (a): First, the gap is widened between VA-file and ISIS$^+$, as we need to compute distance for every data approximate of VA-file. The full dimensional distance computation is very expensive in a very high dimensional space. Second, the gap between

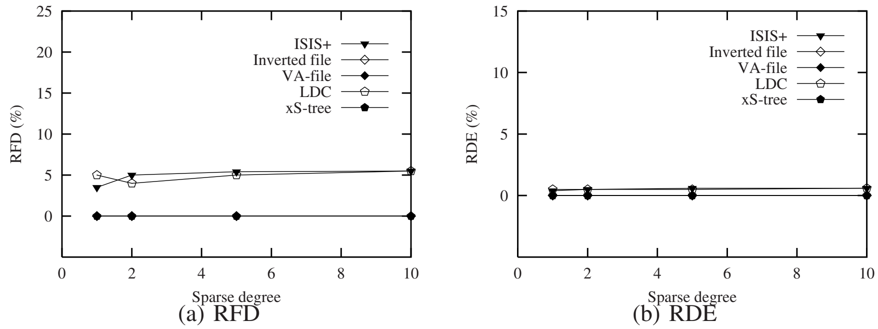


Fig. 8. Approximation quality on different degrees of sparsity

inverted-files and xS-tree is narrowed, since we need to access compressed MBRs in the xS-tree which introduces more overhead costs. As the performance of query response time is almost proportional to that of disk access, we will only show the results of disk access in the following experiments.

Fig. 8 shows the performance on approximation quality. In terms of quality, the VA-file, inverted-files and xS-tree can get 100% accurate answers, as they get all the potential answers for exact distance calculation. The performances of the LDC and ISIS$^+$ are comparable, which are around 5% for RFD, and less than 1% for RDE. Both schemes introduce some false negatives due to the information loss during the dimension quantization and transformation. The ISIS$^+$ is promising as the average disk cost is only around 20% of these three methods. Furthermore, since our technique can permit the immediate generation of results with high quality, this advantage enables the ISIS$^+$ to support online query answering: an important facility for interactive querying and data analysis.

Effect on Dimensionality. In this experiment, we evaluate the various schemes by varying the dimensionality of the sparse data. Fig. 9 shows the disk cost for different dimensionalities from 500 to 1000. The precisions of these methods are similar to the above experiments, and we omit them due to the space constraints.

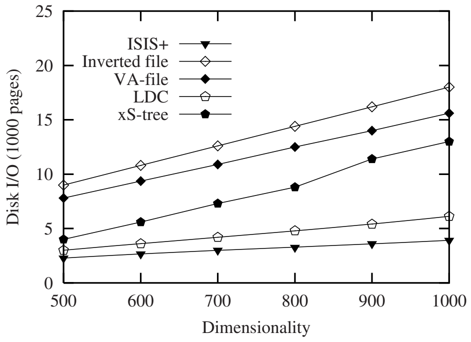


Fig. 9. Performance on different dimensionalities

As the dimensionality increases, the performances of all methods degrade. The index sizes of VA-file and LDC increase proportionally to the dimensionality, as they use 5 bits and 1 bit to represent each dimension. The MBRs of xS-tree also need more space due to the increase of dimensionality. For the inverted-files, since we fix the degree of sparsity in this experiment, the query has to access more inverted list to get all the candidates. ISIS$^+$ still performs better than other methods, i.e. 40% better than LDC which is best among four competitors. The degeneration of ISIS$^+$ is due to more active dimensions, as it has to access the clusters in more subspaces.

5 Conclusion

In this paper, we have addressed the problem of similarity query in high-dimensional sparse databases. We presented an efficient and novel indexing technique for KNN search, called ISIS, which integrates the distance information into the inverted-files.

We also proposed an extension, called ISIS$^+$, which employs subspace clustering, dimension transformation, and inverted-files indexing. With the pre-computed distance information and efficient filter-and-refine strategy, the search process can be accelerated by reducing computational cost and disk access. We conducted extensive experiments to evaluate our proposed techniques against several known techniques, and the results showed that our techniques are superior in most cases.

Acknowledgement

This research was supported by the National Natural Science foundation of China under Grant No.60603045 and 60873063.

References

1. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data for data mining applications. In: Proc. ACM SIGMOD Conference, pp. 94–105 (1998)
2. Agrawal, R., Somani, A., Xu, Y.: Storage and querying of e-commerce data. In: Proc. 27th VLDB Conference, pp. 149–158 (2001)
3. Athitsos, V., Potamias, M., Papapetrou, P., Kollios, G.: Nearest neighbor retrieval using distance-based hashing. In: Proc. of ICDE Conference, pp. 327–336 (2008)
4. Beckmann, J.L., Halverson, A., Krishnamurthy, R., Naughton, J.F.: Extending rdbmss to support sparse datasets using an interpreted attribute storage format. In: Proc. 22nd ICDE Conference, p. 58 (2006)
5. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Comput. Surv. 33(3), 322–373 (2001)
6. Cui, B., Ooi, B.C., Su, J.W., Tan, K.L.: Contorting high dimensional data for efficient main memory processing. In: Proc. ACM SIGMOD Conference, pp. 479–490 (2003)
7. Hartigan, J., Wong, M.: A K-means clustering algorithm. Applied Statistics 28(1), 100–108 (1979)
8. Hui, J., Ooi, B.C., Shen, H., Yu, C., Zhou, A.: An adaptive and efficient dimensionality reduction algorithm for high-dimensional indexing. In: Proc. 19th ICDE Conference, p. 87 (2003)
9. Koudas, N., Ooi, B.C., Shen, H.T., Tung, A.K.H.: Ldc: Enabling search by partial distance in a hyper-dimensional space. In: Proc. 20th ICDE Conference, pp. 6–17 (2004)
10. Li, C., Chang, E.Y., Garcia-Molina, H., Wiederhold, G.: Clustering for approximate similarity search in high-dimensional spaces. IEEE Trans. Knowl. Data Eng. 14(4), 792–808 (2002)
11. Moffat, A., Zobel, J.: Self-indexing inverted files for fast text retrieval. ACM Trans. Information Systems 14(4), 349–379 (1996)
12. Tao, Y., Ye, K., Sheng, C., Kalnis, P.: Quality and efficiency in high-dimensional nearest neighbor search. In: Proc. ACM SIGMOD Conference, pp. 563–576 (2009)
13. Wang, C., Wang, X.S.: Indexing very high-dimensional sparse and quasi-sparse vectors for similarity searches. VLDB J. 9(4), 344–361 (2001)
14. Weber, R., Schek, H.J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proc. 24th VLDB Conference, pp. 194–205 (1998)
15. Yu, C., Ooi, B.C., Tan, K.L., Jagadish, H.V.: Indexing the distance: An efficient method to KNN processing. In: Proc. 27th VLDB Conference, pp. 421–430 (2001)

Highly Scalable Multiprocessing Algorithms for Preference-Based Database Retrieval

Joachim Selke, Christoph Lofi, and Wolf-Tilo Balke

Institut für Informationssysteme, Technische Universität Braunschweig
Mühlenpfordtstraße 23, Braunschweig, Germany
{selke,lofi,balke}@ifis.cs.tu-bs.de

Abstract. Until recently algorithms continuously gained free performance improvements due to ever increasing processor speeds. Unfortunately, this development has reached its limit. Nowadays, new generations of CPUs focus on increasing the number of processing cores instead of simply increasing the performance of a single core. Thus, sequential algorithms will be excluded from future technological advances. Instead, highly scalable parallel algorithms are needed to fully tap new hardware potentials. In this paper we establish a design space for parallel algorithms in the domain of personalized database retrieval, taking skyline algorithms as a representative example. We will investigate the spectrum of base operations of different retrieval algorithms and various parallelization techniques to develop a set of highly scalable and high-performing skyline algorithms for different retrieval scenarios. Finally, we extensively evaluate these algorithms to showcase their superior characteristics.

1 Introduction

Retrieval algorithms are at the heart of every database system and the search for ever more efficient algorithms in terms of scalability and runtimes has propelled research. The basic way of showing an algorithm's superiority is to implement it together with its competitors within the same framework and then evaluate how it behaves in different retrieval scenarios, usually exploring different problem instances, database sizes, and data distributions. It is interesting to note that this experimental evaluation is usually not depending on the hardware used (except for absolute run-time measurements, where however all competitors are affected by the hardware in a similar manner). Thus, any algorithm outperforming its competitors will do so on every platform and due to Moore's law will even get faster in absolute terms over time.

Moore's law states that the density of transistors on a CPU about doubles every two years and held for the last decades. Until recently it basically meant that also algorithms' performance doubles every two years, because increased transistor density was employed to increase clock rates and to support more complex CPU operations. But now processor designers have hit a ceiling as the benefits of even more transistors on a circuit cannot be harvested by traditional optimization techniques due to thermal problems. Chip manufacturers now opt to use the additional potential for replicating parallel processing cores. Quad-core machines are commonplace today, with 6-core processors projected for early 2010 and 80-core chips already existing as research prototype in Intel's Tera-scale project[1].

However, such additional cores do not necessarily result in increased performance, since most applications are build using sequential algorithms. Thus, taking advantage of this new potential depends on the aptitude of the application developer: the potential for parallel performance increase is best described by Amdahl's law. Amdahl's law stipulates that the speedup of any algorithm using multiple processors is strictly limited by the time needed to run its sequential fraction. For example, a program having a sequential fraction of as little as 10% can achieve a speedup factor of at most 10, no matter how many CPUs are available. On the other hand, the speedup factor achievable for its parallel portion is only capped by the actual number of processors. Thus, only highly parallel algorithms can benefit from future hardware improvements.

Actually, this problem already raised considerable attention. In its 2008 report on the IT landscape 2008–2012 Gartner Research ranks the need for multicore software architectures among the 10 most disruptive technologies. Making effective use of this technology will therefore need considerable changes in today's software development: "Running advanced multicore machines with today's software is like putting a Ferrari engine in a go-cart"[2]. The fact is illustrated in Fig. 1: with the advent of the multi-core era, the performance potential between parallel and sequential algorithms is continuously diverging.

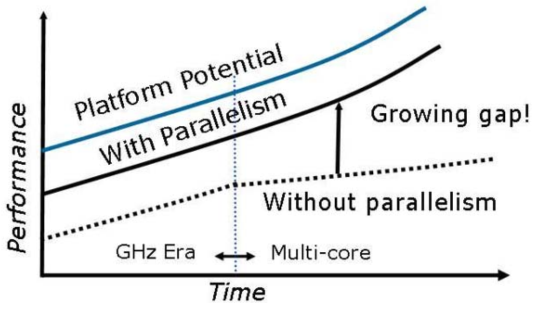


Fig. 1. The need for parallelism for algorithms' performance[3]

Of course, these developments also affect the field of databases which now needs to adapt their retrieval algorithms to the changing hardware landscape, too. In the course of this paper we will argue that for designing efficient and highly scalable multiprocessor retrieval algorithms even the development cycle has to be rethought. Instead of developing new algorithms only based on the drawbacks and shortcomings of the most current top-performers, algorithm designers have to take a closer look at the entire palette of possible base algorithms and exploit parallelizable features wherever possible. Only in this way the sequential part of the algorithm can be effectively

[1] http://techresearch.intel.com/articles/Tera-Scale/1421.htm

[2] "The Impact of Multicore Architectures on Server Scaling", Gartner, 2008.

[3] Image from "An Overview of the Parallelization Implementation Methods in Intel C++ Compilers", Intel Corporation, 2008.

minimized. For this it is also mandatory to deeply understand the basic operations needed for each algorithm. Our paper's contribution thus is threefold:

- Using the example of skyline queries we show how to use the variety of algorithms to set up the design space for parallel retrieval algorithms.
- We investigate which current techniques in parallelization can be used to efficiently implement modular operations.
- We apply our design process to BNL, one of the most prominent algorithms for skyline queries, and show that it leads to parallel implementations showing an almost linear scalability behavior in multi-core architectures.

2 The Parallelization Design Space for Retrieval Algorithms

It is interesting to note that most database retrieval algorithms are using a rather limited set of basic operations. Thus, the actual efficiency and the fine-tuning for specific scenarios is mainly achieved by innovative control flows and additional optimizations like e.g., using specialized index structures. For instance in the area of ranked query processing it was recently shown that is indeed possible to break down algorithms for retrieval tasks as different as skyline queries, top-k queries, and k-dominance queries to only three basic operations [23]. The idea is that such basic operations can be efficiently implemented within the database core and then offer interfaces for retrieval algorithms. Given the new hardware challenges it now becomes necessary to optimize those base operations for parallelization in order to benefit from future performance improvements by Moore's law.

In the following we will demonstrate by the example of skylining algorithms how such operations (in particular object comparisons in a nested-loop) can be parallelized with great effect by abstaining almost completely from sequential parts. The design space for improved algorithms is spanned on the one hand by the different evaluation approaches and their basic operations, on other hand by the novel parallelization techniques for these operations. The 'right' mixture of both will enable us to derive high-performing algorithms as we will see later in the experimental section.

2.1 The Design Space for Skyline Query Evaluation

Taking a closer look at recent algorithms for skyline query evaluation, we found that the algorithms can be classified into several distinguishable groups. In the following, we will cover the most important ones: block-nested-loops algorithms, divide-and-conquer algorithms, and multi-scan algorithms.

Algorithms of the *block-nested-loop class (BNL)* [3, 12, 13] are probably the most prominent algorithms for computing skylines. In fact the basic operation of collecting maxima during a single scan of the input data can be found at the core of several state-of-the-art skyline algorithms [5, 8, 9, 10, 11] and is illustrated in Figure 2. The "block" in its name refers to the fact that it linearly scans over the data set A and continuously maintains a block (or window) of data elements M containing the maximal elements with respect to the data read so far. For each data record $a \in A$ that BNL processes, the function BNL-STEP is called, which eliminates all records in M being dominated by a, and adds a to M, if a is not dominated. The major advantage of BNL is its simplicity and suitability for solving general comparison-based preference

```
function BNL($A$)
    $M \leftarrow \emptyset$
    for each $a \in A$ do
        $M \leftarrow$ BNL-STEP($M, a$)                ▷ update $M$
    end for
    return $M$
end function

function BNL-STEP($M, a$)
    for each $m \in M$ do
        if $m \succ a$ then
            return $M$                         ▷ $a$ is not a maximum
        else if $m \prec a$ then
            $M \leftarrow M \setminus \{m\}$                ▷ $m$ is not a maximum
        end if
    end for
    return $M \cup \{a\}$              ▷ $a$ and all $m \in M$ are maxima
end function
```

Fig. 2. Block-Nested-Loop (BNL) algorithm

```
function DC($A$)
    if $|A| \geq 2$ then
        split $A$ into $A_1$ and $A_2$ (preferably of equal size)
        $M_1 \leftarrow$ DC($A_1$)
        $M_2 \leftarrow$ DC($A_2$)
        for each $(m_1, m_2) \in M_1 \times M_2$ do
            if $m_1 \succ m_2$ then
                $M_2 \leftarrow M_2 \setminus \{m_2\}$
            else if $m_1 \prec m_2$ then
                $M_1 \leftarrow M_1 \setminus \{m_1\}$
            end if
        end for
        return $M_1 \cup M_2$
    else
        return $A$
    end if
end function
```

Fig. 3. Divide & Conquer algorithm

```
function BEST($A$)
    $M \leftarrow \emptyset$
    $A' \leftarrow A$
    while $A' \neq \emptyset$ do
        choose some $m \in A'$                ▷ choose candidate
        $A' \leftarrow A' \setminus \{m\}$
        for each $a \in A'$ do
            if $a \succ m$ then
                $m \leftarrow a$                    ▷ $a$ is the new candidate
                $A' \leftarrow A' \setminus \{m\}$
                restart the for-each-loop
            else if $a \prec m$ then
                $A' \leftarrow A' \setminus \{a\}$              ▷ $a$ is not a maximum
            end if
        end for
        $M \leftarrow M \cup \{m\}$                  ▷ $m$ is a maximum
    end while
    return $M$
end function
```

Fig. 4. The Best/sskyline algorithm

queries [4, 13] (i.e., BNL can also be used to compute the maxima of arbitrary partial orders). Furthermore, a multitude of optimization techniques is applicable to BNL algorithms like e.g. dynamic sorting or indexing [5].

A second class of algorithms for skyline evaluation is based on a straightforward *divide-and-conquer strategy*, as shown in Fig. 3. Given a data set A, it first checks the cardinality of A. In case $|A| = 1$ the algorithm simply returns A. Otherwise, A is split into two sets A_1 and A_2 and the algorithm applies itself recursively on both of them. The two results M_1 and M_2 are subsequently cleaned from local maxima by comparing each element of M_1 to each element of M_2 and removing all dominated elements during this process. The set of A's maxima is constructed by joining the reduced sets M_1 and M_2. Although the algorithm has excellent theoretical properties [3, 17], there is no efficient implementation of this recursive process [5]. The main problem preventing an efficient implementation seems to be that the algorithm either requires massive disk IO, or needs to keep a large amount of intermediate results in main memory. However, when abstaining from recursion, its basic split-and-merge scheme is definitely applicable in a parallel scenario.

The third class of skyline algorithms is based on *multiple scans* of the database instance and includes algorithms like *Best* or *sskyline* [14, 15, 16]. They can especially provide highly efficient cache-conscious implementations. The complete algorithm is shown in Fig. 3. After creating an in-memory copy A' of A, an arbitrary element $m \in A'$ is selected as maximum candidate. After removing m from A', the whole set A' is scanned, and dominated elements are removed from it. If some element is found in A' that dominates the current candidate m, it is removed from A' and used as the new candidate; then, the scan of A' is restarted. If no element of A' dominates the current candidate, then it is a maximum. The whole process is repeated until A' is empty. In our experiments, we found Best/sskyline to make around half the number of comparisons as BNL due to the early elimination of maxima. But to reach its performance, Best/sskyline must scan (and even modify) the data set numerous times. This can only be done efficiently, if the whole input data set A fits in main memory; a requirement not desirable for general skyline algorithms. Nevertheless, as we will see in Section 0 Best/sskyline can form a useful building block in a parallel algorithm.

2.2 The Design Space for Parallelization of Basic Operations

For parallelizing the skyline problem, different strategies are at hand: The first is *classical distribution*, i.e. splitting data into multiple work packages which are distributed among the worker threads. Such threads can work independently of each other and finally, their results are combined. This strategy adapts the split-and-merge concept found within algorithms of the divide-and-conquer class. The advantage of this style of algorithm is that the threads do not need shared memory and thus could also be deployed to different machines (e.g. computer clusters). However, it is necessary to combine the results of the threads which introduces some overhead in terms of the program's sequential part and may lead to suboptimal scalability. In summary, skyline algorithms following the split-and-merge approach show good performance when only few threads can be used, since the overhead introduced increases with a higher degree of parallelism.

The second strategy is to employ algorithms working on a *shared data structure*, i.e., each thread can read and modify the same dataset. This style of algorithm has just recently become viable due to the advent of tightly coupled multicore processors. Still, the main problem of shared-memory algorithms remains at hand: One has to ensure that no data is read or written which has just been accessed by another thread (dirty reads or writes) in order to avoid data inconsistency. When considering for example a block-nested loops algorithm two major critical situations can be identified: a) overtaking threads: in this case, the overtaking thread will lose a comparison with the current element of the slower thread b) deleting/appending: the list structure may corrupt, if two threads try to delete or append the same nodes simultaneously due to the resulting inconsistent linkage of node.

Like in transaction systems, these problems can be tackled by synchronization and locking protocols. However, there is a variety of different approaches to secure a shared data structure, each showing individual runtime performance. We will briefly introduce these common approaches also used in our following algorithms:

- *Full Synchronization:* this simple protocol locks the whole data structure for every access. Obviously, this will not allow any parallelism and is, although secure, unsuitable for performance-oriented algorithms.

- *Continuous Locking:* the data structure is locked at node level for every access and is a straight-forward semi-naïve approach to the problem. However, locking still carries an expensive overhead despite recent hard- and software progress. Thus, this technique suffers severely from the overhead induced by acquiring and releasing such a high number of locks.
- *Lazy Locking:* this approach is similar to continuous locking. However, it aims at using as few locks as possible. Locks are only acquired when they are really needed, i.e. when deleting or inserting nodes. Unfortunately, this approach leads to more complex algorithms which are harder to design and debug. For example, it is necessary to identify all critical situations and provide according safeguards. Also, in case of our implementations, additional security mechanisms (like using flags) are necessary to ensure the data structure's consistency. But from a performance point of view lazy locking algorithms are definitely superior to the previous two locking protocols in highly parallel scenarios. Still, for cases using very few threads (e.g. two threads) split-and-merge algorithms may be a better choice.
- *Lock Free Synchronization:* this technique completely abstains from using locks. Instead, a special hardware instruction within the CPU core is used to implement an optimistic protocol. The base idea is to perform changes to the data structure and check later whether any concurrent modifications had occurred. When a conflict occurs, all modifications are undone and repeated. The performance of lock free protocols scale with the probability of conflicts: the more likely conflicts occur; the worse is the algorithm's expected performance.
  Interestingly, we observed in our experiments a similar or slightly lower performance of lock free synchronization compared to the lazy locking variants. However, the performance ratio of those two techniques is depending on the hardware efficiency of locking compared to the instructions used in lock free synchronization; thus performance may change when using different CPU architectures or operation systems and both algorithm styles seem viable alternatives.

3 Parallel Skyline Computing

In this section, we will utilize our design space for designing parallel algorithms with the mentioned techniques. Up to now, all parallel skyline algorithms proposed in literature directly rely on the basic divide-and-conquer scheme. Following our design considerations, these algorithms already cover an important application scenario and thus will serve as our baseline in the later experiments. In addition, for the *multi-core shared-memory* scenario we will design novel algorithms and show that they indeed outperform their current parallel competitors.

3.1 Algorithms Using Split-and-Merge Parallelization

The multi-processor scenario without shared memory directly calls for algorithms based on the split-and-merge parallelization scheme used by the divide-and-conquer class. Basically, the input data set A is first split into k parts $A_1, \dots, A_k$. Then, in parallel, the skyline of each part is computed (using any sequential algorithm for each part). Finally, the resulting k local skylines $M_1, \dots, M_k$ are merged to produce the global skyline. To foster parallelization, it is sensible to split A in at least as many parts as there are processors.

In particular, the following distributed merging method has been applied in a distributed scenario without shared memory [24]: First, assign the i-th local skyline M_i to node i and make the union of all other local skylines $\overline{M_i} = M_1 \cup \cdots \cup M_{i-1} \cup M_{i+1} \cup \cdots \cup M_k$ accessible to this node. Then, node i compares each element of M_i to all elements in $\overline{M_i}$, removing all dominated elements from M_i. After this step, M_i only contains elements of A's global skyline. Finally, the set of all global maxima $M = M_1 \cup \cdots \cup M_k$ is constructed at some central node. This algorithm will be referred to as Distributed in the following. It is interesting to note that the Distributed algorithm does not rely on shared-memory at any point, thus it particularly applies to cluster-style distributed scenarios. However, its performance suffers from the complicated merge step.

Focusing on scenarios with shared memory this shortcoming is remedied by the pskyline method proposed in [16], which also applies the split-and-merge scheme. After splitting the dataset similar to the Distributed algorithm, pskyline uses a main memory algorithm (Best/sskyline) for the computation of the k local skylines. But by deciding for an efficient main memory implementation, it cannot process all k parts of A in parallel on large data sets. Instead, if r cores are available, the total number of chunks is chosen such that r chunks jointly fit into main memory. After all k local skylines have been computed (and written to disk), an improved merge scheme relying on shared memory is used: in deep-left-tree style, local skylines are merged successively, two at a time. In this shared memory merging step, all r cores can be used simultaneously. The experimental evaluation in [1] indicates that the psykline algorithm is indeed highly scalable in cases where the skyline is large relative to the number of database tuples. However, scalability degrades rapidly for smaller skyline sizes. For example, a speedup of just about 4 is reported for 8 cores and a skyline size of about 20% of the database size.

3.2 Continuously Locked Parallel BNL

Exploiting our algorithmic design space, we now explore algorithms for a shared-memory multi-core scenario departing from split-and-merge techniques. As we have seen the aim is true parallelization with a minimum of sequential overhead. Since algorithms of the BNL class only need a single pass over the input data, the basic idea of the following algorithms is to extend the BNL algorithm in such a way that multiple worker threads can simultaneously share and modify BNL's window. In the following, the window is represented by a linked list data structure.

Of course entering the shared memory part of the design space means that we have to care for the thread safety, i.e., concurrent access to data must be guarded by using an adequate locking scheme or conflict resolution. The simplest viable locking scheme is to continuously lock each node being accessed. Thus, each thread traversing the linked list releases a node's lock only after acquiring the lock for its successor. In particular, this strategy prevents that threads can pass each other. Passing might result in an unnoticed removal of a node by some other thread. This technique of always holding two locks per thread and adhering to the lock/unlock order is known as lock coupling (or hand-over-hand locking) [19]. Within BNL's outer loop, each thread continuously requests a new data record from the central data manager. For each data record, the list is traversed and each node is compared to the current data record, performing deletions of nodes, if needed. In case no node dominated the current record, it is appended to the list. Although this approach is a valid parallel implementation of the BNL algorithm, its way of locking significantly limits its performance. If two threads access neighboring nodes, they will continuously interfere with each other while traversing the list. Furthermore, the omnipresent locking and unlocking operations introduce a large computational overhead.

3.3 Lazy List Parallel BNL

Actually—and in contrast to the continuously locking scheme—the autonomy of threads only has to be constrained when a node has to be modified. This means that for each addition or deletion of a node in the list, a modification lock has to be obtained, whereas simple comparison operations do not require explicit locks. This idea of modification locks can be implemented using a novel concurrent data structure called the Lazy List [20]. We adjusted the basic structure for the use in skyline computation and show the resulting code in Figure 5.

```
procedure ParBNL-LazyList-Thread()
  NextRecord:
  while ($a \leftarrow$ GetNextRecord()) $\neq$ null do
    TraverseList:
    $pred \leftarrow$ head
    $curr \leftarrow pred$.next
    while $curr \neq$ tail do                          ▷ iterate
      if $curr$.item $\succ a$ then
        goto NextRecord                           ▷ process next record
      else if $curr$.item $\prec a$ then
        $pred$.lock                             ▷ try to delete current node
        $curr$.lock
        if Validate($pred$, $curr$) then
          $curr$.deleted = true                  ▷ looks good, go on
          $pred$.next = $curr$.next
          $curr$.unlock
          $pred$.unlock
        else
          $curr$.unlock                           ▷ error, restart
          $pred$.unlock
          goto TraverseList
        end if
        $curr$ = $curr$.next;
      else
        $pred \leftarrow curr$
        $curr \leftarrow curr$.next
      end if
    end while
    $pred$.lock                  ▷ it is $curr$ = tail, try to append $a$
    if Validate($pred$, $curr$) then
      $new \leftarrow$ Node($a$)                    ▷ looks good, go on
      $pred$.next $\leftarrow new$
      $new$.next $\leftarrow$ tail
      $pred$.unlock
    else
      $pred$.unlock                               ▷ error, restart
      goto TraverseList
    end if
  end while
end procedure

function Validate($pred$, $curr$)
  return $\neg pred$.deleted $\wedge$ $\neg curr$.deleted $\wedge$ $pred$.next = $curr$
end function
```

Fig. 5. The parallel BNL Lazy List algorithm

```
procedure ParBNL-LockFree-Thread()
  NextRecord:
  while ($a \leftarrow$ GetNextRecord()) $\neq$ null do
    TraverseList:
    $pred \leftarrow$ head
    $curr \leftarrow pred$.next
    while $curr \neq$ tail do
      $\langle succ, currdel \rangle \leftarrow \langle curr$.next, $curr$.deleted$\rangle$
      if $currdel$ then
        $cas \leftarrow$ CAS$(\langle pred$.next, $pred$.deleted$\rangle$,
                  $\langle curr$, false$\rangle \rightsquigarrow \langle succ$, false$\rangle)$
        if $cas$ then
          $curr \leftarrow succ$
        else
          goto TraverseList
        end if
      else
        if $curr$.item $\succ a$ then
          goto NextRecord
        else if $curr$.item $\prec a$ then
          $cas \leftarrow$ CAS$(\langle curr$.next, $curr$.deleted$\rangle$,
                    $\langle succ, \star \rangle \rightsquigarrow \langle succ$, true$\rangle)$
          if $cas$ then
            CAS$(\langle pred$.next, $pred$.deleted$\rangle$,
                  $\langle curr$, false$\rangle \rightsquigarrow \langle succ$, false$\rangle)$
          else
            goto TraverseList
          end if
          $curr \leftarrow succ$
        else
          $pred \leftarrow curr$
          $curr \leftarrow curr$.next
        end if
      end if
    end while
    $new \leftarrow$ Node($a$)
    $new$.next $\leftarrow$ tail
    $cas \leftarrow$ CAS$(\langle pred$.next, $pred$.deleted$\rangle$,
              $\langle$tail, false$\rangle \rightsquigarrow \langle new$, false$\rangle)$
    if $\neg cas$ then
      goto TraverseList
    end if
  end while
end procedure
```

Fig. 6. The lock-free parallel BNL algorithm using a linked list

In particular, our algorithm uses a binary flag $deleted$ to guarantee that no adjacent nodes are modified concurrently, which might otherwise result in violating the pointer integrity. Before a node can be deleted, locks for the current node and its predecessor have to be acquired. The removal of the current node $curr$ then is always a two-step process: First, $curr.deleted$ is set to true, indicating $curr$'s logical removal (and thus locking it effectively for modifications by its successor), and second, the predecessor's pointer $pred.next$ has to be set to $curr.next$, thus unlinking the current node. By calling the $validate$ function it is checked whether $pred$ and $succ$ have been deleted in the meantime or some node has been inserted in-between.

Depending on the result, either the current node can be deleted, or there had been some concurrent modification of this list, from which the algorithm recovers by restarting the iteration from the beginning of the window. Appending a new node works similar. First, a lock is obtained (locking $pred$ is sufficient in this case), second, the algorithm checks for concurrent modifications, and then either appends the node physically or restarts the iteration.

3.4 Lock-Free Parallel BNL

Most approaches falling into our parallelization design space are using locking protocols, e.g., as our previous algorithm. The alternative is to completely abstain from locking and to implement a non-blocking *optimistic protocol*. This goal is supported by a special hardware operation called compare-and-swap (CAS). The CAS operation *atomically* compares a variable to some given value and, if both are the same, sets the first variable to some given new value. In our algorithm shown in Figure 6, it is denoted as $CAS(x, y \rightsquigarrow z)$: it atomically compares the variable x to the value y and, in case $x = y$, sets x to the value z. The function returns $true$ if the operation succeeded, otherwise it returns $false$. In our case, we will always use compare-and-swap to guard the node pointer $next$ and the deletion flag $deleted$.

Our approach to lock-free lists is inspired by the Harris-Michael algorithm [43, 44]. The linked list is traversed as in the sequential BNL algorithm, except for the following: first, an additional pointer variable $succ$ is used to store the successor of the current node, and second, in each traversal step, the $deleted$ flag of the current node is checked in order to physically remove nodes that have previously been marked for deletion. If the flag is set to $true$, the algorithm tries to unlink the current node using a CAS operation. In case of failure, a concurrent modification has occurred and the list traversal is restarted.

Dominated nodes are removed by first marking the current node as deleted using CAS (line 21). If this operations succeeds, the algorithms tries to physically delete the node (line 24, the star indicates that the current value of $curr.deleted$ does not matter); otherwise, the traversal is restarted. It does not matter, whether the physical removal succeeds or not, since logically deleted nodes will be cleaned up anyway by other threads during their list traversal as previously described (line 10).

Finally, new nodes are appended to the list also by a CAS instruction (line 38). In case of failure, the list traversal is restarted from the beginning. Although we do not use any locking mechanism in this algorithm, checking the status flag of all (critical) CAS operations allows us to detect all concurrent list modifications and respond accordingly.

4 Experiments

In the previous sections we explored our design space from an algorithmic point of view and derived several algorithms for shared-memory skyline computation. But similar to physical tuning methods, only thorough experimentation will reveal the real-world performance of the outlined techniques. In order to investigate the full variety of performance characteristics in an unbiased fashion, the following evaluations will be executed on synthetic datasets. To create these datasets, we used the Independent data generator commonly used in skyline research [3]. We also applied all evaluations to the Anticorrelated, and Correlated data generators. However, all results show the same trends as those seen in the Independent evaluations and were thus omitted for brevity. Of course, data sets of varying size ($n = 100K, 1M, 10M$) and varying dimensionality ($d = 5, 6, 7, 8, 9$) have been used.

All our experiments have been conducted on a *single* node of a 12-node cluster running SUSE Linux Enterprise 10. The node we used is equipped with two Intel Xeon E5472 3.0 GHz quad-core processors, thus providing a total of 8 cores at each node. Our algorithms have been implemented using the Java programming language version 6.0. We only used the built-in mechanisms for locking, compare-and-swap operations, and thread management. The experiments are executed on a Sun Java 6.0u13 64Bit server JVM in HotSpot mixed mode.

We will compare the following algorithms: (i) pskyline (ii) the continuously-locked parallel BNL (referred to as Locked in the following), (iii) the Lazy List parallel BNL (Lazy List), (iv) the lock-free parallel BNL (Lock-Free), and (v) the distributed merging method from Section 0 (Distributed; to provide a fair comparison here, we used BNL as underlying sequential algorithm). Unless stated otherwise, all experiments have been performed on one up to eight CPU cores.

4.1 Memory Usage and the Role of pskyline

As discussed in Section 0, pskyline relies on the sskyline algorithm for computing local skylines, which subsequently are merged by a parallel merging scheme. As sskyline requires data to be present in main memory, this imposes further restrictions on pskyline as well. Investigating this issue in detail is particularly interesting since pskyline has been reported to perform extremely well if the whole database can be loaded into main memory [16]. Since such a scenario is hardly realistic in database retrieval. In the following, we investigated the performance of pskyline under main memory restrictions.

We evaluated pskyline exemplarily on the Independent dataset with $n = 1M$ and $d = 6$ using a fixed number of $r = 8$ CPU cores, resulting in a skyline size of $m = 5577$ in average. We restricted the main memory available to pskyline relative to the skyline size m. In particular, we tested pskyline's performance with memory sizes $1.5m$, $2m$, $5m$, $10m$, and $20m$. Finally, in order to allow the algorithm to perform with its maximal performance, we allow it to load the full database into main memory. We also abstained from writing unused working sets to the hard disk as proposed in the original work on pskyline [16], thus giving pskyline a significant performance boost. Our results are presented in Figure 14, showing the computation time in milliseconds for different main memory sizes.

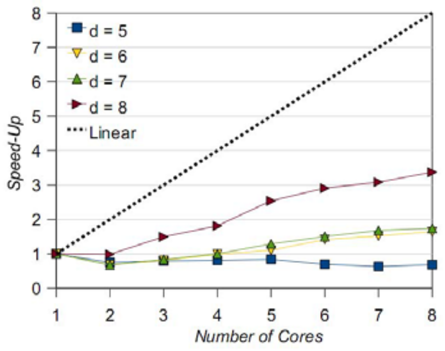


Fig. 7. Speed-Up of Locked (Independent, n = 1M)

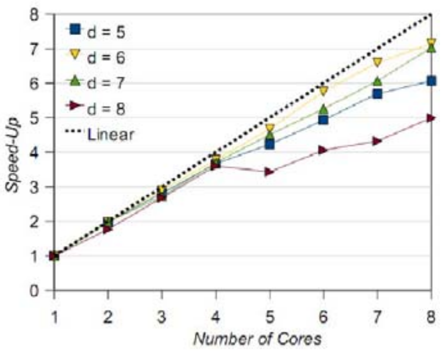


Fig. 8. Speed-Up of Lock-Free (Independent, n = 1M)

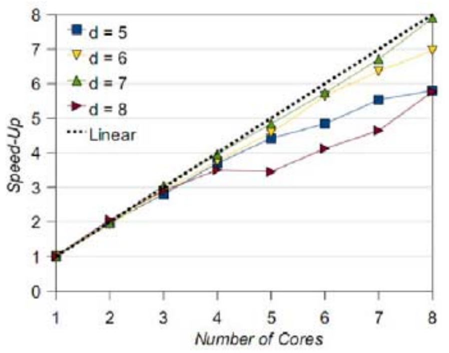


Fig. 9. Speed-Up of Lazy List (Independent, n = 1M)

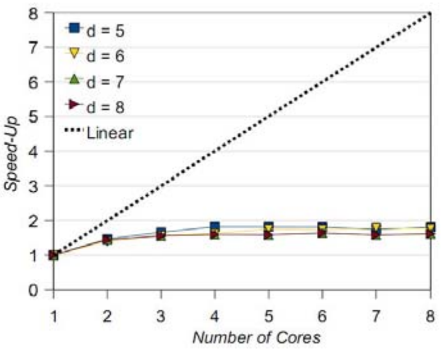


Fig. 10. Speed-Up of Distributed (Independent, n = 1M)

As a comparison baseline, we also evaluated Lazy List on the same dataset (indicated by the dotted line in Figure 14). In fact, Lazy List only requires as much main memory as necessary to hold the result set. It can be observed that the performance of pskyline suffers severely under main memory restrictions. However, if no memory restrictions are applied, pskyline shows competitive and even slightly better performance to Lazy List. These general observations also hold for other scenarios, e.g. using the Correlated or Anticorrelated data. To summarize, pskyline performs very well in main memory database scenarios, but on the whole is memory inefficient. Due to this fact, we will exclude it from the following evaluations. In contrast, the memory consumption of Locked, Lazy List, and Lock-Free had always been around the number $1.5m$, thus being close the optimum of m. The Distributed algorithm requires roughly $2m$ of main memory.

4.2 Speed-Up and Scaling

In this section, we evaluate and compare the multiprocessing scalability and overall performance of the remaining four algorithms. For pure scalability observations, we

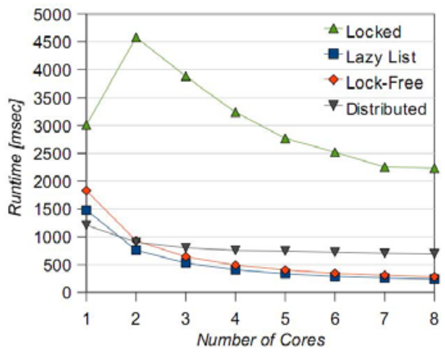


Fig. 11. Runtime comparison (Independent, n =100k, d = 7)

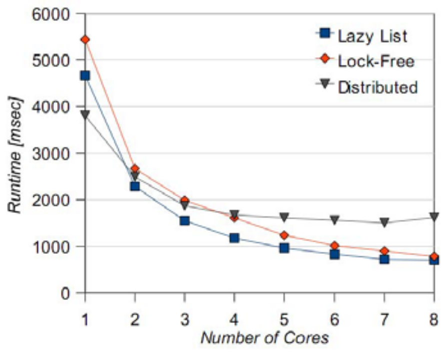


Fig. 12. Runtime comparison (Independent, n = 10M, d = 5)

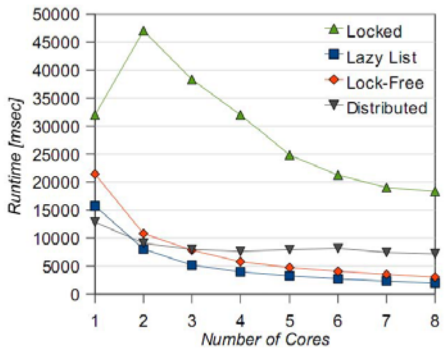


Fig. 13. Runtime comparison (Independent, n = 1M, d = 7)

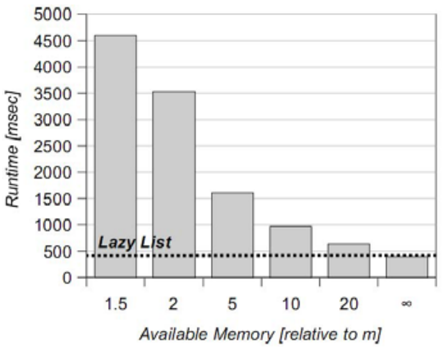


Fig. 14. Runtime of pskyline (Independent, n = 1M, d = 7)

will use the Independent data set, where n = 1M and d = 5−8. Setting d to higher values gives unrealistically large result sets ($m \gg 50\text{K}$). Each algorithm has been executed on $r = 1, \ldots, 8$ cores. The respective results are presented in Figures 7−10. For the Locked algorithm, only a small speed-up can be observed due to high lock contention thrashing. The speedup factor grows with m, and peaks in our test with a value of 3.3 for 8 cores and $d = 8$. It is also interesting to note that the algorithm performs better using just one core than using two of them. This can be explained by the Java VM being able to detect the needlessness of locking, if there are no concurrent accesses, thus dynamically disabling the use of locks.

In contrast, Lazy List (Fig. 9) and Lock-Free (Fig. 8) show significantly better speed-up behavior compared to the Locked version. While showing similar performance to each other, Lazy List performs slightly better overall. It can be observed that both algorithms show nearly linear scaling up to four cores. Starting with the fifth core, the performance gain moderately ceases for both algorithms and data sets with $d = 5$ and $d = 8$, but only slightly degenerates for the other cases. This phenomenon may be explained with decreasing cache locality and increasing communication overhead as our test system uses two quad-core processors. Starting with the fifth core, the second processor must constantly communicate with the first one over the slower Front Side Bus (compared to communication among cores within a single processor). However, we also expect a nearly linear speed-up for true 8-core-processors, which will be available in the near future. For both Lazy List and Lock-Free, we measured maximum speed-ups of 7.9 (in case $d = 7$) and 5.8 (in case $d = 8$) using 8 cores.

Finally, the Distributed algorithm (Fig. 10) shows almost no speedup. Peak scaling was measured with a speed-up factor of 1.74 for 8 cores and $d = 7$.

For assessing the runtime performance of the algorithms in absolute numbers, we measured the computation time for $n = 100K$, 1M and $d = 7$ as well as for $n = 10\text{M}$ and $d = 5$. These cases have been selected according to their representativeness regarding practical cases in preference-based retrieval. All results can be found in Figures 11−13. During this evaluation, it turned out that for the 8-core cases, Lazy List shows the highest performance (1992 ms for $n = 1\text{M}$), followed by Lock-Free (3052 ms for $n = 1\text{M}$), Distributed (7802 ms for $n = 1\text{M}$), and, finally, Locked (18,375 ms for $n = 1\text{M}$). Please note that Locked has been left out in Fig. 13 due to its extremely poor performance.

In summary, we have shown that Lazy List as well as the Lock Free algorithm show very good scaling behavior and overall performance, while ensuring near optimal memory efficiency due to their block-nested-loops lineage.

Source Code and Repeatability: In order to provide a reliable and referencable foundation for further algorithm development, all our presented algorithms can be accessed and downloaded at http://www. ifis.cs.tu-bs.de/javalib/skysim.zip.

5 Conclusion and Outlook

In this paper, we established a design space for parallel database retrieval algorithms, in particular focusing on skyline algorithms as a representative example. Identifying basic operations of those algorithms, we investigated their respective potential for parallelization using different techniques. Finally, we designed exemplarily two innovative and highly scalable algorithms based on the popular block-nested-loops class for the scenario of shared-memory multi-processor systems. In our extensive evaluations, we showcased the superior characteristics and scalability of these algorithms in different settings. Although we did not exploit any special skyline-specific optimization techniques, we were able to outperform state-of-the-art approaches to skyline computation on multiprocessor systems in these scenarios. Especially our lazy list algorithm and lock-free BNL algorithm showed excellent overall performance with nearly linear scaling.

The design considerations and implementation techniques presented in this paper pave the way for also tapping the parallel potential of state-of-the art algorithms also for other retrieval scenarios. Moreover, future work will adapt our techniques to further optimizations for skyline algorithms like tree-based indexing or dynamic sorting.

References

1. Vitter, J.S.: Algorithms and data structures for external memory. Foundations and Trends in Theoretical Computer Science 2(4), 305–474 (2006)
2. Larus, J.: Spending Moore's dividend. Communications of the ACM 52(5), 62–69 (2009)
3. Börzsönyi, S., Kossmann, D., Stocker, K.: The Skyline operator. In: Proceedings of the 17th International Conference on Data Engineering (ICDE 2001), pp. 421–430. IEEE Computer Society, Los Alamitos (2001)
4. Chomicki, J.: Preference formulas in relational queries. ACM Transactions on Database Systems 28(4), 427–466 (2003)
5. Godfrey, P., Shipley, R., Gryz, J.: Algorithms and analyses for maximal vector computation. The VLDB Journal 16(1), 5–28 (2007)
6. Morse, M., Patel, J.M., Jagadish, H.V.: Efficient skyline computation over low-cardinality domains. In: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007), pp. 267–278. ACM Press, New York (2007)
7. Preisinger, T., Kießling, W.: The Hexagon algorithm for Pareto preference queries. In: Proceedings of the 3rd Multidisciplinary Workshop on Advances in Preference Handling, M-PREF 2007 (2007)
8. Eng, P.-K., Ooi, B.C., Tan, K.-L.: Indexing for progressive skyline computation. Data 4 Knowledge Engineering 46(2), 169–201 (2003)
9. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: Dayal, U., Ramamritham, K., Vijayaraman, T.M. (eds.) Proceedings of the 19th International Conference on Data Engineering (ICDE 2003), pp. 717–719. IEEE Computer Society, Los Alamitos (2003)
10. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Transactions on Database Systems 30(1), 41–82 (2005)
11. Bartolini, I., Ciaccia, P., Patella, M.: Efficient sort-based skyline evaluation. ACM Transactions on Database Systems 33(4) (2008)
12. Bentley, J.L., Clarkson, K.L., Levine, D.B.: Fast linear expected-time algorithms for computing maxima and convex hulls. Algorithmica 9(2), 168–183 (1993)
13. Daskalakis, C., Karp, R.M., Mossel, E., Riesenfeld, S., Verbin, E.: Sorting and selection in posets. In: Mathieu, C. (ed.) Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009), pp. 392–401. SIAM, Philadelphia (2009)
14. Torlone, R., Ciaccia, P.: Finding the best when it's a matter of preference. In: Ciaccia, P., Rabitti, F., Soda, G. (eds.) Proceedings of the 10th Italian Symposium on Advanced Database Systems (SEBD 2002), pp. 347–360 (2002)
15. Boldi, P., Chierichetti, F., Vigna, S.: Pictures from Mongolia: Extracting the top elements from a partially ordered set. Theory of Computing Systems 44(2), 269–288 (2009)
16. Park, S., Kim, T., Park, J., Kim, J., Im, H.: Parallel skyline computation on multicore architectures. In: Proceedings of the 25th International Conference on Data Engineering (ICDE 2009), pp. 760–771. IEEE Computer Society, Los Alamitos (2009)
17. Bentley, J.L., Kung, H.-T., Schkolnick, M., Thompson, C.D.: On the average number of maxima in a set of vectors and applications. Journal of the ACM 25(4), 536–543 (1978)
18. Sun, M.: A primogenitary linked quad tree data structure and its application to discrete multiple criteria optimization. Annals of Operations Research 147(1), 87–107 (2006)
19. Bayer, R., Schkolnick, M.: Concurrency of operations on B-trees. Acta Informatica 9(1), 1–21 (1977)

20. Heller, S., Herlihy, M., Luchang co, V., Moir, M., Scherer III, W.N., Shavit, N.: A lazy concurrent list-based set algorithm. Parallel Processing Letters 17(4), 411–424 (2007)
21. Harris, T.L.: A pragmatic implementation of non-blocking linked-lists. In: Welch, J.L. (ed.) DISC 2001. LNCS, vol. 2180, pp. 300–314. Springer, Heidelberg (2001)
22. Michael, M.M.: High performance dynamic lock-free hash tables and list-based sets. In: Proceedings of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 2002), pp. 73–82. ACM Press, New York (2002)
23. Levandoski, J., Mokbel, M., Khalefa, M.: FlexPref: A Framework for Extensible Preference Evaluation in Database Systems. In: International Conference on Data Engineering (ICDE), Long Beach, CA, USA (2010)
24. Cosgaya-Lozano, A., Rau-Chaplin, A., Zeh, N.: Parallel computation of skyline queries. In: Proceedings of the 21st International Symposium on High Performance Computing Systems and Applications (HPCS 2007). IEEE Computer Society, Los Alamitos (2007)

IO^3: Interval-Based Out-of-Order Event Processing in Pervasive Computing

Chunjie Zhou* and Xiaofeng Meng

School of Information, Renmin University of China, Beijing, China
{lucyzcj,xfmeng}@ruc.edu.cn

Abstract. In pervasive computing environments, complex event processing has become increasingly important in modern applications. A key aspect of complex event processing is to extract patterns from event streams to make informed decisions in real-time. However, network latencies and machine failures may cause events to arrive out-of-order. In addition, existing literatures assume that events do not have any duration, but events in many real world application have durations, and the relationships among these events are often complex. In this work, we first analyze the preliminaries of time semantics and propose a model of it. A hybrid solution including time-interval to solve out-of-order events is also introduced, which can switch from one level of output correctness to another based on real time. The experimental study demonstrates the effectiveness of our approach.

Keywords: pervasive computing, complex event, out-of-order, time interval.

1 Introduction

In pervasive computing environments, event processing has raised increased interest in the past few years [1,4,3]. A growing numbers of applications nowadays take the form of time ordered or out-of-ordered series, and every event has time duration. All these factors about time are very important in extracting patterns from atomic events in pervasive computing. However, the present literatures about time management in pervasive computing usually refer to only one aspect of the problem.

Existing research works [1] almost focus on instantaneous events. However, purely sequential queries on instantaneous events are not enough to express many real world patterns. For example, it has been observed that in many diabetic patients, the presence of hyperglycemia1 overlaps with the absence of glycosuria [10]. This insight has led to the development of effective diabetic testing kits. Clearly, there is a need for an efficient algorithm that can solve events with duration.

Meanwhile, the real-time processing in time order of event streams generated from distributed devices is a primary challenge in pervasive computing environments. However, most systems [2]assume a total ordering among event arrivals. It has been illustrated that the existing technology would fail in such circumstances, either missing

* This research was partially supported by the grants from the Natural Science Foundation of China under Grant No. 60833005; National High-Tech Research and Development Plan of China under Grant No. 2007AA01Z155, 2009AA011904; the Ph.D. Programs Foundation of Ministry of Education of China No.200800020002

resulting matches or incorrectly producing incorrect matches. Let us consider a popular application for tracking books in a bookstore [3] where RFID tags are attached to each book and RFID readers are placed at strategic locations throughout the store, such as book shelves, checkout counters and the store exit. If a book shelf and a store exit sensed the same book but none of the checkout counters sensed it in between the occurrence of the first two events, then we can conclude that this book is being shoplifted. If events of sensing at the checkout counters arrive out-of-order, we cannot ever output any results if we want to assure correctness of results. Otherwise, if we focus on real-time alarm, we may output the wrong results. Clearly, it is imperative to deal with both in-order as well as out-of-order arrivals efficiently and in real-time.

The contributions and the rest of this work are organized as follows: Section 2 gives the related works. Section 3 provides some preliminaries, including the time semantics and the model of interval-based out-of-order events. Section 4 describes a hybrid solution. Section 5 gives the experiment results and we conclude in Section 6.

2 Related Works

About the out-of-order problem, from the aspect of different scenarios, the literatures can be divided into two types, one focus on real time,another pay more attention to correctness. Since the input event stream to the query engine is unordered, it is reasonable to produce unordered output events. So [6] permits unordered sequence output and proposes the aggressive strategy.

If ordered output is needed, additional semantic information such as K-Slack factor or punctuation is needed to ”unblock” the on-hold candidate sequences from being output. A native approach [2] is using K-Slack as a priori bound on the out-of-order of the input streams. The biggest drawback of K-slack is rigidity of the K that cannot adapt to the variance in the network latencies that exists in a heterogeneous RFID reader network. Another solution is applying punctuations, namely, assertions inserted directly in the data stream confirming that for instance a certain value or time stamp will no longer appear in the future input streams [6]. [6] use this technique and propose a solution called conservative method. However, such techniques, while interesting, require for some service to first be creating and appropriately inserting such assertions.

About the interval-based events, there has been a stream of research [7,8,9,10]. Kam et. al. [7] designs an algorithm that uses the hierarchical representation to discover frequent temporal patterns. However, the hierarchical representation is ambiguous and many spurious patterns are found. Papapetrou et. al. [8] proposes the H-DFS algorithm to mine frequent arrangements of temporal intervals.This approach does not scale well when the temporal pattern length increases. Wu et. al. [9] devises an algorithm called TPrefix for mining non-ambiguous temporal pattern from interval-based events. TPrefixSpan has several inherent limitations: multiple scans of the database are needed and the algorithm does not employ any pruning strategy to reduce the search space. In order to overcome the above drawbacks, [10] gives a lossless representation to preserve the underlying temporal structure of the events, and proposes an algorithm to discover frequent temporal patterns from interval-based events. However, they only use this representation for classification but don’t consider out-of-order problems.

3 Preliminaries

Each event is denoted as $(ID, V_s, V_e, O_s, O_e, S_s, S_e, K)$. Here V_s and V_e respectively denote valid start and end time; O_s and O_e respectively denote occurrence start and end time; S_s corresponds to the system server clock time upon event arrival; K corresponds to an initial insert and all associated retractions, each of which reduce the S_e compared to the previous matching entry in the table.

Definition 1 (time interval) *Suppose $T_1 \subseteq \ldots \subseteq T_n$ is a linear hierarchy H of time units. For instance, $H = minute \subseteq hour \subseteq day \subseteq month \subseteq year$. A time interval T_i in H is an n-tuple $(t_{i_1}, \ldots, t_{i_n})$ such that for all $1 \le i \le n$, t_{i_i} is a time-interval in the time-value set of T_i [5].*

Definition 2 (out-of-order event) *Consider an event stream $S: e_1, e_2, ..., e_n$, where $e_1.ats < e_2.ats < ... < e_n.ats$. For any two events e_i and e_j $(1 \le i, j \le n)$ from S, if $e_i.ts < e_j.ts$ and $e_i.ats < e_j.ats$, we say the stream is an ordered event stream. If however $e_j.ts < e_i.ts$ and $e_j.ats > e_i.ats$, then e_j is flagged as an out-of-order event.*

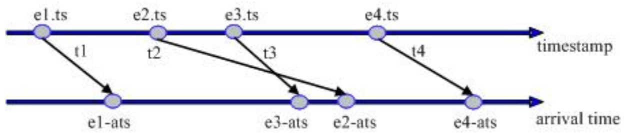


Fig. 1. out-of-order event

In the example shown in Figure 1, the timestamp of events $e_1 \sim e_4$ are listed in order. But we can see that the earlier event e_2 received later than event e_3, which is called out-of-order.

In the following query format, Event Pattern connects events together via different event operators; the WHERE clause defines the context for event pattern by imposing predicates on events; the WITHIN clause describes the time range during which events that match the pattern must occur. Real-time Factor specifies the real-time requirement intensity of different users.

```
<Query>::=EVENT <event pattern>
   [WHERE <value constraints>]
   [WITHIN <time constraints>]
   [Real-time Factor {0,1}]
   <event pattern>::=SEQ/PAL((Ei(relationship)Ej)
                    (!Ek)(relationship)El))(1≤i,j,k,l≤n)
   relationship::=overlap,before,after
   <time constraints>::= Time Window length W
```

In pattern queries among interval-based events, the state transition not only depends on the event type, but also the relationships and the predicates among events. So in this paper, we use tree-based query plans for query patterns.

4 Interval-Based Out-of-Order Solution

Our method framework is shown in Figure 2, which includes three components. ”Terminal Layer” is the sources of events. ”Event Buffer” stores the received events from ”Terminal Layer”, and handles some query processing. ”Database Management” conserves historical records, event relationships and some knowledge base rules, as we have introduced in [11].

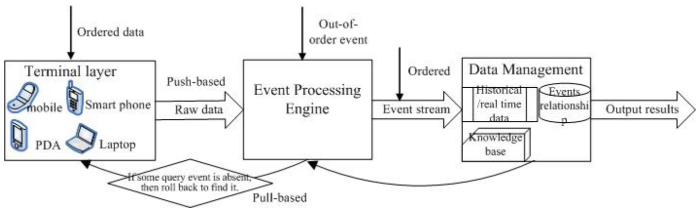


Fig. 2. Interval-based out-of-order solution framework

Besides the framework, specific query expression is also a challenge problem. The expression of interval-based queries may be ambiguous. For example, the same expression query (A ($overlap$) B ($overlap$) C) may have different meanings, as shown in Figure 3. In order to overcome this problem, the hierarchical representation with additional information is needed [10]. Then 5 variables is proposed, namely, contain count c, finish by count f, meet count m, overlap count o, and start count s to differentiate all the possible cases. The representation for a composite event E to include the count variable is shown as follows:

$$E = (\dots(E_1\ R_1[c,f,m,o,s]\ E_2)\ R_2[c,f,m,o,s]\ E_3)\dots\ R_{n-1}[c,f,m,o,s]\ E_n) \quad (1)$$

Thus, the temporal patterns in Figure 3 are represented as:
(A Overlap [0,0,0,1,0] B) Overlap [0,0,0,1,0] C
(A Overlap [0,0,0,1,0] B) Overlap [0,0,0,2,0] C
(A Overlap [0,0,0,1,0] B) Overlap [0,0,1,1,0] C

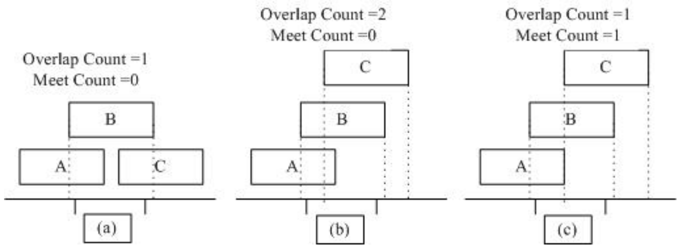


Fig. 3. Interpretation of pattern (A ($Overlap$) B) ($Overlap$) C

In real world, different applications have different requirements for consistency. Some applications require a strict notion of correctness, while others are more concerned with real-time output. So we add an additional attribute (”Real-time Factor”) into every query, as shown in section 3. If the users focus on real-time output, the ”Real-time Factor” is set to ”1”; Otherwise, it is valued as ”0”. Due to users' different requirements of consistency, there are two different methods, which are introduced as follows.

4.1 Real-Time Based Method

If the ”Real-time Factor” of a query is set to ”1”, the goal is to send out results with as small latency as possible. When users submit queries to ”Events Buffer”, which handles the query processing and outputs the corresponding results directly. Once out-of-order data arrival occurs, we provide a mechanism to correct the results that have already been erroneously output. This method is similar to, but better than the method in [6] because of the tree-plan expression, which can reduce the compensation time and frequency, as shown in section 3.

For example, the query is $(A(overlap)B(!D)(before)C)$ within 10 mins. A unique time series expression of this query $\{OS_a, OS_b, OE_a, OE_b, OS_c, OE_c\}$ can be gotten based on the above interval expression method. For the event stream in Figure 4, when an out-of-order seq/pal event $OS_b(6)$ is received, a new correct result $\{OS_a(3), OS_b(6), OE_a(7), OE_b(9), OS_c(11), OE_c(12)\}$ is constructed and output as $<+, \{OS_a(3), OS_b(6), OE_a(7), OE_b(9), OS_c(11), OE_c(12)\}>$. When an out-of-order negative event $OS_d(15)$ is received, a wrong output result $\{OS_a(13), OS_b(16), OE_a(17), OE_b(20), OS_c(22)\}$ is found and send out a compensation $<-, OS_a(13), OS_b(16), OE_a(17), OE_b(20), OS_c(22)>$.

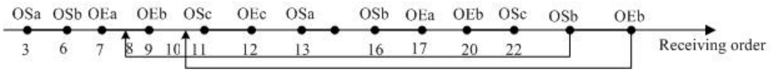


Fig. 4. Input events

4.2 Correct Based Method

If the ”Real-time Factor” of a query is set to ”0”, the goal is to send out every correct result with less concern about the latency. When the user submit a query to ”Events Buffer”, we first extract the corresponding event type. Based on the event model introduced in section 3, we can get the event sequence by a backward and forward depth first search in the DAG. Meanwhile, we can transform the query into a certain time series based on the above 5 variables, which make the representation of relationships among events unique. Compared with the time series of the query, the set of event sequence can be further filtered.

The buffer in the ”Database Management” is proposed for event buffer and purging using the interval-based K-ISlack semantics. It means that both the start time and the end time of the out-of-order event arrivals is within a range of K time units. A CLOCK value which equals to the largest end time seen so far for the received events is maintained. The CLOCK value is updated constantly. According to the sliding window semantics, for any event instance e_i kept in the buffer, it can be purged from the stack if $(e_i.starttime + W) < CLOCK$. Thus, under the out-of-order assumption, the above condition will be $(e_i.starttime + W + K) < CLOCK$. This is because after waiting for K time units, no out-of-order event with start time less than $(e_i.starttime + W)$ can arrive. Thus e_i can no longer contribute to forming a new candidate sequence.

5 Experiments

Our experiment involves two parts: one is the event generator; another is the event process engine. The event generator is used for generating different types of events continuously. The event process engine includes two units: the receiver unit and the query unit.

The experiments are run on two machines, which CPU is 2.0GHz and RAM is 2.0G and 3.0G respectively. PC1 is used for running the Event Generator programs and PC2 for the Event Process Engine. In order to make the experimental results more credible, we run the program for 300 times, and take average value of all results. In the following, we will examine key performance metrics. $P_{io^3}$is varied from 0% to 45%, and in order to be simple, the effects of window buffer size will be considered in our future work.

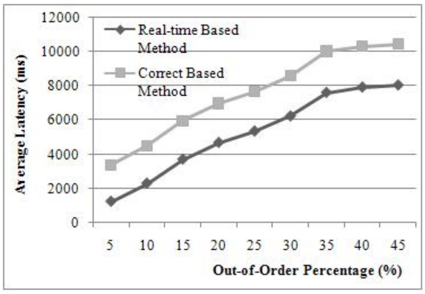


Fig. 5. Trend of Average Latency

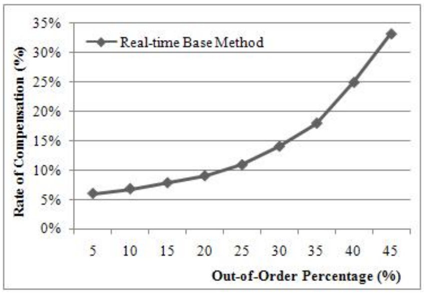


Fig. 6. Trend of Rate-of-Compensation

Figure 5 shows that the average event latency of both methods increases with the increase of out-of-order percentage, and the average latency of Correct Based Method increases faster than Realtime Based Method.

Figure 6 is only in connection with Realtime Based Method, which has compensation operations. The rate of compensation is determined by (NoC/NoR). From the figure, we can see the increasing out-of-order percentage leads to more compensation operations to be generated, which reduce the efficiency of algorithm.

We also experiment accuracy of results. In Figure 7, we can see the result accuracy of Correct Based Method is independent of out-of-order percentage, while the result accuracy of Realtime Based Method drops with the increase of out-of-order percentage.

We examine the average excution time in Figure 8, which denotes the summation of operator execution times. From the figure, two observations can be found: 1) the average excution time increased as the out-of-order event percentage increases for more

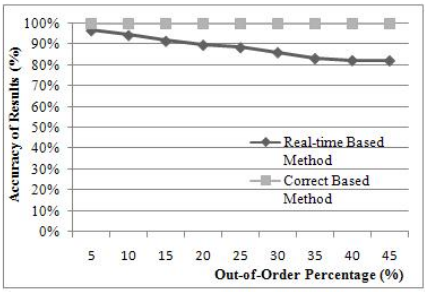


Fig. 7. Accuracy of Methods

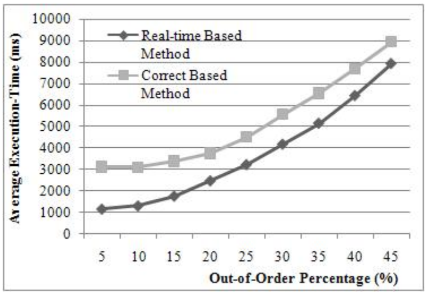


Fig. 8. Trend of Average Execution-Time

recomputing is needed; 2) the average execution time of Correct Based Method is larger than Realtime Based Method at beginning, while with the increase of out-of-order percentage, they will trend to the same.

6 Conclusion and Future Work

The goal of this work is to solve query processing of interval-based out-of-order events in pervasive computing. We analyze the preliminaries, propose a model of interval-based out-of-order events, and a hybrid solution including time-interval is also introduced. In the future work, we will consider other influence factors, including the size of buffer, the out-of-order percentage, the average step-length of out-of-order events, in order to find the balance point.

Acknowledgement

We would like to thank Pengfei Dai of Beijing University of Posts and Telecommunications for his helpful comments in the experiments.

References

1. Pei, J., Han, J., Mortazavi, B., Pinto, H., Chen, Q.: Prefixspan: Mining Sequential Patterns Efficiently by Prefix-projected Pattern Growth. In: Proceedings of the 17th International Conference on Data Engineering (ICDE), pp. 215–226 (2001)
2. Babu, S., et al.: Exploiting K-constraints to Reduce Memory Overhead in Continuous Queries over Data Streams. ACM Transaction on Database Systems 29(3), 545–580 (2004)
3. Wu, E., Diao, Y., Rizvi, S.: High Performance Complex Event Processing over Streams. In: Proceedings of the 32nd SIGMOD International Conference on Management of Data (SIGMOD), pp. 407–418 (2006)
4. Mei, Y., Madden, S.: ZStream: a Cost-based Query Processor for Adaptively Detecting Composite Events. In: Proceedings of the 35th SIGMOD International Conference on Management of Data (SIGMOD), pp. 193–206 (2009)

5. Alex, D., Robert, R., Subrahmanian, V.S.: Probabilistic Temporal Databases. ACM Transaction on Database Systems 26(1), 41–95 (2001)
6. Liu, M., Li, M., Golovnya, D., Rundenstriner, E.A., Claypool, K.: Sequence Pattern Query Processing over Out-of-Order Event Streams. In: Proceedings of the 25th International Conference on Data Engineering (ICDE), pp. 274–295 (2009)
7. Kam, P.S., Fu, A.W.: Discovering Temporal Patterns for Interval-based Events. In: Kambayashi, Y., Mohania, M., Tjoa, A.M. (eds.) DaWaK 2000. LNCS, vol. 1874, pp. 317–326. Springer, Heidelberg (2000)
8. Papapetrou, P., Kollios, G., Sclaroff, S., Gunopulos, D.: Discovering Frequent Arrangements of Temporal Intervals. In: Proceedings of the 5th IEEE International Conference on Data Mining, ICDM (2005)
9. Wu, S., Chen, Y.: Mining Nonambiguous Temporal Patterns for Interval-based Events. IEEE Transactions on Knowledge and Data Engineering 19(6), 742–758 (2007)
10. Patel, D., Hsu, W., Lee, M.L.: Mining Relationships among Interval-based Events for Classification. In: Proceedings of the 34th SIGMOD International Conference on Management of Data (SIGMOD), pp. 393–404 (2008)
11. Zhou, C.J., Meng, X.F.: A Framework of Complex Event Detection and Operation in Pervasive Computing. In: The PhD Workshop on Innovative Database Research, IDAR (2009)

Peer-to-Peer Similarity Search Based on M-Tree Indexing

Akrivi Vlachou[1,⋆], Christos Doulkeridis[1,⋆], and Yannis Kotidis[2]

[1] Dept.of Computer and Information Science, Norwegian University of Science and Technology
[2] Dept.of Informatics, Athens University of Economics and Business
{vlachou,cdoulk}@idi.ntnu.no, kotidis@aueb.gr

Abstract. Similarity search in metric spaces has several important applications both in centralized and distributed environments. In centralized applications, such as similarity-based image retrieval, usually a server indexes its data with a state-of-the-art centralized metric indexing technique, such as the M-Tree. In this paper, we propose a framework for distributed similarity search, where each participating peer stores its own data autonomously, under the assumption that data is indexed locally by peers using M-Trees. In order to support scalability and efficiency of search, we adopt a super-peer architecture, where super-peers are responsible for query routing. We propose the construction of metric routing indices suitable for distributed similarity search in metric spaces. We study the performance of the proposed framework using both synthetic and real data.

1 Introduction

Similarity search in metric spaces has received significant attention in centralized settings [1,6], but also recently in decentralized environments [3,5,8]. A prominent application is distributed search for multimedia content, such as images, video or plain text. Existing approaches for P2P metric-based similarity search mainly rely on a structured P2P overlay, which is used to intentionally store objects to peers [5,8]. The aim is to achieve high parallelism and share the high processing cost over a set of cooperative computers. In contrast, in this paper we focus on the scenario of autonomous peers that store multimedia content and collaborate in order to process similarity queries over distributed data. A P2P architecture for image retrieval was proposed in [9]. In more details, content providers are simple peers that keep multimedia content, usually generated on their own. Each peer joins the collaborative search engine, by connecting to one of the information brokers that act as super-peers, using the basic bootstrapping protocol. In this scenario the super-peers are responsible for the execution of the queries. In such a distributed search engine, the objective is to find all objects that are similar to a given query object, such as a digital image or a text document. Objects are represented in a high dimensional feature space and a metric distance function defines the similarity of two objects. One of the most commonly used centralized indexing techniques for searching in metric spaces is the M-Tree [2] that consists of a hierarchy of hyper-spheres.

[⋆] This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme.

In this paper, we propose a distributed search mechanism that relies on a super-peer architecture, assuming that cooperative peers store and index their data using an M-Tree in an autonomous manner. Each peer connects to a super-peer and publishes the set of hyper-spheres stored at the root of its M-Tree to its super-peer, as a summarization of the stored data. The super-peers store the collected hyper-spheres using an M-Tree index, in order to direct queries only to relevant peers efficiently, thus establishing a *peer selection mechanism*. Capitalizing on their local metric index structures, super-peers exchange summary information to construct metric-based routing indices, which improve the performance of query routing significantly. Then, given a range query, this *super-peer selection mechanism* enables efficient query routing only to that subset of super-peers that are responsible for peers with relevant query results.

Following the same spirit, in SIMPEER [3], P2P metric-based indexing is supported using the iDistance [7] technique. An extension of SIMPEER for recall-based range queries is presented in [4]. In contrast, this paper provides an alternative technique for similarity search in metric spaces, based on a popular metric index (M-Tree) for data access both on peers and super-peers.

2 Preliminaries

We assume an unstructured P2P network that consists of N_p peers. Some peers have special roles, due to their enhanced features, such as availability, stability, storage capability and bandwidth capacity. These peers are called super-peers SP_i ($i = 1..N_{sp}$), and they constitute only a small fraction of the peers in the network, i.e. $N_{sp} << N_p$. Peers that join the network connect to one of the super-peers directly. Each super-peer maintains links to peers, based on the value of its degree parameter DEG_p, which is the number of peers that it is connected to. In addition, a super-peer is connected to a limited set of at most DEG_{sp} other super-peers ($DEG_{sp} < DEG_p$). In our system, peers that join the network autonomously store their own data. Each peer P_i holds n_i d-dimensional points, denoted as a set S_i ($1 \leq i \leq N_p$). Assuming horizontal data distribution to the N_p peers, the size of the complete set of points is $n = \sum_{i=1}^{N_p} n_i$ and the dataset S is the union of all peers' datasets S_i ($S = \cup S_i$). Each peer maintains its own data objects, while the d-dimensional points are features extracted from the objects.

Similarity search in metric spaces focuses on supporting queries that retrieve objects similar to a query point, when a metric distance function $dist$ measures the objects' (dis)similarity. More formally, a metric space is a pair $M = (\Delta, dist)$, where Δ is a domain of feature values and $dist$ is a distance function with the following properties: 1) $dist(p, q) > 0$, $q \neq p$ and $dist(p, p) = 0$ (non negativity), 2) $dist(p, q) = dist(q, p)$ (symmetry), and 3) $dist(p, q) \leq dist(p, o) + dist(o, q)$ (triangle inequality). Similarity search in metric spaces involves two different types of queries, namely *range* and *nearest neighbor* queries. In this paper, we focus on range queries since k-NN queries can be transformed to range queries, if the distance of the k-th nearest neighbor is known. Radius estimation techniques for distributed nearest neighbor search have been studied in [3].

The M-Tree [2] is a distance-based indexing method, suitable for disk-based implementation. An M-Tree can be seen as a hierarchy of metric regions, also known as hyper-spheres or balls. More precisely, all the objects being indexed are referenced in the leaf nodes, while an entry in a non-leaf node stores a pointer to a node at the next lower level along with summary information about the objects in the subtree being pointed at. The objects in the internal nodes are database objects that are chosen (during the insertion) as representative points. For a non-leaf node N, the entries are quad-tuples $\{(p, r(p)), D, T\}$, where p is an representative object, $r(p)$ is the corresponding covering radius, D is a distance value, and T is a reference to a child node of N. The basic property is that for all objects o in the subtree rooted at T, we have $dist(p, o) \leq r(p)$. For each non-root node N, let object p' be the parent object, i.e. the object in the entry pointing to N. The distance value stored in D is the distance $dist(p, p')$ between p and the parent object p' of N. These parent distances allow more efficient pruning during search than would otherwise be possible. Similarly, for a leaf node N, the entries consist of pairs of the form (o, D), where o is a data object and D is the distance between o and the parent object of N.

3 Metric-Based Routing Indices

In our framework, each peer P_i that connects to a super-peer SP_j publishes a summary of its data, in order to make its content searchable by other peers. In our framework, we take advantage of the existing M-Tree index and each peer P_i publishes to its responsible super-peer SP_j, the hyper-spheres contained in the root of the M-Tree, as a summary of the stored data. This set of hyper-spheres covers all data objects stored at P_i, thus SP_j is able to determine if P_i stores data relevant to a potential range query, by searching for hyper-spheres that overlap with the query. SP_j needs to support efficient retrieval of peer hyper-spheres, and consequently selection of the peers that store relevant data to a similarity query. For this purpose, SP_j inserts the collected hyper-spheres into a local M-Tree, also mentioned as *super-peer M-Tree*.

The remaining challenge is to construct routing indices for processing similarity queries over the entire super-peer network. For this purpose, each super-peer maintains an M-Tree, also called *routing M-Tree*, to store hyper-spheres (collected from other super-peers) that describe the data accessible through each neighbor in the super-peer topology. A super-peer SP_i sends the descriptions of the hyper-spheres contained in the root of the super-peer M-Tree to its neighbors. This message has the following format: $(msgId, \{(p_i, r(p_i))\})$, where $msgId$ is an identifier that is unique for each SP_i, and $\{(p_i, r(p_i))\}$ represents the set of SP_i's hyper-spheres corresponding to the root of SP_i's M-Tree. Each hyper-sphere is defined by a representative object p_i and the corresponding covering radius $r(p_i)$. Each neighboring super-peer SP_j that receives a set of hyper-spheres for the first time performs two operations. First, SP_j stores locally the hyper-spheres in the routing M-Tree and attaches to them the identifier of the neighboring super-peer SP_i, from which the hyper-spheres were received. Second, SP_j propagates the hyper-spheres to all its neighbors, except for the one it received them from (SP_i). Any super-peer SP_k that is contacted by SP_j performs the same operations. However, notice that SP_k stores in its routing M-Tree the identifier of its neighbor SP_j together with the hyper-spheres, and not the identifier of the owner super-peer SP_i.

This construction protocol works also for network topologies that contain cycles. Since hyper-spheres of any super-peer SP_i are accompanied by a unique $msgId$, each recipient super-peer SP_k can perform duplicate elimination, in case SP_i's hyper-spheres are also received from a different network path. Notice that the granularity of the routing information stored at any super-peer is at the level of its neighbors $O(DEG_{sp})$ and not at the level of the network $O(N_{sp})$. Therefore, the constructed routing indices are scalable with network size.

We now elaborate more on the internal structure of nodes in the routing M-Tree. For internal nodes, the routing M-Tree entry is $\{(p, r(p)), D, T\}$, where $(p, r(p))$ is the representative object p and its covering radius $r(p)$, D is the distance to the parent object and T is a reference to a subtree. For leaf nodes, the routing M-Tree entry is $\{(p, r(p), SP(p)), D\}$, where $(p, r(p), SP(p))$ consists of the representative object p, its covering radius $r(p)$, and $SP(p)$ is the neighbor super-peer responsible for the hyper-sphere, whereas D is the distance to the parent object.

4 Metric-Based Similarity Search

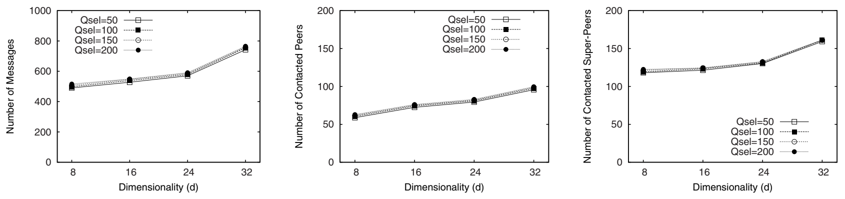
Our framework creates routing M-Trees on each super-peer and supports efficient query processing, in terms of local computation costs, communication costs and overall response time. A query may be posed by any peer P_q and is propagated to the associated super-peer SP_q, which becomes responsible for local query processing and query routing, and finally returns the result set to P_q.

4.1 Super-Peer Local Query Processing

Given a range query $R(q, r)$, query processing at SP_i is performed by exploiting the summary information stored in the super-peer M-Tree. The aim is to retrieve the subset of local peers that need to be contacted. The peers that store data enclosed in the range query $R(q, r)$ have to be contacted, since these results are necessary to be retrieved and reported back to SP_q, in order to form the exact and complete result set. Therefore, SP_i uses its super-peer M-Tree to identify hyper-spheres of peers that intersect with the query. Recall that the retrieved hyper-spheres contain the peer identifier of the owner peer. Thus, the subset of peers that can contribute to the query result is determined and the range query is forwarded to the corresponding peers. This enables efficient similarity search over all data stored by peers associated to SP_i, since the query is posed only to peers having data that may appear in the result set, essentially forming an effective *peer selection mechanism* at a super-peer. Each recipient peer processes the query using its local M-Tree, in the traditional way of processing range queries in M-Trees. Consequently, each peer reports its results to SP_i, which in turn is responsible for returning the results to SP_q.

4.2 Query Routing

After having described the local query processing on each super-peer, we proceed to present the details on query routing at super-peer level. Henceforth, we assume that each



(a) Number of messages (b) Contacted peers (c) Contacted super-peers

Fig. 1. Scalability with dimensionality for clustered dataset

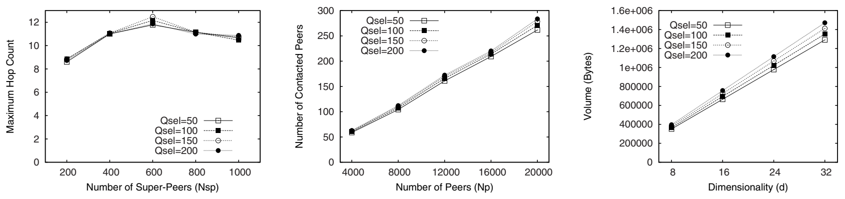
super-peer that receives the query also performs local query processing, as described above. Given a range query $R(q, r)$, the querying super-peer SP_q needs to selectively propagate the query to a fraction of its neighboring super-peers and each intermediate super-peer SP_i that receives the query repeats the same process. The routing algorithm on any super-peer SP_i is based on its routing M-Tree. When a super-peer SP_r receives a range query $R(q, r)$, SP_r uses the routing M-Tree to efficiently retrieve all hyperspheres that have an overlap with the query. Then, the set of neighbor super-peers is determined and the query is forwarded to them only. This forms the *super-peer selection mechanism* that enables routing of queries at super-peer level. Afterwards, the relevant data is collected and sent back to the neighboring super-peer from which the query was received. Finally, SP_q collects all results of its neighboring super-peers and sends the result set back to the peer P_q that posed the query.

5 Experimental Evaluation

In order to evaluate the performance of our approach, we implemented a simulator prototype in Java. For the P2P network topology, we used the GT-ITM topology generator[1] to create well-connected random graphs of N_{sp} peers with a user-specified average connectivity (DEG_{sp}). We used synthetic data collections, in order to study the scalability of our approach. The uniform and clustered datasets are generated as described in [3] and they are horizontally partitioned evenly among the peers by keeping n/N_p=1000 in all setups. Additionally, we employed a real data collection (VEC), which consists of 1M 45-dimensional vectors of color image features. In all cases, we generate 100 queries uniformly distributed and we show the average values. For each query, a peer initiator is randomly selected. Although different metric distance functions can be supported, in this set of experiments we used the Euclidean distance function. We measure the: (i) number of messages, (ii) volume of transferred data, (iii) number of transferred objects, (iv) maximum hop count, (v) number of contacted peers, (vi) number of contacted super-peers, and (vii) response time.

Initially, we focus on the case of clustered dataset. We use a default setup of: N_{sp}= 200, N_p=4000, DEG_{sp}=4, n=4M, and the selectivity of range queries ranges from 50 to 200 objects. We study the effect of increasing dimensionality d to our approach. In

[1] Available at: http://www.cc.gatech.edu/projects/gtitm/



(a) Scalability with N_{sp} (b) Scalability with N_p (c) Uniform data

Fig. 2. Scalability for clustered and uniform dataset

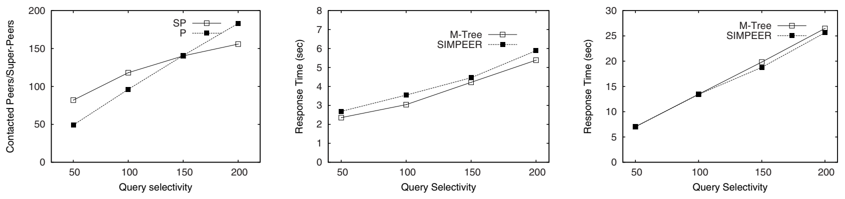
Fig. 1(a), the number of messages required for searching increases when the dimensionality increases. Then, in Fig. 1(b) and 1(c), we measure the number of contacted peers and super-peers respectively. Although the number of super-peers that process the query is between 120 and 150, the number of peers is much lower, ranging from 60 to 100 peers.

In the following, we study the scalability of our approach with respect to the network parameters, by fixing d=8. For this purpose, we increase the number of super-peers N_{sp} (Fig. 2(a)) and peers N_p (Fig. 2(b)). We observe that the maximum hop count increases only slightly, always remaining below 12, when the number of super-peers is increased by a factor of 5. On the other hand, in Fig. 2(b), the increasing number of peers only affects the number of contacted peers, however the increase is only marginal compared to the network size.

In Fig. 2(c), we examine the case of uniform data. Clearly, this is a hard case for our approach, as a query may in worst case have to contact all peers, in order to retrieve the correct results. This actually occurs in our experiments, causing also a large number of messages to be sent. We show the volume of transferred data in Fig. 2(c). Compared to the case of the clustered dataset, the total volume transferred increases by a factor of 3-4. However, when the maximum hop count is measured, this value is small (equal to 6, for d=8-32), even smaller than in the case of clustered dataset, since the probability of finding the results in smaller distance increases.

In addition, we evaluate our approach using the real dataset (VEC). We used a network of 200 super-peers and 1000 peers, thus each peer stores 1000 data points. In Fig. 3(a), the number of contacted peers and super-peers are depicted for increasing query selectivity from 50 to 200 points. The results are comparable to the case of the clustered dataset, but slightly worse, as the VEC dataset is not clustered. However, notice that the absolute numbers are comparable to the results obtained using the synthetic dataset, which is a strong argument in favor of the feasibility of our approach.

Finally, we study the comparative performance of the proposed framework to SIMPEER [3]. We performed a set of experiments using both approaches, assuming a modest 4KB/sec as network transfer rate. In the case of the uniform dataset, our framework outperforms SIMPEER in terms of response time, as depicted in Fig. 3(b). In contrast, when a clustered dataset is used, SIMPEER is marginally better than our framework, as shown in Fig. 3(c). For the clustered dataset, SIMPEER is able to accurately discover the underlying clusters in the data, resulting in better performance. When the data distribution is uniform, our framework based on M-Trees is more efficient than



(a) Real data (b) Uniform data (c) Clustered data

Fig. 3. Real data and comparison to SIMPEER in terms of response time

SIMPEER, since the performance of our metric-based routing indices is not influenced by the absence of a clustering structure in the data.

6 Conclusions

Similarity search in metric spaces has several applications, such as image retrieval. In such applications that require similarity search in metric spaces, usually a server indexes its data with a state-of-the-art centralized metric indexing technique, such as the M-Tree. In this paper, we study the challenging problem of supporting efficient similarity queries over distributed data in a P2P system. The experimental results show that our approach performs efficiently in all cases, while the performance of our framework scales with all network and dataset parameters.

References

1. Chavez, E., Navarro, G., Baeza-Yates, R., Marroquin, J.L.: Searching in metric spaces. ACM Computing Surveys (CSUR) 33(3), 273–321 (2001)
2. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: Proc. of VLDB, pp. 426–435 (1997)
3. Doulkeridis, C., Vlachou, A., Kotidis, Y., Vazirgiannis, M.: Peer-to-peer similarity search in metric spaces. In: Proc. of VLDB, pp. 986–997 (2007)
4. Doulkeridis, C., Vlachou, A., Kotidis, Y., Vazirgiannis, M.: Efficient range query processing in metric spaces over highly distributed data. Distributed and Parallel Databases 26(2-3), 155–180 (2009)
5. Falchi, F., Gennaro, C., Zezula, P.: A content-addressable network for similarity search in metric spaces. In: Moro, G., Bergamaschi, S., Joseph, S., Morin, J.-H., Ouksel, A.M. (eds.) DBISP2P 2005 and DBISP2P 2006. LNCS, vol. 4125, pp. 98–110. Springer, Heidelberg (2007)
6. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces. ACM Transactions on Database Systems (TODS) 28(4), 517–580 (2003)
7. Jagadish, H.V., Ooi, B.C., Tan, K.-L., Yu, C., Zhang, R.: iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. ACM Transactions on Database Systems (TODS) 30(2), 364–397 (2005)
8. Novak, D., Zezula, P.: M-Chord: a scalable distributed similarity search structure. In: Proc. of InfoScale, p. 19 (2006)
9. Vlachou, A., Doulkeridis, C., Mavroeidis, D., Vazirgiannis, M.: Designing a peer-to-peer architecture for distributed image retrieval. In: Boujemaa, N., Detyniecki, M., Nürnberger, A. (eds.) AMR 2007. LNCS, vol. 4918, pp. 182–195. Springer, Heidelberg (2008)

Update Migration: An Efficient B+ Tree for Flash Storage*

Chang Xu, Lidan Shou, Gang Chen, Cheng Yan, and Tianlei Hu

College of Computer Science department, Zhe Jiang University
Hang zhou, China
chinawraith@163.com,
{should,cg,yancheng,htl}@zju.edu.cn

Abstract. More and more evidence indicates that flash storage is a potential substitute for magnetic disk in the foreseeable future. Due to the high-speed random reads, flash storage could improve the performance of DBMS significantly in OLTP applications. However, previous research has shown that small-to-moderate random overwrites on flash are particularly expensive, which implies that the conventional DBMS is not ready to run on the flash storage. In this paper, we propose the design of a variant of B+ tree for flash storage, namely the Update-Migration B+ tree. In the UM-B+ tree, small quantity of updates will be migrated, rather than being executed directly, to its parent node in the form of update records when a dirty node is evicted from main memory. Further accesses to the child node will cause the update records stored in the parent node to be executed when reading the child node from the permanent storage (flash). We propose the detailed structure and operations of UM-B+ tree. We also discuss expanding the UM-B+ tree to the transaction system based on the *Aries/IM*. Experiments confirm that our proposed UM-B+ tree significantly reduces the random overwrites of B+ tree in a typical OLTP workloads, therefore securing a significant performance improvement on flash storage.

1 Introduction

In the past years, the continuous increase in the capacity of flash memory has been driving flash storage devices to a wide spectrum of applications. This trend is likely to continue in the next few years [1] [2]. The advantages of flash storage include fast random-read, low energy consumption, shock resistance, and silent working. These make the flash-type disk, also known as the *solid state disk* (SSD), a promising alternative to the magnetic hard disk. As a natural consequence, the database community has been considering the possibility of building flash-based database management systems in recent years.

* This work was supported in part by the National Science Foundation of China (NSFC Grant No. 60803003, 60970124) and by Chang-Jiang Scholars and Innovative Research Grant (IRT0652) at Zhejiang University.

When tackling flash storage, the most obvious issue to address is the expense of overwrites, as an overwrite needs to *erase* a large consistent area on the flash in the first place. While the *erase* operation is extremely slow (as shown in Table 1), the total number of erases on a flash chip determines its life expectancy as well. Though some mapping techniques like Flash Translation Layer (FTL) have been consolidated in flash storage devices to relieve this situation, recent studies [3,4] have revealed that small-to-moderate random overwrites in a large area would cause performance degradation and should be avoided.

Table 1. Operation Costs of Samsung K9XXG08UXM flash chip

Read	Write	Erase
25us	200us	1.5ms

Nevertheless, random overwrites are frequent operations in the transaction processing workload of a conventional DBMS. In particular, considering a most common index, the B+ tree, the updates to the index cause random overwrites whenever a node is evicted from the memory buffer of the index. As a result, the updates to a flash-based implementation of a conventional B+ tree are characterized by a large number of slow random overwrites.

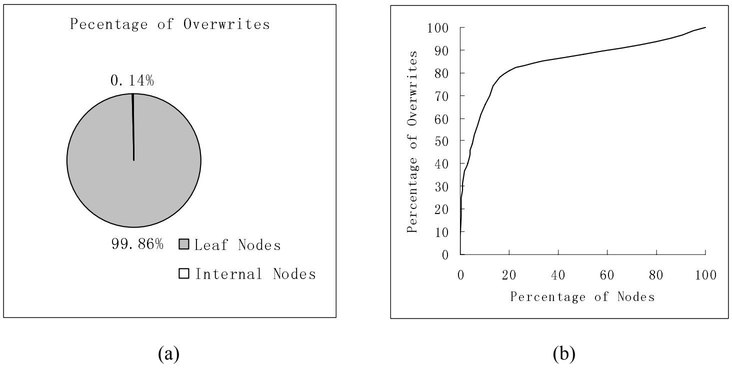


Fig. 1. Overwrite statistics of TPC-C on Mysql

In this paper, we propose the design of a purely flash-based B+ tree indexing scheme, namely the UM-B+ tree, for OLTP workloads. Our work is motivated by the following observations from the conventional B+ tree: First, we observe that most internal nodes are memory-resident and most updates to a B+ tree happen at individual leaf nodes. Second, updates to the index has locality. Nodes tend to be repetitively updated. To verify these two observations, we conduct an experiment using the TPC-C benchmark on the *MySql* DBMS with *innoDB* engine.[1]

[1] The scale of the data is 10 warehouses and the buffer size is 1/10 of the physical data.

Figure 1.(a) indicates that 99.86% of the write operations to the B+ tree indexes are on their leaf nodes, while Figure 1.(b) indicates that about 80% of the write operations concentrate on 20% of the nodes. Third, as an obvious fact, a child node is always located from the parent during the traversal of the B+ tree. This motivates the idea of "saving" the updates on the leaf nodes to their ancestor nodes in the tree. In the UM-B+ tree, each internal node provides some space for storing the update records of its child nodes. When a child node is evicted from the main memory buffer, we try to "migrate" the new updates on the child node to its parent node. In contrast, when a child node is read into memory, these migrated updates execute to the memory copy of the child node during the locating. In this way, small number of updates are cached in the memory and commited to the flash-resident node efficiently. The most desirable property of the UM-B+ tree is that it has nearly the same topology as a conventional B+ tree and no additional main memory data structures need to be employed. Therefore, all existing algorithms with B+ tree such as SMO and concurrency control algorithms can be reused on the UM-B+ tree at no extra cost.

The rest of the paper is organized as follows. Section 2 introduces the related work about indexes in flash storage. We describe our UM-B+ tree structure in Section 3 and 4.2 and discuss the high availability issues in Section 5. Section 6 presents our experimental results.

2 Related Work

The past years have witnessed a handful of studies on the indexes for flash storage. For example, ISBF [6], BFTL [5] and RBFTL [7] all use a reservation buffer to store recent updates and flush them into the flash storage in a batch. The LA-tree [10] uses cascaded buffers instead of a central one according to the tree structure and tune the buffer adaptively to get the best performance. Unfortunately, the extra buffer(s) significantly increase the complexity of the system, as we need to manage the updates in the buffers independently, and it is especially difficult to do so in a highly concurrent environment. The in-page logging(*IPL*) technique [3][9] divides an erase block to two parts – one page is used to log the updates of the other pages. But this approach is not suitable for tree structures as it manages the updates in a physically consistent scale. The FD-tree proposed in [8] has evolved from the LSM tree[11]. It divides the tree into two parts: a memory-resident small one to store new updates and a disk-resident large one to hold persistent records. After the memory-resident tree overflows, it is merged into the disk-resident one in a bulk process. The FD-tree exploits the high-speed sequential operation on SSD. However, it is inflexible as, for example, a merge operation may block all the other users until it completes.

Compared with the previous work, our approach has the following advantages:

– First, as the update records on a leaf node are saved in its ancestor, we do not need to maintain any extra buffers. Saving update records in an ancestor node has the performance benefit that in case a leaf node is to be accessed, its migrated update records must have already been accessed during the top-down traversal of the B+ tree.

– Second, the UM-B+ tree is fully compliant with the B+ tree. This indicates that the transactional protocol of B+ tree could be reused at trivial cost. Therefore, high availability of the UM-B+ tree index can be guaranteed.
– Third, as many of the previous works partially optimizes for large mount of updates on flash storage, our proposed UM-B+ tree could achieve good performance in both update-intensive and search-intensive workloads. This indicates that UM-B+ tree fits to typical OLTP workloads, in which many read-only transactions and a few update transactions work concurrently.

3 Overview

In this section we shall give a brief overview to the proposed UM-B+ tree scheme.

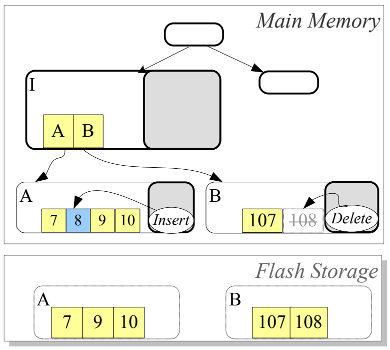
3.1 Preliminaries

First we describe the preliminaries for the rest of this paper. Consider a conventional B+ tree implemented for a flash storage, each *internal node* of the tree contains *index entries* as an ordered list of $< key, pointer >$ tuples. Entries in a subtree under *pointer* have key values within the range defined by *key* and its subsequent key *next key*. A *leaf node* contains entries of a key and its corresponding record id(rid). Two neighboring leaf nodes are connected by a bi-direction pointer. There are two major access types. A *point search* locates a unique leaf entry from the root node downwards, while a *Range search* visits the consecutive items on the leaf nodes, via the connecters between them. Generally, all the nodes of the B+ tree are persistent in the flash storage. Generally we can assume the existence of a relatively small *buffer* in the main memory to store the tree nodes. In addition, as the fan-out of the B+ tree is typically large, we can make a reasonable assumption that all internal nodes can be loaded into the main memory, though this is not a requirement for both the conventional B+ tree and our proposed UM-B+ tree.

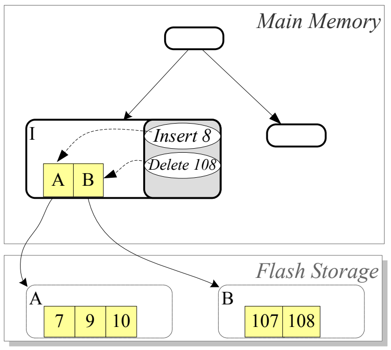
3.2 How Update Migration Works

The structure of the UM-B+ tree is same as the standard B+ tree, except that each node in the UM-B+ tree should allocate a part of its space for accommodating “update records”. A leaf node needs to keep the update operations on itself in its own data area, while an internal nodes need to keep those that are migrated from its children. Once a leaf node is updated and then evicted from the buffer, the updates migrate to its parent.

Figure 2 illustrates an example of how updates in UM-B+ tree work. Suppose that leaf node A is read into the buffer and then inserted by an entry $8*$. An update record $Insert(8*)$ will be saved in A. As Figure 2(b) illustrates, this update record will migrate to A's parent node I when A is evicted from the



(a) *A* is read into memory and inserted with key *8*, *B* is read into memory and deleted key *108*.



(b) after *A&B* are evicted from main memory, the updates migrate to the parent node *I*

Fig. 2. An example of update migration in a UM-B+ tree

buffer. Node A in the flash storage is an outdated version, which we call the *flash version*. If node A is read into the memory buffer again, the update record $Insert(8*)$ will return to it, restoring A to its latest version, namely the *memory version*, containing entry $8*$. Intuitively, the restoring process is efficient as A is always located via node I during the traversal. A similar update migration process can be seen for leaf node B, where entry $108*$ is deleted and an update record $Delete(108*)$ is added. It can be seen that the above update migration saves two node overwrites into the flash storage, one for node A and the other for node B.

4 The UM-B+ Tree

4.1 Node Structure

As mentioned in the previous section, updates on the nodes of UM-B+ tree are recorded as *update records*(UR). There are two types of update records, namely *primary update records* and *migrated update records*. A primary $UR_{primary} =< UT, pos >$ contains the update type UT, and the position(index) pos of the data entry the update occurs, while a migrated update record $UR_{migrated} =< UT, pos, (entry) >$ contains the content of the *entry* being updated as well. The leaf nodes contain primary URs only, while the internal nodes contain migrated URs. When inserting/deleting an entry into/from a leaf node, a primary update record is added to the node. When a primary update record is being migrated, a migrated update record is created and added to the parent node. If the primary update is an insertion, the content of the entry needs to be contained, as it would be lost when the leaf node is evicted from the buffer.

In the UM-B+ tree, each node space is divided into two parts: the *Entry Area* which stores the entries and the *U-Area* which stores the update records. In each leaf node, the *U-Area* is only used to accommodate the *primary* URs of the "delta" updates between its flash version and memory version. We define an upper bound τ_u as the maximum number of "delta" updates on a single leaf, so that the size of *U-Area* on leaf nodes is fixed to $2 * \tau_u$ bytes, which consumes small space. When the number of "delta" updates accumulates to be more than τ_u, the node is overwritten into the flash storage when being evicted. In this case we say a *merge* happens. We also define a lower bound τ_l as the minimum number of "delta" updates. That is to say, if the number of "delta" updates on a leaf node is less than τ_l, the node is not allowed to be merged.

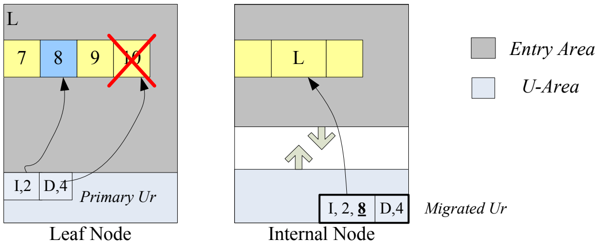


Fig. 3. Node Structure of UM-B+ tree

In an internal node, larger *U-Area* needs to be allocated to accommodate the *migrated URs* from its children. To improve the space utilization, we adopt an adaptive strategy here. Specifically, the *Entry Area* grows forwards while the *U-Area* grows backwards, and the bound between them is tuned adaptively. Initially, we reserved about 1/4 of the internal node for *U-Area* when populating the tree. This will not increase the space consumption significantly as internal nodes only comprise a very small part of the tree. When an internal node splits, the new nodes will have a smaller *Entry Area* and a larger *U-Area*. Figure 3 illustrates the structure of one internal node and one leaf node.

4.2 Operations

We describe the operations of the UM-B+ tree in this subsection. For clearness, we define three types of status for the nodes in buffer, which are *Clean*, *Dirty* and *Merged*. A *Clean* node means the node is as same as the flash version. A *Dirty* node means that the "delta" updates on the node is between 1 and τ_u, while a *Merged* node means that the "delta" updates on the node are more than τ_u. We abbreviate them to "C", "D" and "M".

Update. Update on UM-B+ tree is straightforward. Firstly we reach the proper leaf node N via the *locate* routine. Then the update is executed on N and a *primary UR* is inserted into its *U-Area*. If the *U-Area* overflows, then we mark N as "M". Otherwise, N is marked as "D".

Search. Search on UM-B+ tree includes *range search* and *point search*. We only describe the former one for the *point search* could be seen as a special case of the range search. A *range search* returns a series of entries whose key values satisfy the specified range boundaries. *Range search* firstly *locates* the lower bound. Then it traverses the entries on the leaves one by one until it reaches the upper bound. When crossing to a new leaf N, we need to return the possible "delta" updates if N is read from the flash storage. This could be done via the *locate* routine. The pseudo-code of the RangeSearch is described in Algorithm 1.

Algorithm 1. RangeSearch(*lowthreshold*, *upthreshold*)

```
1: $e,N \leftarrow locate(lowthreshold)$;
2: while $e.key < upthreshold$ do
3:     add $e.recordid$ to $Result$;
4:     if $e$ is the last entry of $N$ then
5:         $N \leftarrow$ next node of $N$;
6:         if $N$ is read from flash storage then
7:             $e \leftarrow$ first entry of $N$; $N \leftarrow Locate(e)$;
8:         end if
9:         $e \leftarrow$ first entry of $N$;
10:    else
11:        $e \leftarrow$ next entry of $N$;
12:    end if
13: end while
```

The cost of crossing between the nodes is a little higher than in the standard B+ tree, as we may need to access the ancestors. But in general cases the process does not produce extra I/Os because the ancestors usually reside in the buffer.

Migrate. The migrate routine is triggered when a node N with status "D" is evicted from the buffer. In most cases N is a leaf , then its *primary* URs are transformed to the *migrated* type then migrate to parent node N_p. If N_p overflows and there exists a child who migrates more than τ_l URs, then we *merge* the child to release the *U-Area*. Otherwise N_p *splits.*

If N is an internal node, the *migrated* URs also attempt to migrate to the parent. The difference is that N is merged directly if there are no enough space. The pseudo-code of migrate routine is described in Algorithm 2.

Locate. Given a key constraint, the locate routine starts from the root then searches the proper path via the key comparison. On each node N, a binary search is done in the *Entry Area* for locating a proper entry. If N is leaf, then the entry is returned. If N is internal, then the child node is visited according to the pointer of the entry. Before visiting a child node N', we need to return the corresponding *migrated* URs to it. As an anti-process of migration, if N' is a leaf node, the URs about N' are transformed into the *primary* type and executed on N again. In this way, leaf N' is restored to the memory version.

Algorithm 2. Migrate(N)

```
1: $N_p \leftarrow Fetch$($N$'s parent);
2: move $UR$s on $N$ into the $U$-$Area$ of $N_p$, mark $N_p$ as 'D';
3: **if** $N_p$ is overflow **then**
4:    **if** $N$ is internal node **then**
5:       $Return(URs, N)$, $merge(N)$;
6:    **else**
7:       $N' \leftarrow$ the child with most $UR$s;
8:       **if** the $UR$s about $N'$ is more than $\tau_l$ **then**
9:          $Return(UR, N')$, $merge(N')$;
10:      **else**
11:         $split(N_p)$;
12:      **end if**
13:   **end if**
14: **end if**
```

Split. When an internal node splits in the UM-B+ tree, we move the rightmost entries one by one into the new node, together with the relevant URs, if any. Unlike B+ tree, we do not distribute the entries evenly but stop the process when the new node is half-full. In this way, we cluster fewer volatile children in one node and more stable ones in another. Because of the adaptive bound between the *Entry Area* and *U-Area*, the possibility of further splits is reduced.

The split process of a leaf node is simple for we just need to distribute the entries averagely. After the split, an entry about the new node is inserted into the parent. All the reference nodes are mark as "M" when the split completes.

5 High Availability

In this section we expand the UM-B+ tree into the transaction system. Because of its fully compliance with B+ tree, the mature protocols [12][13][14] [16] [18] of the B+ tree could be transplanted to the UM-B+ tree at trivial cost. We will describe how to adapt the *Aries/IM* method for our UM-B+ tree. Due to the space limit, we only discuss the modifications on the UM-B+ tree. The details of *Aries/IM* could be found in [14].

5.1 Concurrency Control

The mechanisms for concurrency control include the *Latch* and the *Lock*. The lock scheme of *Aries/IM* could be transplanted directly because: (1) the entry in the leaf node is same as the B+ tree, so that the *rid*-lock is available. (2) whenever we latch a leaf node, it is the memory version and the entries in the *Entry Area* are sorted by the key. Therefore, the *next key* lock scheme is available.

The latch-coupling logic in *Aries/IM* permits at most two nodes to be latched simultaneously at any time. To avoid *dead latch*, parent node could be latched while waiting for the latch of the child during the traversal, but the opposite case is forbidden. In UM-B+ tree, if we find some relevant *migrated* URs of the child in the locate routine, the latch of the current node should be upgraded to *Exclusive* before we latch the child. Then the URs could return safely. Additionally, migration routine should release the leaf node before searching the parent and re-latched it to examine whether it is still the tail of the LRU list. If not, the buffer manager releases the parent and restarts from picking the new tail.

5.2 Recovery

In *Aries* [15], every node records the log sequence number (LSN) of the log record that describes the most recent update to the node, which monotonically increases over time. During recovery, by comparing the $node\_LSN$ with the LSN of a log record for that node, we can unambiguously determine whether the latest version of the node contains that log record's update. If not, the log needs to redo. Compared with the standard B+ tree, the UM-B+ tree has a potential danger for it migrates the updates outside the corresponding node. Because of the "STEAL and NO FORCE" policy, these updates may be flushed into the flash storage in other nodes. This will produce "lost" updates in the flash storage if the database crashes.

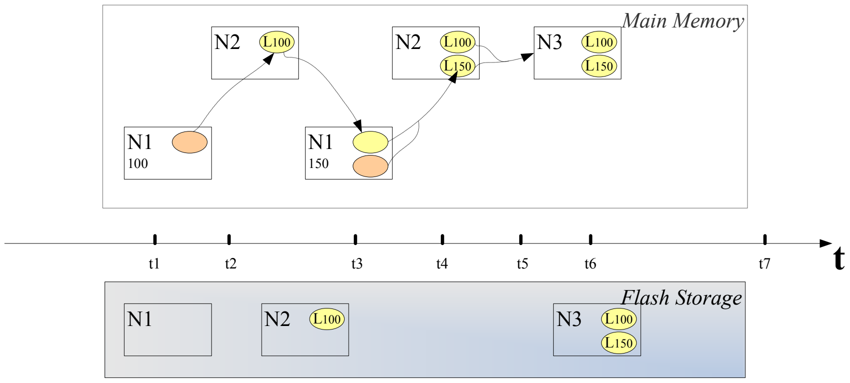


Fig. 4. An example for crashing of UM-B+ tree

The Figure 4 illustrates an example of crash. At t_1 the leaf node N_1 is read into buffer and an update occurs on it. The LSN of the corresponding log is 100. Then N_1 is evicted and the UR migrate to the parent N_2. At t_2 N_2 is merged into the flash storage. At t_3 N_1 is read into buffer again(before that N_2 must be read in and the UR return to N_1, recovering $N1\_LSN$ to 100) and another update with LSN 150 happens. At t_4 N_1 is evicted again and the two URs migrate to N_2. At t_5 N_2 splits and the two URs migrate to the new node N_3. Then N_3 is merged at t_6. Unfortunately the database crashes at $t7$. We can see the flash version of N_1 is still earlier than LSN 100, and there exists one *migrated* UR(with LSN 100) in node N_2 and two *migrated* URs(with LSN 150) in node N_3. That is to say, the flash version of N_1 is not the newest version, because there are some newer updates "lost" in other nodes, which could cause inconsistency after the system restarts.

We address this problem by modifying the migration and the recovery process. Firstly, the *migrate* URs also contains the LSN of the correspond leaf node. That is to say, the LSN of the leaf node migrates with the URs and returns to the node when the URs execute again. Secondly, each migration (includes the migration in an internal splitting) produces a special log, named *migration log*, which consists of the original node id(*ONid*) and the goal node id(GNid). After the migration, the LSN of the goal node is set to the LSN of the migration log.

If database crashes, the *Aries* protocol recovers the data via a 3-phrase process: *analysis*, *redo* and *undo*. During the analysis phrase, we scan the log starting from the record after last checkpoint and build a *migration graph*, which is built from the migration logs. For each migration log, an directional edge from the ONid to the GNid is added into the graph.

During the redo process, whenever a node N is read from flash storage, we traverses the migration graph to check the nodes which are in the successors of N. If there exists some *migrated* URs about N on them, then the one with the newest(largest) LSN returns to N. The left *migrated* URs about N are erased, then N is removed from the migration graph. We forbidden the migration during the redo. Any modified nodes are merged into the flash storage when evicted. In this way any node N could recover to the newest version correctly because:

1) In the flash storage, assume one node N's LSN is L_1 and there exists some *migrated* URs about N with LSN L_2. If $L_1 < L_2$, then the URs with L_2 must consist of the "delta" updates starting from L_1 to L_2. This is an obvious conclusion because whenever N merges, it updates the flash version to the newest.

2) All "lost" updates could be traced from the migration graph. As mentioned above, we set the LSN of the goal node to the LSN of the migration log after each migration. Therefore, whenever the goal node is flushed into the flash storage, the WAL(Write Ahead Log) protocol guarantees the corresponding migration log to be flushed into the permanent storage first.

The undo process rollbacks all the updates which is done by an uncommitted transaction. In Aries/IM undo is treated as normal process. Specifically an undo action rollbacks the changes according to the log and produces a compensate log. The undo process is totally logical. Therefore, the migration mechanism of the UM-B+ tree is permitted and the whole system works as normal.

5.3 Checkpoint

To expedite the recovery, database executes checkpoint routine periodically to flush the modified data into the permanent storage. Conventional checkpoint flushes all the dirty nodes, which would produce large mount of random overwrites in the flash storage. We propose an optimal checkpoint based on the migration mechanism. The process could be described as follows:

1) Sort the nodes in the buffer according to (I)their levels in the tree, (II) their key ranges in the level.

2) Scan the nodes in the order. During the scan, the nodes with status "M" are merged directly. The nodes with status "D" attempt to migrate the URs upwards. An node N is merged only if : (I) the parent of N could not accomodate the URs of N, or (II) N is internal node and the *U-Area* of N is over half-full.

Because we scan the nodes from bottom to root and in a unified direction for each level, the updates in the leaves are merged in the form of *migrated* URs in their ancestors as high as possible. Therefore, the checkpoint routine could be completed in minimal I/Os. Figure 5 illustrates an example of checkpoint routine.

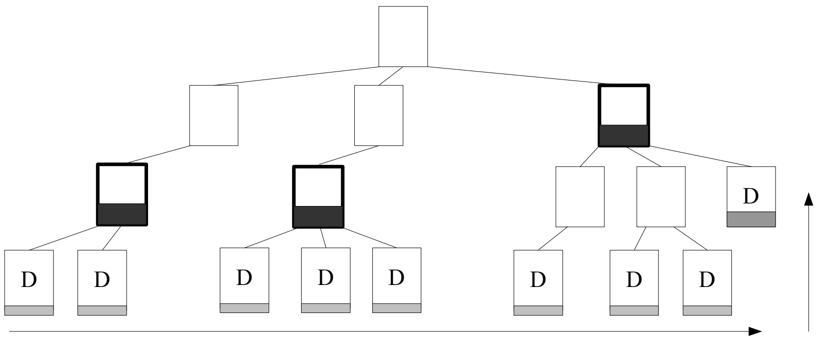


Fig. 5. Checkpoint routine of UM-B+ tree. The "D" nodes are scanned in the direction from left to right and from bottom to top. After the migration, the 9 overwrites of "D" nodes are clustered to 3 overwrites of the ancestors.

6 Experiments

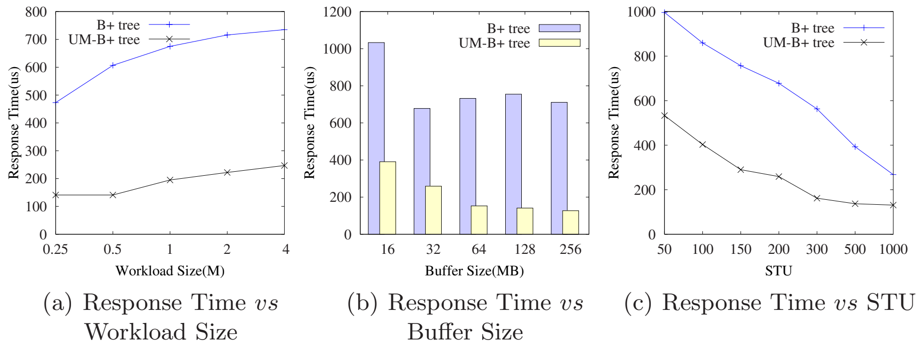
In this section we evaluate the UM-B+ tree against the standard B+ tree, and some other flash-oriented indices. For each scheme, we build a primary index with two attributes with one integer(4 bytes) and one float(4 bytes). The default node size of the all indexes is 4KB. The default τ_u is 16 and τ_l is 8. Initially we populate about 100 million items into each index. In the main memory, we preserve a buffer with the LRU policy. The default size of the buffer is 32MB.

We first evaluate the performance of the UM-B+ tree on a $NAND$ flash chip emulator, since we could not find a direct way to interface the flash chip. The emulator is built on the $DiskSim$ with SSD extension [17] and customized for the Samsung's K9XXG08UXM series NAND-flash part(see Table 1). Then we also evaluate the performance of different indices on a enterprise Solid State Disk. The workloads we used are synthetic, which consist of a sequence of updates and searches. To weight the performance in different cases, we produce different workloads with a various *ratio* between *Search Tasks* and *Update Tasks*(STU). The *Update Tasks* always consists of 60% insertions, 30% updates and 10% deletions. The *Search Task* starts a range search in a scale between 1 to 60 keys. The default number of the workload is 1M and the default STU is 200%. Our experiments are conducted on a Dell PowerEdge R900 server running Windows Server 2003.

6.1 On NAND Chip

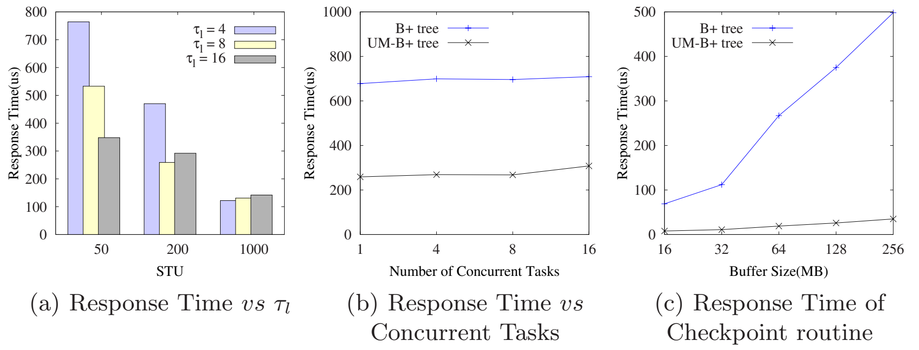
Firstly we compare the performance of UM-B+ tree and standard B+ tree in the NAND chip emulator. In Figure 6.(a), we can see the UM-B+ tree shows a better response time and scalability than the standard B+ tree on various workloads size. That's because larger workload produces more overwrites, which will burden the chip for more translations and reclamations. Figure 6.(b) indicates that B+ tree is not sensitive to the increment of the buffer when it is larger than a threshold(32MB). Respectively, our UM-B+ tree performs better when the buffer size increases. This is intuitively correct for more updates could be cached in the buffer for the UM-B+ tree. Figure 6.(c) illustrates the performance on the various STU, both B+ tree and UM-B+ tree show better performance when the ratio of search comes up. But in any case, UM-B+ tree has a better performance than standard B+ tree.

To further illustrate the effect of the migration mechanism, we tune the parameter τ_l of the UM-B+ tree. A larger τ_l means that we force more updates



(a) Response Time *vs* Workload Size (b) Response Time *vs* Buffer Size (c) Response Time *vs* STU

Fig. 6. Mean Response time of UM-B+ tree and B+ tree on various (a).Workload Size, (b).Buffer Size and (c).STU



(a) Response Time *vs* τ_l (b) Response Time *vs* Concurrent Tasks (c) Response Time of Checkpoint routine

Fig. 7. Mean Response time of UM-B+ tree and B+ tree on various (a).τ_l, (b).Concurrent Tasks(b) and (c).Checkpoint routine

Table 2. Comparison: Random overwrites and Erase Operations of UM-B+ tree and B+ tree

	Random OverWrite			Erase Operation		
	STU=50	STU=200	STU=1000	STU=50	STU=200	STU=1000
B+ tree	833163	425098	118226	83798	40512	7416
UM-B+ tree	328039	110413	19322	29299	6296	0

migrate, at the cost that more splits occur and more internal nodes are produced. In Figure 7.(a), we can see a larger τ_l only benefits in a low STU, for the reason of that in a high STU, the mean response time is determined by the reads of *Search Tasks*. This indicates the τ_l parameter of UM-B+ tree could be tuned according to the I/O characteristics of the system.

Figure 7.(b) indicates the performance of UM-B+ tree in concurrent environment. In this experiment we distribute the tasks to different number of threads. The result indicates that the mean response time of both B+ tree and UM-B+ tree remains while more threads execute concurrently. Figure 7.(c) illustrates the performance of checkpoint routine. We can see that with the enlargement of the buffer, the checkpoint of B+ tree is more expensive for more dirty nodes need to be merged. In the UM-B+ tree, we cluster the updates so that is completes in a much shorter time.

To summarize, when comparing the UM-B+ tree and the standard B+ tree on the flash chip, the result indicates that UM-B+ tree shows better performance for it reduces significant random overwrites. The Table 2 compares the number of random overwrites and erase operation of flash chips between UM-B+ tree and B+ tree. We can see that UM-B+ tree could gain performance improvement as it reduces the random overwrites and erases significantly via the migration mechanism.

6.2 On Raw SSD Device

We also evaluate the performance of our UM-B+ tree on an Intel-25 Extremely SSD. For comparison, we conduct the queries on some flash-oriented indices. In Table 3, we can see the UM-B+ tree outperforms in all the homogeneous indices of B+ tree(includes standard B+ tree, BFTL and IPL). The FD-tree shows comparable performance in our experiments. The reason is that FD-tree

Table 3. Mean Response Time(us) of Different Indices on SSD

	STU=50	STU=100	STU=150	STU=200	STU=500	STU=1000
B+ tree	836	735	673	625	510	455
BFTL	494	633	891	1042	1011	1139
IPL	1790	2092	2314	2101	1917	1642
UM-B+ tree	476	390	303	261	244	231
FD-tree	190	244	289	343	369	386

transforms random overwrites to sequential writes, which is extremely suitable to the SSD. The disadvantage is that FD-tree is heterogeneous to the B+ tree. Therefore, it is difficult to inherit the merits like efficient range search or high concurrency control. We notice that the performance of FD-tree degrades when the ratio of *Search Task* increases. Among all the indices, only the UM-B+ tree is fully compliant with B+ tree, while the other ones are mostly optimized for populating data, as they show poor performance in search-intensive scene.

7 Conclusion

In this paper we propose a variant of the B+ tree, named UM-B+ tree, for flash storages. In UM-B+ tree, the small mount of the updates migrate from leaves to their parents to reduce random overwrites. As the locates in the UM-B+ tree are always from parent to child, the migrated updates return to the leaves efficiently. Because the UM-B+ tree is fully compliant to the B+ tree in the physical structure, we also discuss about expanding the UM-B+ tree to the transaction system, based on the *Aries/IM* protocol. The experiments demonstrate that UM-B+ tree is an efficient substitute of B+ tree in OLTP workloads.

References

1. Gray, J., Fitzgerald, B.: Flash disk opportunity for sever applications. ACM Queue 6(4), 18–23 (2008)
2. Lai, S.K.: Flash memories: Successes and challenges. IBM Journal of Research and Developmen 52(4-5), 529–535 (2008)
3. Lee, S.W., Moon, B.: Design of flash-based dbms: an in-page logging approach. In: SIGMOD Conference, pp. 55–66 (2007)
4. Bouganim, L., Jónsson, B.T., Bonnet, P.: uFLIP: Understanding Flash IO Patterns. In: CIDR (2009)
5. Wu, C.-H., Chang, L.-P., Kuo, T.-W.: An Efficient B-Tree Layer for Flash-Memory Storage Systems. In: Chen, J., Hong, S. (eds.) RTCSA 2003. LNCS, vol. 2968, pp. 409–430. Springer, Heidelberg (2004)
6. Lee, H.-S., Park, S., Song, H.-J., Lee, D.-H.: An Efficient Buffer Management Scheme for Implementing a B-Tree on NAND Flash memory. In: Lee, Y.-H., Kim, H.-N., Kim, J., Park, Y.W., Yang, L.T., Kim, S.W. (eds.) ICESS 2007. LNCS, vol. 4523, pp. 181–192. Springer, Heidelberg (2007)
7. Xiang, X., Yue, L., Liu, Z., Wei, P.: A reliable B-tree implementation over flash memory. In: SAC, pp. 1487–1491 (2008)
8. Li, Y., He, B., Luo, Q., Yi, K.: Tree Indexing on Flash Disks. In: ICDE, pp. 1303–1306 (2009)
9. Na, G.-J., Moon, B., Lee, S.-W.: In-Page Logging B-Tree for Flash Memory. In: DASFAA, pp. 755–758 (2009)
10. Agrawal, D., Ganesan, D., Sitaraman, R., Diao, Y., Singh, S.: Lazy-Adaptive Tree: An Optimized Index Structure for Flash Devices. PVLDB 2(1), 361–372 (2009)
11. O'Neil, P.E., Cheng, E., Gawlick, D., O'Neil, E.J.: The Log-Structured Merge-Tree (LSM-Tree). Acta Inf. 33(4), 351–385 (1996)

12. Mohan, C.: Concurrency Control and Recovery Methods for B+-Tree Indexes: ARIES/KVL and ARIES/IM. In: Performance of Concurrency Control Mechanisms in Centralized Database Systems, pp. 248–306 (1996)
13. Kornacker, M., Mohan, C., Hellerstein, J.M.: Concurrency and Recovery in Generalized Search Trees. In: SIGMOD Conference, pp. 67–72 (1997)
14. Mohan, C., Levine, F.E.: ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging. In: SIGMOD Conference, pp. 371–380 (1992)
15. Mohan, C., Haderle, D.J., Lindsay, B.G., Pirahesh, H., Schwarz, P.M.: ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. ACM Trans. Database Syst. 17(1), 94–162 (1992)
16. Lehman, P., Yao, S.: Efficient locking for concurrent operationson B-trees. ACM Trans Database Sys. 6(4), 650–670 (1981)
17. Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J.D., Manasse, M.S., Panigrahy, R.: Design Tradeoffs for SSD Performance. In: USENIX Annual Technical Conference, pp. 57–70 (2008)
18. Sagiv, Y.: Concurrent Operations on B-Trees with Overtaking. In: PODS, pp. 28–37 (1985)

Optimizing Write Performance for Read Optimized Databases

Jens Krueger[1], Martin Grund[1], Christian Tinnefeld[1], Hasso Plattner[1], Alexander Zeier[1], and Franz Faerber[2]

[1] Hasso–Plattner–Institut
August–Bebel–Str. 88
14482 Potsdam, Germany
[2] SAP AG
Dietmar-Hopp-Allee 16
69190 Walldorf

Abstract. Compression in column-oriented databases has been proven to offer both performance enhancements and reductions in storage consumption. This is especially true for read access as compressed data can directly be processed for query execution.Nevertheless, compression happens to be disadvantageous when it comes to write access due to unavoidable re-compression: write-access requires significantly more data to be read than involved in the particular operation, more tuples may have to be modified depending on the compression algorithm, and table-level locks have to be acquired instead of row-level locks as long as no second version of the data is stored. As an effect the duration of a single modification — both insert and update — limits both throughput and response time significantly. In this paper, we propose to use an additional write-optimized buffer to maintain the delta that in conjunction with the compressed main store represents the current state of the data. This buffer facilitates an uncompressed, column-oriented data structure. To address the mentioned disadvantages of data compression, we trade write-performance for query-performance and memory consumption by using the buffer as an intermediate storage for several modifications which are then populated as a bulk in a merge operation. Hereby, the overhead created by one single re-compression is shared among all recent modifications. We evaluated our implementation inside SAP's in memory column store. We then analyze the different parameters influencing the merge process, and make a complexity analysis. Finally, we show optimizations regarding resource consumption and merge duration.

1 Introduction

Nowadays, enterprise data management database systems are classified being optimized either for online transaction processing (OLTP) or online analytical processing (OLAP). In fact, enterprise applications today are primarily focused on the day-to-day transaction processing needed to run the business while the analytical processing necessary to understand and manage the business is added on after the fact. In contrast to this classification, single applications such as Available-To-Promise (ATP) or Demand Planning exist, which cannot be exclusively referred to one or the other workload category. These application initiate a mixed workload in terms of that they process small sets of transactional data at a time including write operations as well as complex, unpredictable mostly-read queries on large sets of data. Having a mixed workload is nothing new - the insight that it is originated by a single application is new. Given this and the fact that databases are either build for OLTP or OLAP, it is evident that there is no DBMS that adequately addresses the needed characteristics for these complex enterprise applications. For example, within sales order processing systems, the decision of being able to deliver the product at the requested time relies on the ATP check. The execution of this results in a confirmation for the sales order containing information about the product quantity and the delivery date. Consequently, the checking operation leads to a database request summing up all available resources in the context of the specific product. Apparently, materialized aggregates could be seen as one solution to tackle the expensive operation of on-the-fly aggregation. However, they fail in processing real-time order rescheduling due to incoming high priority orders leading to a reallocation of all products. Considering this operation as essential part of the present ATP application encompasses characteristics of analytical workloads with regards to low selectivity and low projectivity as well as aggregation functionality is used and read-only queries are executed. Along the afore mentioned check operation the write operations to declare products as promised to customers work on fine-granular transactional level. While looking at the characteristics of these write operations it is obvious that they belong to the OLTP category. In contrast to the analytic-style operations these write-queries are inherent of a high selectivity and high projectivity. This simplified example of a complex enterprise application shows workload characteristics, which match with those associated with OLTP and OLAP. As a consequence, nowadays database management systems cannot fulfill the requirements of specific enterprise applications since they are optimized for one or the other category leading to a mismatch of enterprise applications regarding the underlying data management layer. To meet this issue, enterprise applications have become increasingly complicated to make up for shortcomings in the data management infrastructure.

Besides, enterprise applications have become more sophisticated, data set sizes have increased, requirements on the freshness of input data have increased, and the time allotted for completing business processes has been reduced. At the same time hardware has evolved; for example the size of main memory has been significantly increased and multi-core CPU's allow calculation on-the-fly. Column databases are a data management technology in which data is organized along columns, in contrast to conventional RDBMSs where it is organized along rows. Column databases are particularly attractive for analytical queries as described in [19] [3], which often require subsets of relations and aggregations across columns. This is due to the fact that analytical applications are largely attribute-focused rather than entity-focused [2], in the that sense only a small number of columns in a table might be of interest for a particular query. This allows for a model where only the required columns have to be read while the rest of the table can be ignored. This is in contrast to the row-oriented model, where all columns of a table - even those that are not necessary for the result - must be accessed due to their tuple-at-a-time processing paradigm. Reading only the necessary columns exhibits a more cache-friendly I/O pattern, for both on-disk and in-memory column store databases. In the case of an on-disk column store, few disk seeks are needed to locate the page of a block that has the first field of a particular column. From there, data can be read in large blocks. In the case of an in-memory column store, sequentially laid out columns typically ensure a good second-level cache hit ratio, since modern main memory controllers use pre-fetching mechanisms which exploit spatial locality.

2 The Reason for Write-Optimized Storages

The most common light-weight compression technique for column stores is domain coding using a dictionary (see [3,19,12]) Using dictionary compression, frequently appearing patterns and variable length symbols are replaced by smaller fixed length symbols.

Depending on the data distribution this allows reduction of memory consumption and on the other hand performance improvement for query execution. This performance improvement comes with specific query plan operators that work directly on compressed data and thus increase the available bandwidth regarding to uncompressed data [21].

The consequence is that the performance of read operations is increased by the cost of rebuilding the compression scheme as soon as a new value might change the sort order of the used dictionary or breaks the currently used bit encoding scheme.

Following this assumption, inserts and updates invoked by OLTP style applications cannot be executed in a timely manner since the re-compression might delay each query more until a stable version of the data is reached, only valid until the next bulk of modification operations arrives.

For transactional workloads it becomes unavoidable to find a solution to this problem. A typical approach to this problem is to use a dedicated delta buffer for write operations and later on move the data from the delta buffer to the main storage [4,19]. While MonetDB X100 uses a chunk based approach for their delta columns equal to the PAX approach, C-Store uses a LSM tree optimized for disk access our approach proposes a memory only delta buffer with a disk based delta log for transaction safety. Besides the delta buffer the point in time and the way how the delta values are merged becomes important. Both of the before mentioned approaches do not discuss this. Referring to the original use cases of e.g. C-Store loading and updating delta values is typically done off-line.

To allow transactional workloads as described before and in the same system allow analytic style queries loading, merging and querying must be performed online.

3 SAP BWA as Read Optimized Column Store

SAPs Netweaver Business Warehouse Accelerator (BWA) is a main memory column-store database system. All relational tables are completely loaded into memory and stored column-wise. In order to save RAM and to improve access rates, the in-memory structure are highly compressed. This is achieved by different variants of Light Weight Compression (LWC) techniques like run-length encoding or multiple versions of fixed-length encoding.

3.1 Data Representation

In SAP Netweaver BWA the default compression mechanism is dictionary compression with bitwise encoding of columns. Here, all distinct values of a column are extracted, sorted and inserted into a dictionary. The column itself keeps only references to the dictionary, where each reference value is stored using as many bits as needed for fixed length encoding. Each value inside the encoded column is associated with an implicit row number — the document ID. By default all columns are sorted based on the sort order of the key column.

In addition to the default mode, SAP BWA offers a sparse mode for each table that allows to choose the best sort column based on statistical data and sort all other depending on the first one.

Internally each column is represented using two vectors. The first vector — the dictionary — is a compressed list of all distinct values stored in the columns. The second vector represents the row value inside the column. As values bit encoded keys from the value vector are used. This configuration shows that by default two lightweight compression methods are applied - domain coding and bit encoding.

When it comes to modification operations this compressed data representation is not suitable for modifications as shown in the following cases:

1. *Update without new value* - the changed value is already in the dictionary, replace the old value with the new value from the dictionary
2. *Update with new value* - the changed value is not in the dictionary, new value may change the sort order of the dictionary and force more values to be changed than only the selected one. In case the cardinality of the distinct values reaches 2^{n+1} with n is the old cardinality of distinct values the complete document vector must be rewritten.
3. *Append without new value* - append a new entry to the document using an existing value id from the dictionary
4. *Append with new value* - append new value, with same complexity as for *Update with new value*
5. *Delete without value change* - Since the offset of the document vector is the implicit row number, a delete may force a complete lock of the table until all readers finished their queries. In case of long running OLAP queries this can decrease the performance of a write operation dramatically.
6. *Delete with value change* - Combined complexity of an *Update with new value* and a *Delete without value change*.

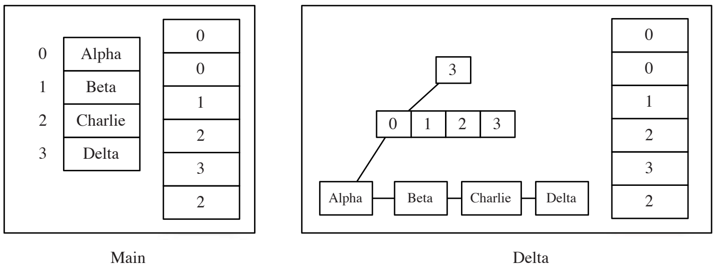


Fig. 1. Structure of Main and Delta Storage

Each of the operations has a different impact on the overall performance for modification operations. To increase the speed for those operations SAP BWAoffers a delta buffer for modifications. Instead of using the same representation as for the compressed main store, the delta buffer does not use a compressed sorted list but a CBS+ Tree[16] to store the values. The CBS+ Tree allows high modification rates on the one hand and is optimized towards main memory systems. All values stored in the value tree are not compressed but the document vector still applies bit encoding to save memory. Figure 1 shows the structure of a sample attribute with a main storage and a delta buffer.

To conclude a relation stored in SAP BWAis a set of columns. Each column is associated with a main storage column and if modifications exist a delta buffer column. Furthermore it is required that all columns have the same length and all inserts go directly to the delta buffer while updates are handled as a delete operation with a proceeding insert operation on the same key. To make all operations immediately visible at runtime all operations have to be performed once on the main store and once on the delta buffer of the requested attribute.

Since deletions cannot be performed immediately a valid row bit vector is used to identify those rows that are no longer valid and must not be returned during searches. Furthermore this bit vector is later used to identify those rows that are not merged and possibly refer to values that are no longer needed.

4 The Merge Process

Merging delta and main index at some point in time is required to maintain the performance of column-store in write intensive scenarios. The reasons for merging are two-fold. On the one hand merging the delta store into the main relation decreases the memory consumption since better compression techniques can be applied. On the other hand merging the delta allows better search performance due to the ordered value dictionary of the main store.

The main requirement of the merge process is that it runs in asynchronously, has as less impact as possible on all other operations and does not affect the transaction behavior. To achieve this goal the merge creates copies of the data and only seldom acquires locks. During the merge, the process consumes additional resources (CPU and main memory).

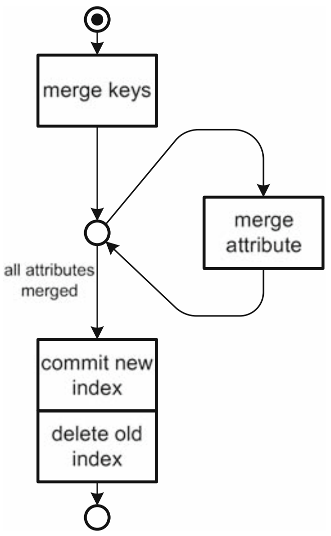


Fig. 2. Merge Process as Petri Net

4.1 Detailed Description of the Merge Algorithm

The merge phase can be separated into three phases - *prepare merge*, *attribute merge* and *commit merge*. During the prepare merge phase the following actions are executed. At first the table is locked for all queries and a new empty delta buffer structure is created called delta 2. All updates and inserts will be sent to this new delta structure since the original delta and the main structure are merged. In addition it creates a copy of the current valid document ids to identify the snapshot of valid documents at merge start time. As soon as those actions are finished, the table lock is released.

Now for every attribute (and key attributes before all other attributes) the following process is executed. Since each of the attributed is compressed using a dictionary (see 3) at first the dictionary of the main and the delta store must be merged to create a consistent view of all available values. Due to possible changes in value ids a mapping table is created to map the value ids from the old main store and the delta store to the new value ids from the merged dictionary. Applying the valid document list, the mapping table and the value ids from the main store the value ids from the original main vector are copied and secondly the value ids from the delta vector. The valid document list is hereby used to identify the rows that where removed from the table. Since either updates of rows or deletion can lead to a pint where a value is no longer referenced the new document vector is searched for unreferenced values which are than removed from the mapping list and left out during creation of the new dictionary.

As a last step in the attribute merge phase the new attribute is written to a secondary storage for durability reasons in case of failures. Optimizations when writing this file are possible but left out of the discussion for future research. When the attribute merge is finished for this attribute the next attribute merge starts or phase three starts.

In phase three the relation is committed. The commit phase starts with acquiring an exclusive table lock. When the lock is acquired the original valid document id list fetched at the beginning of the merge process is compared to the current valid document id list to identify additional deletions during merge. Rows that were deleted during the merge will be marked invalid using the valid-bit-vector and will remain invisible until they are removed in the next merge.

As a next step the compressed binary representation of all attributes are replaced by the new ones, the old table and delta 1 are unloaded from memory, and delta 2 is renamed to delta 1. When the unload step is finished the commit phase finishes and the merge process is finished making the new compressed version of the table available.

Queries and Lock Granularity. During merge runtime all queries are answered from the storages of the main, delta 1, and delta 2. The complete merge process only works with copies of the data and thus is totally independent of the original data. Since all new modifications are reflected in delta 2, main and delta 1 can be safely merged. During merge runtime only at two points in time exclusive locking is required: first during the prepare phase when delta 2 is created and the valid document id list is copied and second during the commit phase.

Failure Case and Recovery. Each phase during the merge can fail independently. During the prepare phase critical situation is the creation of delta 2 and the locking of delta 1. The entry of the prepare phase and the end are logged in the delta log. Recovery is performed with erasing the eventual existing delta 2 and freeing the lock. Afterwards the delta merge process can be re-started.

If an failure occurs during the attribute merge phase (e.g. power failure) recovery can be performed by reloading the table plus delta 1 and delta 2. All files stored on disk are erased and the merge process starts from the beginning.

In the commit phase the recovery is handled as described before for the other phases. Since a table lock is acquired during the commit phase the merge process can easily erase and unload all partial merge data and restart the process.

In addition restarting capabilities of main memory systems become very important. Figure 3 shows a comparison. In this experiment we compare the reload time of the fully compressed main table and the tree structured uncompressed delta representation. From the graphs we can derive that it is always beneficial to perform the merge instead of delaying it.

4.2 Complexity Analysis of the Merge Algorithm

This section aims to identify the complexity of the different merge steps and to identify all parameters that possibly influence the merge. The expected influence will be validated in the next section. The goal of this analysis is to answer the question when is the best possible point in time to merge the delta values into the main document vector. As described earlier the merge of an attribute consists out of the following stages:

1. **Create a value id mapping for all main and delta values**
   Simultaneously iterate once over both value vectors and create a global mapping of both value vectors.

   $$O(|Dict_M| + |Dict_D|)$$

   The linear dependency for the delta values is achieved due to the linear iteration costs for the CBS+ Tree.
2. **Create the new document vector**
   Iterate over delta document vector to retrieve the highest document id; iterate over the main document vector and copy the valid value ids to the new main vector, iterate over the delta document vector and copy the values to the new main vector.

   $$O(2 \cdot |Doc_D| + |Doc_M| + |BV_{valid}|)$$

3. **Remove unreferenced values**
   Iterate over all documents in the new main vector to extract all referenced values, then iterate over all referenced values and compare to the global mapping of the original delta and main value vectors.

   $$O(|Doc_{new\_M}| + |Dict_{new\_M}|)$$

4. **Create the new dictionary**
   Creating the new dictionary is done in a similar way like merging the new dictionary. It reuses the sorted list created during the mapping of the main and delta values. Since this list is already sorted no additional for sorting occur.

   $$O(|Dict_M| + |Dict_D|)$$

5. **Adjust value ids**
   It is possible that due to changes of value ids from the remove unreferenced values step, no longer used value ids are used. These must be adjusted and set to the new values from the dictionary. In cooperation this step uses the mapping from the first step and the new dictionary created one step before.

   $$O(|M_{vid}| + |Doc_{new\_M}|)$$

Observation. The implementation of our merge process shows a very important characteristic: All operations linearly depend on the size of the delta and main storage. Even though the implementation of the value storage in for a delta attribute is based on a tree representation the merge process benefits from the linear iteration on all value items.

4.3 Evaluation

To verify the complexity of the given access patterns a test case has been devised which varies the main arguments separately to show their contribution to the

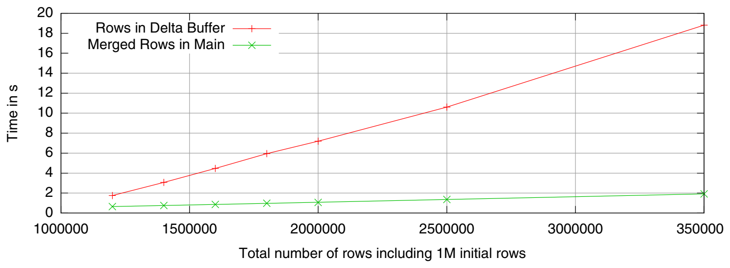


Fig. 3. Time needed to fully load the table into memory compared between delta and main

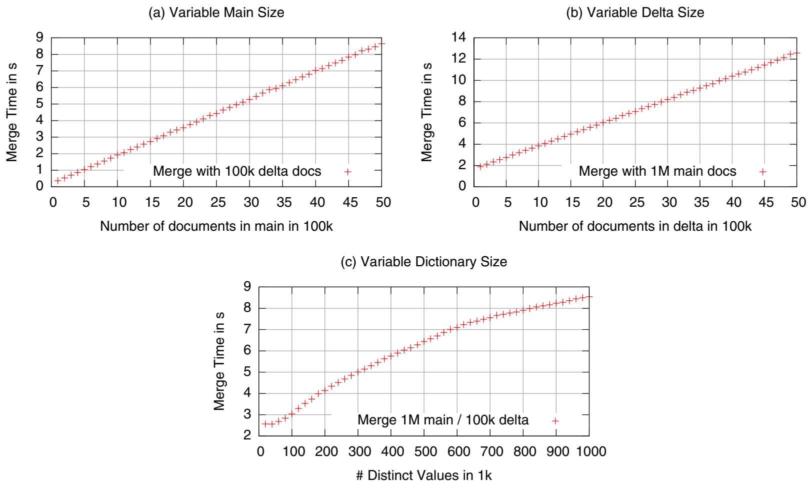


Fig. 4. Evaluation Results

merge duration. The setup for the test case is as following: a single integer column is created in SAP BWAwith 1M rows in the main document vector and 100k rows in the delta document vector with one value in both the main and the delta dictionary - thus all rows having the same identical value. In the following test each of the parameters is varied over a specific range while the others remain at their base value.

Document Vector Size. Varying main or delta document vector sizes results in a linear increase of the merge duration - see figure 4. The observation proves that main and delta document size have a linear contribution to the complexity of the merge process.

Dictionary Size. Dictionary sizes have been varied from their base value of one up to the maximum number of entries in the main or delta vector, so that the value in the test column are 100% distinct entries. The results in Figure 4 show an increase towards the fully distinct data set which validates the assumption of a linear contribution to the merge time.

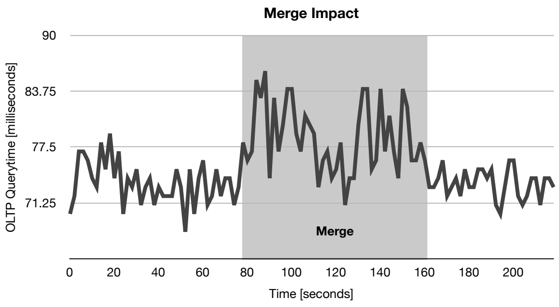


Fig. 5. Impact on OLTP Queries on Low System Load

Merge Impact to Running Queries. Due to the implementation of the merge, the impact on the rest of the system is determined by resource competition and lock granularity. As described earlier locking is performed in the beginning and in the end using an exclusive table lock.

To measure the impact regarding resource competition we created a workload that contains queries from two different kinds — OLTP and OLAP. The workload is based on real customer data taken from a SAP financials and accounting system and issues the following queries:

1. Primary key select based on accounting year, customer id, accounting area, accounting entry id and accounting line
2. Primary key select on configuration tables
3. Select the sum of all open item grouped by customer

From the area of reporting we extracted a query from the SAP Business Warehouse. This query calculates all open items of one customer based on business area, postal code and accounting area and groups by a due date in a grid of 0-29, 30-59, 60-90 and greater than 90 days.

The main table containing the accounting entry line items consists out of 260 million tuples with 300 columns with a 250GB disk size and a compressed size of 20GB in memory. The system used for execution is a 8 core (3.0Ghz) Intel Xeon 5450 blade with 32GB RAM and SuSe Linux Enterprise Edition.

Figure 5 shows the query execution time over a given time span basically executed against a single table with a low system utilization. During the time of merge the overall response time is slowed down by ≈ 6%. When running the system in a high load scenario (see Figure 6) this degradation becomes almost invisible.

We could show the same behavior for OLAP queries as well — see Figures 7 and 8. Our observation leads to the conclusion that as long as enough resources are available the query execution is not affected by the merge process.

As a result we can construct two major cases where executing the merge process may result in system failure. In our observation we used system loads up to 600% by starting up to 6 server processes and querying them simultaneously. Figure 9 shows the distribution of the load during our tests. Our assumption is

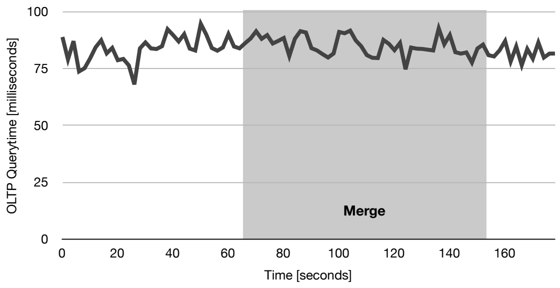


Fig. 6. Impact on OLTP Queries on High System Load

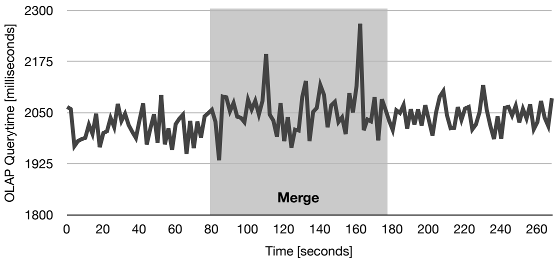


Fig. 7. Impact on OLAP Queries on Low System Load

that as long as one CPU has enough free resources to perform the merge this process will create big negative impact on the other running queries.

The other major obstacle is the memory consumption during the merge process. Currently the complete new main storage is kept inside memory until the merge is finished. In the last step when the merge process acquires an exclusive table lock the old data is unloaded and freed. Until this point double the amount of the main plus delta storage is required.

5 Optimizations and Future Work

Based on our implementation and evaluation the current merge process has a great advantage: Due to its simplistic nature it is easy to calculate when it is best to execute the merge and how long it will take. On the other hand this does not take into account how applications are build and does not leverage current hardware trends towards multi-core systems with more than 32 cores. To address this issue we started to analyze customer systems to identify how much data characteristics could help to improve the merge process.

Exemplarily we show two results from a customer system of the consumer packaged goods industry and its main tables from the financials and accounting system. Figure 10 shows the result from the distinct value analysis for one calendar year of the accounting document entries table **BKPF**. The picture shows

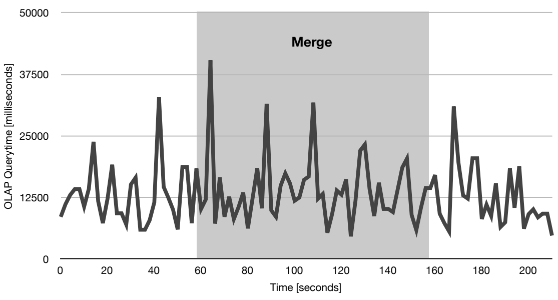


Fig. 8. Impact on OLAP Queries on High System Load

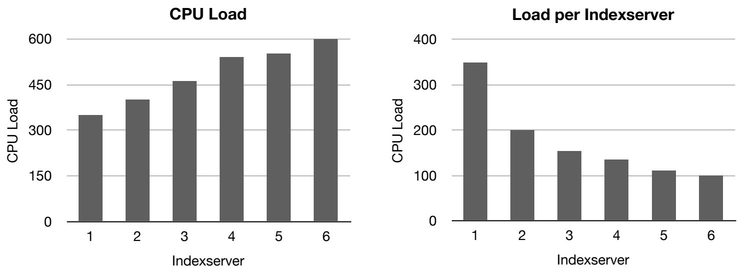


Fig. 9. System Load Distribution

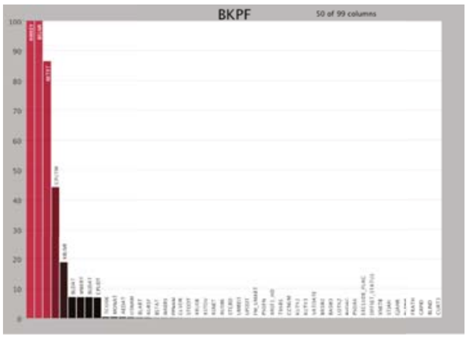


Fig. 10. Distinct Value Analysis for Accounting Document Entries

only the distinct values from the first 50 of 99 and only the first 10 attributes have lots of distinct values. Based on source code analysis and interviews we saw that the rest of the values typically is either a default value or a null value. In addition most of the expected values are known in advance [14]. Here, the merge process should leverage this knowledge.

The most common compression scheme used is domain coding, but typically the database has no common interpretation of what the domain acutally is, besides the datatype used. This becomes even worse since applications tend to

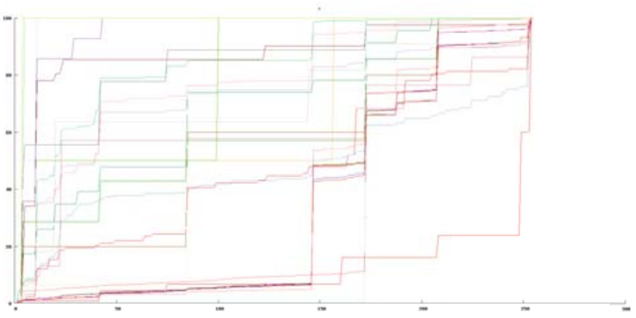


Fig. 11. Dictionary Fill Level per Attribute over Time

use very generic datatypes to be as flexible or compatible as possible. To prove our assumption we analyzed the total distinct values used by the application and processed all change documents created by the system to arrange the dictionary status in a timely manner — change documents are used by SAP applications to record changes for legal reasons. Figure 11 shows the result of this analysis. The result is that for many different attributes already after a few month it is possible to define the complete dictionary without the need to recreate this dictionary again during the merge.

6 Related Work

Pure vertical partitioning into a "column-store" has been a recent topic of interest in the literature. Copeland and Khoshafian [7] introduced the concept of a Decomposition Storage Model (DSM) as a complete vertical, attribute-wise partitioned schema. This work has been the foundation for multiple commercial and non-commercial column store implementations. Popular examples are MonetDB/X100 [4], C-Store [19] or Sybase IQ [8]. Recent research has shown their ability to outperform conventional databases in read-only OLAP scenarios with low selectivity. SybaseIQ and C-store are pure disk based approaches to a column-store implementation. Since updates must be propagated to multiple files on disk, they are inappropriate for OLTP workloads, because updates and inserts are spread across different disk locations. Data compression also limits their applicability to OLTP scenarios with frequent updates. As described in [4], MonetDB/X100 implements dedicated delta structures to improve the performance of updates. The MonetDB/X100 storage manager treats vertical fragments as immutable objects, using a separate list for deleted tuples and uncompressed delta columns for appended data. A combination of both is used for updates. In contrast to our research, the merge process it self is not considered any further.

Another existing part of research focuses on mixed workloads based on conventional RDBMSs. [5] [13] assume that certain queries will be longer running than others and allocate resources so that class-based SLAs can be met. Our target applications require that all queries execute rapidly. [9] modifies a row store to better support analytics, which favors the transactional workload. Our approach modifies a column store to support transactions, which favors the analytical workload.

[15] handles a mixed workload by placing a row store and a column store side-by-side in the same system. One copy of the data is stored in row format while the other is stored in column format. The row and column representations are interleaved across the two database instances to better distribute the load as queries are routed regarding their access pattern towards the optimal physical data representation. Column compression and merging were not implemented, however they should be present in a production system. The merge optimizations we propose could be applied in that case to improve the performance of the column store.

The authors of [1,20,6] describe how effective database compression based on lightweight compression techniques by attribute-level can be and furthermore be leveraged for query precessing in read-optimized application scenarios. Due to recent improvements in CPU speed which have outpaced main memory and disk access rates by orders of magnitude, the use of data compression techniques are more and more attractive to improve the performance of database systems. Those techniques are used in our research and lead to our concept of dividing the storages since we focus on a mixed workload. Other work, such as [10] and [11] study read optimized databases to improve the performance of database systems in read-intensive environments.

Furthermore, there has been substantial research on delta-indexing [17,18]. The concept of maintaining differential files was particularly considered for very large databases where both the delta and the main index where queried in order to provide latest results. As in our scenario, in-place updates are too expensive and thus collected and scheduled as a batch job.

7 Conclusion

In this paper we showed an implementation strategy for merging values from a delta buffer that is optimized for modification operations into a compressed read-optimized version. Furthermore we showed for each of the steps that the participating data stores — main, delta and dictionary — have a linear contribution to the overall merge costs.

We validated our concept with an implementation used in the read-optimized database SAP BWA. In the end we showed potential improvements for the merge process and how important it is to observe real world customer data to improve this process.

References

1. Abadi, D., Madden, S., Ferreira, M.: Integrating compression and execution in column-oriented database systems. In: SIGMOD 2006, pp. 671–682. ACM, New York (2006)

2. Abadi, D.J.: Query Execution in Column-Oriented Database Systems. PhD thesis
3. Boncz, P.A., Manegold, S., Kersten, M.L.: Database architecture optimized for the new bottleneck: Memory access. In: VLDB, pp. 54–65 (1999)
4. Boncz, P.A., Zukowski, M., Nes, N.: Monetdb/x100: Hyper-pipelining query execution. In: CIDR, pp. 225–237 (2005)
5. Brown, K.P., Mehta, M., Carey, M.J., Livny, M.: Towards automated performance tuning for complex workloads. In: VLDB, pp. 72–84 (1994)
6. Chen, Z., Gehrke, J., Korn, F.: Query optimization in compressed database systems. In: SIGMOD 2001, pp. 271–282. ACM, New York (2001)
7. Copeland, G.P., Khoshafian, S.: A decomposition storage model. In: SIGMOD Conference, pp. 268–279 (1985)
8. French, C.D.: "one size fits all" database architectures do not work for DDS. In: SIGMOD Conference, pp. 449–450 (1995)
9. French, C.D.: Teaching an OLTP database kernel advanced data warehousing techniques. In: ICDE, pp. 194–198 (1997)
10. Harizopoulos, S., Liang, V., Abadi, D.J., Madden, S.: Performance tradeoffs in read-optimized databases. In: VLDB, pp. 487–498 (2006)
11. Holloway, A.L., DeWitt, D.J.: Read-optimized databases, in depth. PVLDB 1(1), 502–513 (2008)
12. Legler, T., Lehner, W., Ross, A.: Data mining with the SAP NetWeaver BI accelerator. In: VLDB 2006, pp. 1059–1068. VLDB Endowment (2006)
13. Pang, H., Carey, M.J., Livny, M.: Multiclass query scheduling in real-time database systems. IEEE Trans. Knowl. Data Eng. 7(4), 533–551 (1995)
14. Plattner, H.: A common database approach for OLTP and OLAP using an in-memory column database. In: SIGMOD Conference, pp. 1–2 (2009)
15. Ramamurthy, R., DeWitt, D.J., Su, Q.: A case for fractured mirrors. In: VLDB, pp. 430–441 (2002)
16. Rao, J., Ross, K.A.: Making b^{+}-trees cache conscious in main memory. In: SIGMOD Conference, pp. 475–486 (2000)
17. Rappaport, R.L.: File structure design to facilitate on-line instantaneous updating. In: SIGMOD 1975, pp. 1–14. ACM, New York (1975)
18. Severance, D.G., Lohman, G.M.: Differential files: their application to the maintenance of large databases. ACM Trans. Database Syst. 1(3), 256–267 (1976)
19. Stonebraker, M., Abadi, D.J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E.J., O'Neil, P.E., Rasin, A., Tran, N., Zdonik, S.B.: C-store: A column-oriented dbms. In: VLDB, pp. 553–564 (2005)
20. Westmann, T., Kossmann, D., Helmer, S., Moerkotte, G.: The implementation and performance of compressed databases. SIGMOD Rec. 29(3), 55–67 (2000)
21. Willhalm, T., Popovici, N., Boshmaf, Y., Plattner, H., Zeier, A., Schaffner, J.: Simd-scan: Ultra fast in-memory table scan using on-chip vector processing units. PVLDB 2(1), 385–394 (2009)

Towards an Algebraic Framework for Querying Inductive Databases

Hong-Cheu Liu[1], Aditya Ghose[2], and John Zeleznikow[3]

[1] Department of Computer Science and Mutimedia Design,
Diwan University,
Madou, Tainan County, 72153 Taiwan
hongcheu.liu@gmail.com

[2] School of Computer Science and Software Engineering,
University of Wollongong,
Wollongong, NSW 2522 Australia
aditya_ghose@uow.edu.au

[3] School of Information Systems,
Victoria University,
Melbourne, Vic. Australia
John.Zeleznikow@vu.edu.au

Abstract. In this paper, we present a theoretical foundation for querying inductive databases, which can accommodate disparate mining tasks. We present a data mining algebra including some essential operations for manipulating data and patterns and illustrate the use of a fix-point operator in a logic-based mining language. We show that the mining algebra has equivalent expressive power as the logic-based paradigm with a fix-point operator.

1 Introduction

While knowledge discovery from large databases has gained great success and popularity, there is conspicuously lack of a unifying theory and a generally accepted framework for data mining. The integration of data mining with the underlying database systems has been formalised in the concept of inductive databases which is a very active area in the past decade. The key ideas of inductive database systems are that data and patterns (or models) are first class citizen.

The one of crucial criteria for the promising success of inductive databases is reasoning about query evaluation and optimisation. Currently, research on inductive query languages has not led to significant commercial deployments due to performance concern and practical limitations. By contrast, relational database technology was based on algebraic and logical frameworks and then followed on both theoretical and system fronts, leading to an extremely successful technology and science. Therefore a theoretical framework that unifies different data mining tasks as well as different data mining approaches is necessary and would help the field and provide a basis for future research [1,2].

In this article, we present an algebraic framework based on a complex value data model. Compared to the model presented in [3,4], we believe that it is more appropriate to adopt a complex value data model as data and mining results are normally represented as complex values. Some natural and dominant data mining computations can be expressed as compositions and/or combinations of known mining tasks. For example, an analyst might find a collection of frequent item-sets bought. He or she may further analyses these sets using a decision tree to predict under what conditions customers are classified for credit rating and such frequent co-purchases are made by this category of customers. This kind of problems can be easily expressed as an expression by using the proposed data mining algebra operators. The algebraic framework also provides a promising approach for query optimisation.

Constraints play a central role in data mining and constraint-based data mining is now a recognised research topic. The area of inductive databases, inductive queries and constraint-based data mining has become a unifying theory. Declarative query language acts an important role in the next generation of Web database systems with data mining functionality.

One of important properties of declarative query languages is *closure*. The result of a query is always a relation which can be available for further querying. This closure property is also an essential feature of inductive query languages.

The inductive query languages proposed in the literature require users to only provide high-level declarative queries specifying their mining goals. The underlying inductive database systems need sophisticated optimiser with aim to selecting suitable algorithms and query execution strategies in order to perform mining tasks. Another tight-coupling approach using SQL implementation gives unrealistic heavy-burden on the users to write complicated SQL queries. So it is reasonable to explore alternative methods that make inductive databases realisable with current technology. In this paper, we also consider a logical framework for querying inductive databases.

2 An Algebraic Framework for Data Mining

In this section, we present an algebraic framework for data mining. The framework is based on a complex value data model.

2.1 A Data Mining Algebra

Let $\Omega = \{R_1, ..., R_n\}$ be a signature, where R_i, $1 \leq i \leq n$, are database relations. The data mining algebra over Ω is denoted as $\mathcal{DMA}(\Omega)$. A family of core operators of the algebra is presented as follows. **Set operations:** Union ($\cup$), Cartesian product ($\times$), and difference (-) are binary set operations. **Tuple operations:** Selection (σ) and projection (π) are defined in the natural manner. **Powerset:** $powerset(r)$ is a relation of sort $\{\tau\}$ where $powerset(r) = \{\nu \mid \nu \subseteq r\}$. **Tuple Creation:** If $A_1, ..., A_n$ are distinct attributes, $tup\_create_{A_1,...,A_n}(r_1, ..., r_n)$ is of sort $< A_1 : \tau_1, ..., A_n : \tau_n >$, and $tup\_create_{A_1,...,A_n}(r_1, ..., r_n) = \{< A_1 : \nu_1, ..., A_n : \nu_n >\mid \forall i(\nu_i \in r_i)\}$. **Set Creation:** $set\_create(r)$ is of sort $\{\tau\}$, and $set\_create(r) = \{r\}$. **Tuple Destroy:** If r is of sort $< A : \tau^{'} >$, $tup\_destroy(r)$ is a relation of sort $\tau^{'}$ and $tup\_destroy(r) = \{\nu \mid < A : \nu > \in r\}$. **Set Destroy:**If $\tau = \{\tau^{'}\}$, then $set\_destroy(r)$ is a relation of sort $\tau^{'}$ and $set\_destroy(r) = \cup r = \{w \mid \exists \nu \in r, w \in \nu\}$. **Aggregation:** The standard set of aggregate functions SUM, COUNT, AVG, MIN, MAX are defined in the usual manner. For example, if r is of sort $< A : \tau_1, B : \tau_2 >$, $\mathcal{G}_{<A>}^{function<B>}(r)$ is the relation over $< A, S >$. $\mathcal{G}_{<A>}^{function<B>}(r) = \{< a, s > \mid \exists < a, v > \in r \wedge s = \Sigma\{t < B > \mid t \in r, t < A, B > = < a, b >\}$, where Σ is one of the allowed aggregate operator.

Definition 1. *If r is a relation of l-tuples, then the append operator, $\Delta_{\delta(i_1,...,i_k)}(r)$ is a set of $l+1$ tuples, $k \leq l$, where δ is an arithmetic operator on the components $i_1, ..., i_k$. The last component of each tuple is the value of $\delta(i_1, ..., i_k)$.*

Example 1. Let the sort of a relation r be $R : \{< X : \mathbf{dom}, Y : \mathbf{dom} >\}$. A value of this sort is $\{< X : 2, Y : 3 >, < X : 5, Y : 4 >\}$. Then $\Delta_{X+Y=Z}(r) = \{< X : 2, Y : 3, Z : 5 >, < X : 5, Y : 4, Z : 9 >\}$.

Definition 2. *[5] Let us consider two relations with the same sort $\{Item, Count\}$. $r \bowtie^{sub,k} s = \{t \mid \exists u \in r, v \in s$ such that $u[Item] \subseteq v[Item] \wedge \exists t^{'}$ such that $(u[Item] \subseteq t^{'} \subseteq v[Item] \wedge |t^{'}| = k)$, $t = < t^{'}, v[Count] >\}$*

Here, we treat the result of $r \bowtie^{sub,k} s$ as multi-set meaning.

Example 2. Consider the sort $\{< Item : \{\mathbf{dom}\}, Count : \mathbf{dom} >\}$ and two relations $r = \{< \{a\}, 0 >, < \{b, f\}, 0 >\}$ and $s = \{< \{a, b, c\}, 3 >, < \{b, c, f\}, 4 >\}$ of that sort. The result of $r \bowtie^{sub,2} s$ is $\{< \{a, b\}, 3 >, < \{a, c\}, 3 >, < \{b, f\}, 4 >\}$.

Definition 3. *Let X be a set. The map(f) operator applies a function f to every member of the set X, i.e., $map(f) : X \rightarrow \{f(x) \mid x \in X\}$.*

2.2 Frequent Item-Set Computation

Given a database $D = (Item, Support)$ and support threshold δ, the following fix-point algorithm computes frequent item-set of D.

Let f^k be a function which applies to a set T and results in the set of degree-k subset of T. For any two sets S and s, s is said to be a degree-k subset of S if $s \subset S$ and $|s| = k$.

Algorithm *Fix-point*
Input: An object-relational database D and support threshold δ.
Output: L, the frequent item-sets in D.
Method:

```
begin
    $L_1 := \sigma_{Support \geq \delta}(\ _{Item}\mathcal{G}_{sum(Support)} map(f^1)(D)))$
    for $(k = 2; T \neq \emptyset; k++)$ {
    $S := sub\_join(L_{k-1}, D)$
    $T := \sigma_{Support \geq \delta}( {}_{Item}\mathcal{G}_{sum(Support)}(S)$
    $L_k := L_{k-1} \cup T$ }
    return $L_k$;
end
```

```
procedure $sub\_join$
(T: frequent k-itemset; D: database)
for each itemset $l_1 \in T$,
for each itemset $l_2 \in D$,
$c = l_1 \bowtie^{sub,k} l_2$
if $has\_infrequent\_subset$ (c, T)
then delete $c$ else add $c$ to $S$;
return S;
```

```
procedure $has\_infrequent\_subset$
(c: $k$-itemset, T: frequent (k-1)-itemsets);
for each (k-1)-subset $s$ of c
if $s$ not $\in T$ then return TRUE;
return FALSE;
```

2.3 Decision Tree Induction

In this subsection, we give an algebraic expression for constructing a decision tree in $\mathcal{DMA}$. The process is based on the well-known C4.5 algorithm [6] for tree induction.

We assume that the algorithm is called with three parameters: D, *attribute_list*, and *Attribute_selection_method*. We refer to D as a data partition. Initially, it is the complete set of training tuples and their associated class labels. The parameter *attribute_list* is a list of attributes describing the tuples, i.e., $D = (A_1, A_2, ..., A_n, Class)$. *Attribute_selection_method* specifies a heuristic procedure for selecting the attribute that best discriminates the given tuples according to class. The procedure employs the attribute selection measure - information gain. The complete algebraic expression is formulated by Algorithm *Generate_decision_tree* as shown below.

The output of the algorithm will be a complex value relation $\mathcal{T}$ which holds the set of inequalities on the edges from the root to one leaf, together with their labels.

Algorithm: *Generate_decision_tree*
input

– Data partition, D, which is a set of training tuples and their associated class labels;
– *attribute_list*, the set of candidate attributes;
– *Attribute_selection_method*, a procedure to determine the splitting criterion.

Output: A complex value relation which holds the set of all inequalities on the edges from the root to leaves together with their labels.
Method

```
1. $\mathcal{T} := \{\}$;
2. $Split := \{\}$
3. if $Count(\pi_{Class}(D)) = 1$ then
4. $\mathcal{T} := \mathcal{T} \cup set\_create(tuple\_create(Split, \pi_{Class}(D)))$.
```

```
 5. **If** $attribute\_list = \{\}$ **then**
 6. $\mathcal{T} := \mathcal{T} \cup \pi_{Class}\sigma_{Count=Max}(\mathcal{G}_{Class}^{Count<R-Class>})$
 7. apply $Attribute\_selection\_method(D, attribute\_list)$ to find the best splitting_attribute
 8. $Split := Split \cup \{splitting\_attribute\}$
 9. **if** $splitting\_attribute$ is discrete-valued **and** multi-way splits allowed **then**
10. $attribute\_list \leftarrow attribute\_list - splitting\_attribute$;
11. **for each** outcome $j$ of $splitting\_criterion$
12. $D_j = \sigma_j(D)$
13. **if** $D_j = \emptyset$ **then**
14. $\mathcal{T} := \mathcal{T} \cup \pi_{Class}\sigma_{Count=Max}(\mathcal{G}_{Class}^{Count<R-Class>})$
15. **else** $Generate\_decision\_tree(D_j, attribute\_list)$;
16. **endfor**
17. **return** $\mathcal{T}$
```

The tree starts as an empty relation. If the tuples in D are all of the same class, the resulting output relation contains only one class value. Note that steps 5 and 6 are terminating conditions. Otherwise, the algorithm calls *Attribute_selection_method* to determine the splitting criterion. A branch is grown from the current tuple for D_j for each of the outcomes of the splitting criterion.

Similarly, the procedure $Attribute\_selection\_method(D, attribute\_list)$ can also be specified as an algebraic expression in $\mathcal{DMA}$.

Example 3. As described in Introduction, an analyst might find a collection of frequent item-sets bought. He or she may further analyses these sets using a decision tree to determine the circumstances (e.g., class for credit rating) under which such frequent co-purchases are made by this category of customers.

This query is easily to be expressed in $\mathcal{DMA}$. It is formulated as $Generate\_decision\_tree(Fix\text{-}point(D), (age, ...), Attribute\_selection\_method)$.

3 A Logical Framework for Data Mining

In this section, we give some basic data mining concepts based on logic. Inductive database queries can be formalised in a higher order logic satisfying some constraints.

Definition 4. *Given an inductive database $\mathcal{I}$ and a pattern class $\mathcal{P}$, a pattern discovery task can be specified as a query q such that $q(\mathcal{I}) = \{t \in \mathcal{P} \mid \mathcal{I} \models \varphi(t)\}$, where φ is a higher-order formula.*

Definition 5. *A constraint is a predicate on the powerset of the set of items $\boldsymbol{I}$, i.e., $C : 2^I \mapsto \{true, false\}$. An itemset X satisfies a constraint C if $C(X)$ is true.*

Definition 6. *Let $\mathcal{I}$ be an instance of an inductive databases with a pattern class $\mathcal{P}$ and a complex value relational schema $R = (A_1, ..., A_n)$. An association rule* *is an element of the set* $\mathcal{L} = \{A \Longrightarrow B \mid A, B \in \{A_1, ..., A_n\}$ *such that* $< A \Longrightarrow B > \in q(\mathcal{I})$ *if and only if* $freq(A \cup B, r) \geq s$ *and* $freq(A \cup B, r)/freq(A, r) \geq c$. *Where* $freq(X, r)$ *is the frequency of* X *in the set of* r, s *is the support threshold and* c *is the confidence threshold.*

Definition 7. *Given an inductive database* $\mathcal{I}$, *an inductive clause is an expression of the form* $P(u) \leftarrow R(u_1), ..., R_(u_n)$, *where* $n \geq 1$, R_i *are relation names and* u, u_i *are free tuple of appropriate arity.*

Example 4. Let a transaction relation be $T = (ID, Items)$ and each item in the transaction database has an attribute value (such as profit). The constraint $C_{avg} \equiv avg(S) \geq 25$ requires that for each item-set S, the average of the profits of the items in S must be equal or greater than 25. The frequent pattern mining task is to find all frequent item-sets such that the above constraint holds. We express it as inductive clauses as follows.

$$\begin{array}{ll}
freq\_pattern(support, < Items >) & \leftarrow T(ID, Items), \\
& support = freq < Items > /Count(T), \\
& support \geq s \\
F\_pattern(Items, AVG) & \leftarrow freq\_pattern(support, Items), \\
& Value(Item, value),\ item \in Items, \\
& AVG = SUM(value)/SUM(Items) \\
Ans(Items) & \leftarrow F\_pattern(Items, AVG), AVG \geq 25
\end{array}$$

It is simple to specify Naive Bayesian classification by means of a deductive database program. The detailed program can be found in [5].

We present a deductive program performing the partitioning-based clustering task, as follows. $P(Y, C_i) \leftarrow r(X)$, $1 \leq i \leq k$, $Y_i = X$; $Cluster(Y, C_i, m_i) \leftarrow P(Y, C_i)$, $m_i = mean\{Y\}$ Where *mean* is a function used to calculate the cluster mean value; *distance* is a similarity function. The following two rules show the clustering process. An operational semantics for the following datalog program is fix-point semantics.

Example 5. The clustering process can be expressed as follows.

$$\begin{array}{ll}
new\_cluster(X, C) \leftarrow & r(X),\ Cluster(Y, C, m),\ Cluster(Y, C', m'), \\
& c \neq c',\ distance(X, m) < distance(X, m'), \\
Cluster(X, C, m) \leftarrow & new\_cluster(X, C),\ m = mean\{new\_cluster.X\}
\end{array}$$

Theorem 1. *Any data mining queries expressible in* $\mathcal{DMA}$ *with while loop can be specified as inductive clauses in* $Datalog^{cv,\neg}$.

Proof Sketch. $\mathcal{DMA}$ is equivalent to $CALC^{cv}$. A query is expressible in Datalog$^{cv,\neg}$ with stratified negation if and only if it is expressible in complex value calculus $CALC^{cv}$. $CALC^{cv}$ is equivalent to $CALC^{cv}$ + fixpoint. So $\mathcal{DMA}$ + while loop is equivalent to $CALC^{cv}$ + fix-point. Any data mining queries expressible in $\mathcal{DMA}$ with while loop can be specified as inductive clauses in Datalog$^{cv,\neg}$.

4 Query Optimisation Issue

An important step towards efficient query evaluation for inductive databases is to identify a suitable algebra in which query plans can be formulated. The algebraic framework presented in Section 2 provides a promising foundation for query optimisation. However, there exist many challenges for optimisation issues. For example, it is difficult to establish a cost model for mining operations; to formally enumerate all the valid query plans for an inductive query and then choose the optimal one is not straightforward.

We argue that if SQL would allow expressing our sub-join, $\bowtie^{sub,k}$, in an intuitive manner and algorithms implementing this operator were available in a DBMS, this would greatly facilitate the processing of fix-point queries for frequent itemset discovery.

A Datalog expression mapping to our fix-point operator has more intuitive than SQL expressions. In our opinion, a fix-point operator is more appropriate exploited in the deductive paradigm which is a promising approach for inductive database systems.

We may improve performance by exploiting relational optimisation techniques, for example, optimizing subset queries, index support, algebraic equivalences for nested relational operators [7]. In the deductive paradigm, we may also apply pruning techniques by using the 'anti-monotonicity'.

5 Conclusion

We have presented an algebraic framework for querying inductive databases. The framework would be helpful for understanding querying aspect of inductive databases.

We have also presented a logic programming inductive query language. The results provide theoretical foundations for inductive database research and could be useful for query language design in inductive database systems.

References

1. Dzeroski, S.: Towards a general framework for data mining. In: Džeroski, S., Struyf, J. (eds.) KDID 2006. LNCS, vol. 4747, pp. 259–300. Springer, Heidelberg (2007)
2. Calders, T., Lakshmanan, L., Ng, R., Paredaens, J.: Expressive power of an algebra for data mining. ACM Transactions on Database Systems 31(4), 1169–1214 (2006)
3. Blockeel, H., Calders, T., Fromont, l., Goethals, B., Prado, A., Robardet, C.: An inductive database prototype based on virtual mining views. In: ACM Proc. of KDD (2008)
4. Richter, L., Wicker, J., Kessler, K., Kramer, S.: An inductive database and query language in the relational model. In: Proc. of EDBT, pp. 740–744. ACM, New York (2008)
5. Liu, H.C., Yu, J., Zeleznikow, J., Guan, Y.: A logic-based approach to mining inductive databases. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4487, pp. 270–277. Springer, Heidelberg (2007)
6. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco (1993)
7. Liu, H.C., Yu, J.: Algebraic equivalences of nested relational operators. Information Systems 30(3), 167–204 (2005)

Efficient Aggregate Licenses Validation in DRM

Amit Sachan[1], Sabu Emmanuel[1], and Mohan S. Kankanhalli[2]

[1] School of Computer Engineering, Nanyang Technological University, Singapore
[2] School of Computing, National University of Singapore, Singapore
amit0009@ntu.edu.sg, asemmanuel@ntu.edu.sg, mohan@comp.nus.edu.sg

Abstract. DRM systems involve multiple parties such as owner, distributors and consumers. The owner issues redistribution licenses to its distributors. Distributors in turn using their received redistribution licenses can issue new redistribution licenses to other distributors and usage licenses to consumers. For rights violation detection, all newly generated licenses must be validated against the redistribution license used to generate them. The validation becomes complex when there exist multiple redistribution licenses for a media. In such cases, it requires evaluation of an exponential number of validation equations with up to an exponential number of summation terms. To overcome this, we propose a prefix tree based method to do the validation efficiently. Experimental results show that our proposed method can reduce the validation time significantly.

1 Introduction

Digital rights management(DRM)systems generally [4][5] involve multiple parties such as owner, distributors and consumers. The owner gives the rights for redistribution of contents to distributors by issuing redistribution license. The distributors in turn use their received redistribution license to generate and issue new different types of redistribution licenses to their sub-distributors and usage licenses to consumers. A redistribution license allows a distributor to redistribute the content as per the permissions and constraints [6] specified in it. Thus, as part of the rights violation detection, it is necessary to validate these newly generated licenses against the received redistribution licenses with distributor. A validation authority does the validation of all the newly generated licenses.

Both redistribution(L_D) and usage licenses(L_U) for a content K are of the form: $\{K;\ P;\ I_1, I_2, ..., I_M;\ A\}$, where P represents permissions (e.g. play, copy, etc. [5]), I_i represents i^{th} ($1 \leq i \leq M$) instance based constraint and A represents aggregate constraint. Instance based constraints in redistribution licenses are in the form of ranges such as region allowed for distribution, etc. Instance based constraints in usage licenses such as expiry date, region allowed etc. may be in the form of a single value or range[5]. The range/value of an instance based constraint in the further generated redistribution and usage licenses using a redistribution license must be within the respective instance based constraint range in the redistribution license [5]. Aggregate constraint decides the number of *counts* that can be distributed or consumed using a redistribution or usage license respectively. The sum of *counts* in all the licenses generated using a redistribution license must not exceed the aggregate constraint $counts(A)$ in it.

For business flexibility, distributors may need to acquire multiple redistribution licenses for the same content[5]. In this case, if all instance based constraints' ranges/values in an issued license are within the respective constraint range in at least one redistribution license then the issued license is said to be instance based validated[5]. The problem of aggregate validation becomes harder in case a license can be instance based validated using more than one redistribution licenses(say a set S). This is because the validation authority needs to select a redistribution license from the set S for aggregate validation. Selecting a redistribution license randomly may cause potential loss to the distributors as shown in Sec. 2. Thus, we propose a better aggregate validation approach using validation equations in Sec. 2. But, the approach requires validation using exponential number of validation equations, containing up to exponential number of summation terms. This necessitates an efficient validation mechanism. So, we propose an efficient aggregate validation method using the prefix tree based structure[1][2][3]. The experiments show that our approach reduces the validation time and memory requirement significantly. To the best of our knowledge, the work presented in this paper is the first for the efficient aggregate licenses validation in DRM.

Rest of this paper is organized as follows. Section 2 discusses problem definition. Section 3 describes our proposed validation method. Section 4 presents the performance analysis. Finally, Section 5 concludes the paper.

2 Problem Definition

In case of multiple licenses, a newly generated license can be instance based validated using more than one redistribution license. For aggregate validation, selecting one redistribution license randomly out of multiple redistribution licenses may cause potential loss to the distributors as illustrated in example 1.

Example 1. Consider three redistribution licenses acquired by a distributor to distribute the play permissions according to two instance based constraints(validity period T, and region allowed R) and aggregate constraint A.
$L_D^1 = \{K; Play; I_D^1 : T = [10/03/09, 20/03/09], R = [X, Y]; A_D^1 = 2000\}$
$L_D^2 = \{K; Play; I_D^2 : T = [15/03/09, 25/03/09], R = [X]; A_D^2 = 1000\}$
$L_D^3 = \{K; Play; I_D^3 : T = [15/03/09, 30/03/09], R = [Y]; A_D^3 = 3000\}$

Now, the distributor generates the usage license $L_U^1 = \{K; Play; I_U^1 : T = [15/03/09, 19/03/09], R = [X]; A_U^1 = 800\}$. L_U^1 can be instance based validated using L_D^1 and L_D^2[5]. Let the validation authority randomly picks L_D^2 for aggregate validation then remaining *counts* in L_D^2 will be 200(i.e. 1000-800). Next, let the distributor generates $L_U^2 = \{K; Play; I_U^2 : T = [21/03/09, 24/03/09], R = [X]; A_U^2 = 400\}$. L_U^2 can only be instance based validated using L_D^2. The validation authority will now consider L_U^2 as invalid as L_D^2 now cannot be used to generate more than remaining 200 counts. In this case, a better solution would be to validate L_U^1 using L_D^1, and L_U^2 using L_D^2. This will result in both L_U^1 and

L_U^2 as valid licenses. Thus, the challenge is to do the aggregate validation such that the distributors can use their redistribution licenses in an intelligent way. We present a method to do the aggregate validation using validation equations.

A Method for Aggregate Validation: Let a distributor has N received redistribution licenses for a content and the set of redistribution licenses be represented as $S^N = [L_D^1, L_D^2, ..., L_D^N]$. Let $SB^r[S]$ denotes the r^{th} subset of the set S of redistribution licenses. Thus if a set S contains k received redistribution licenses then $r \leq 2^k - 1$. An issued license is said to belong to a set S if it can be instance based validated using all licenses in the set. E.g. L_U^1 in example 1 belongs to the set $[L_D^1, L_D^2]$.

Let $C[S]$ denotes the sum of permission counts in all previously issued licenses that belongs to the set S of redistribution licenses. Let $E^i[S]$ be the i^{th} redistribution license in the set S and $A(x)$ be the aggregate *count* in the received redistribution license x. For deriving the first equation, we use the fact that the aggregate of the *permission counts* in all previously issued licenses must not exceed the sum of the allowed *permission counts* in all the redistribution licenses with the distributor. Further, each valid issued license belongs to only one set of redistribution licenses out of the total $2^N - 1$ possible sets(due to N redistribution licenses). Therefore, in equation form we can write it as:

$$\sum_{r=1}^{2^N-1} C[SB^r[S^N]] \leq \sum_{i=1}^{N} A(E^i[S^N]) \tag{1}$$

The LHS of equation 1 represents the sum of *counts* in the issued licenses that belongs to the set formed by any possible combination of all N licenses(each possible combination can be represented by a subset of the set S^N). The RHS denotes the summation of maximum allowed *permission counts* in all the redistribution licenses with the distributor(as S^N is the set of all N licenses).

Although equation 1 can limit the *counts* issued in total using all the redistribution licenses but it cannot prevent the violation of individual licenses or set of licenses, which are proper subset of the set S^N, as shown in example 2 below.

Example 2. Consider the redistribution licenses in example 1. The above inequality ensures only the sum of all the play *counts* in the licenses issued must be less than 6000 i.e. $C_{Play}[L_D^1] + C_{Play}[L_D^2] + C_{Play}[L_D^3] + C_{Play}[L_D^1, L_D^2] + C_{Play}[L_D^1, L_D^3] + C_{Play}[L_D^2, L_D^3] + C_{Play}[L_D^1, L_D^2, L_D^3] \leq 6000$. But, it may not be able to prevent the violation due to issuing of excess counts for other combination of licenses. Equation 1 can be satisfied even if aggregate of the *counts* generated for play permission using only L_D^1becomes more than 2000 i.e. $C_{Play}[L_D^1] > 2000$, or using only L_D^1 and L_D^3 becomes more than 5000 i.e. $C_{Play}[L_D^1] + C_{Play}[L_D^3] + C_{Play}[L_D^1, L_D^3] > 5000$, but both these conditions are invalid. Thus, if there are N redistribution licenses then violation can happen in $2^N - 2$ ways(all proper subsets of S^N). So, we require an inequality for each subset of S^N. For r^{th} subset of set S^N, $SB^r[S^N]$, the validation equation is given as:

$$\sum_{l=1}^{2^m-1} C[SB^l[SB^r[S^N]]] \leq \sum_{i=1}^{m} A(E^i[SB^r[S^N]]) \tag{2}$$

where, $m = |SB^r[S^N]|$ is the cardinality of the set $SB^r[S^N]$. Equation 2 can be interpreted similar to equation 1 by replacing S^N by $SB^r[S^N]$. These inequalities ensure that in case of violation at least one inequality will not be satisfied.

Requirement of Efficient Aggregate Validation: If a newly generated license can be instance based validated using k number of redistribution licenses then the set formed due to k licenses will be present in $2^N - 2^{(N-k)}$ validation equations. Validation using such a large number of validation equations every time a new license is issued is computationally intensive. So, instead of doing validation online, we collect the logs of the sets of redistribution licenses(which issued licenses belong) and *permission counts* in issued licenses. We refer each entry corresponding to an issued license as a *record*. During the offline aggregate validation firstly different set *counts* can be aggregated and secondly the aggregated set *counts* can be applied to the validation equations. If M number of records are present, the time complexities for the first and second step would be $O(M * 2^N)$ and $O(3^N)$ respectively. The time complexities for both *set counts* aggregation and validation may be quite high for practical purposes. Thus, an efficient method is required to reduce the total validation time required.

3 Proposed Efficient Aggregate Validation Method

In this section, we present validation tree, a prefix tree based structure to do the validation of validation equations(equations 1 and 2) efficiently. The proposed structure and validation algorithm is based on the observation that validation equation for a set S aggregates the set counts of all the sets that are subset of the set S. The structure can compactly represent the log records for offline validation and use properties of prefix tree structure to do the validation efficiently.

Generation of *Validation Tree*: Initially, a *root* node is created. The tree is then expanded using the log records. Each node stores the following fields: name of a redistribution license(L), a count value(C), and links to the child nodes. The count value C determines the count associated with the set formed by the redistribution license in the node and all its prefix nodes(nodes in the path from the root node to current node). Redistribution licenses are indexed in the order they were acquired by distributor i.e. if L_D^j is acquired before L_D^k then $j < k$ and a redistribution license can act as a prefix only to the redistribution licenses having index greater than the index of redistribution license, as shown in Fig. 1. Child nodes of a node are ordered in increasing order of their indexes.

Records Insertion: Let the set of redistribution licenses in the record that needs to be inserted be given by:R=[r, R']and the *permission count* value be given by *count*, where, r is the first redistribution license and R' is the set of remaining redistribution licenses. Initially *root* node is allocated to T. Algorithm *Insert*(T, R, *count*) is used to insert the records in the *validation tree*. Fig. 1 shows the *validation tree* designed based on the Alg. 1 for the records in Fig. 3.

Validation using *Validation Tree*: To do the validation efficiently, we use the fact that if a set S_1 is not a subset of another set S then any superset S_2 of S_1 also cannot be a subset of the set S. Thus, in a prefix tree based structure, if

Algorithm 1. *Insert*(T, R=[r, R'], *count*)

```
1. If $T$ has a child $T'$ such that $T'.L=r$ then no action is taken.
2. Else add a node $T'$ such that $T'.L=r$ and $T'.C=0$ as the child node of $T$.
3. If $R'$=null set then $T'.C= T'.C+count$. Else, call Insert($T'$, $R'$, $count$).
```

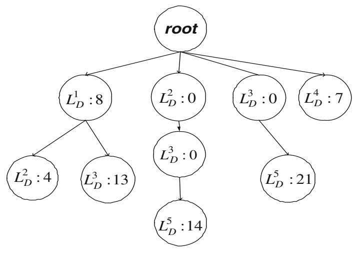


Fig. 1. Generation of *validation tree*

Serial Number	Set of redistribution licenses	Count
1	$[L_D^1]$	8
2	$[L_D^1, L_D^2]$	4
3	$[L_D^2, L_D^3, L_D^5]$	14
4	$[L_D^1, L_D^3]$	13
5	$[L_D^3, L_D^5]$	9
6	$[L_D^4]$	7
7	$[L_D^3, L_D^5]$	12

Fig. 2. Table of log records

the set of redistribution licenses formed by the redistribution license at a node N_1 and all its prefix nodes is not a subset of a set S then any other node having the node N_1 as a prefix node cannot be a subset of the set S. Thus, we need not to travel the child nodes of the node N_1. Another fact we use is that a set containing n licenses cannot be a subset of a set containing less than n licenses.

For N redistribution licenses, we can map each set of redistribution licenses corresponding to a validation equation into an N bits bit-vector. The bits from LSB to MSB correspond to a particular license with index from lower to higher. If a redistribution license is present in a set then the bit corresponding to it is 1 else it is 0. E.g. If N=10 then the bit-vector for the set $[L_D^1, L_D^3, L_D^9, L_D^{10}]$ will be 1100000101. So, if we consider bit-vectors in integer format then all possible sets can be represented using the values from 1 to $2^N - 1$.

Algorithm 2. Validation(T)

```
Temporary Variables: $A_v$=0, $C_v$=0
for $i$=1 to $2^N - 1$ do
    $licNumber$=0.
    for $j$=1 to $N$ do
        if $(1 << (j-1)$ AND $i) \neq 0$
        <<: left shift operator
        then
            $A_v = A_v + A(j)$.
            $licNumber = licNumber + 1$;
    Call $C_v = VaLHS(T, i, licNum)$.
    if $C_v \leq A_v$ then
        Print(Valid Equation).
    else
        Print(Invalid Equation).
```

Algorithm 3. *VaLHS*(T,B,$licNum$)

```
Subroutine: Process($T$,$B$,$licNum$) {
while ($licNum > 0$) do
    foreach child of $T$ do
        Let the current child be $T'$.
        Temporary variables: $i$ and $j$.
        $i$=index of license in node $T'$
        $j$=1<<($i-1$).
        if ($B$ AND $j \neq 0$) then
            $C_v = C_v + T'.C$.
            $licNum = licNum$-1.
            if ($licNum > 1$) then
                Call Process($T'$, $B$, $licNum$).
} Return($C_v$).
```

The *validation tree* is used to compute the LHS of the equations 1 and 2. RHS(let A) is calculated directly using redistribution licenses E.g. the RHS(A) for the set $[L_D^1, L_D^3, L_D^9, L_D^{10}]$ is given by: $A=A(L_D^1)+A(L_D^3)+A(L_D^9)+A(L_D^{10})$. Let T represents the *root* node initially, and B represents the bit-vector for the set of redistribution licenses for validation equation. The algorithm *Validation*(T) evokes the validation process for all possible validation equations. First, it calculates the RHS of each validation equation. Second, it calls $VaLHS(T,B,licNum)$to calculate the LHS. Finally, it compares the RHS and LHS to validate the equation. The algorithm $VaLHS(T,B,licNum)$ traverse the *validation tree* for the set of redistribution license determined by the bit-vector B. $licNum$ is the number of redistribution licenses in the set corresponding to the current validation equation. For illustration, consider the validation using validation equation for the set $[L_D^1,L_D^2,L_D^4]$ for validation tree in Fig. 1. B and $licNum$ for this set will be 01011 and 3 respectively. Let C_v denotes the LHS of the current validation equation for the set $[L_D^1,\ L_D^2,\ L_D^4]$, it is initialized to 0 for every validation equation. The algorithm traverses the nodes *root*, *root*$\rightarrow$ L_D^1, *root*$\rightarrow$ L_D^2, *root*$\rightarrow$ L_D^4 and *root*$\rightarrow$ $L_D^1 \rightarrow L_D^2$. The final value of C_v for this case is 19.

4 Performance Analysis

We performed experiments on on Intel(R) core(2) 2.40 GHZ CPU with 2 GB RAM. To perform the experiments, first we created a number of redistribution licenses and issued licenses. The set of redistribution licenses to which each issued license can be instance based validated along with the *permission counts* is saved in the log records. For experiments, each redistribution license is assumed to contain aggregate *permission counts* in between 5000 and 15000. Each issued license is assumed to contain permission counts in between 10 and 30.

We evaluate our proposal against the direct approach and a modified approach. In the direct approach, we scan the log records to find the subsets for the set corresponding to each validation equation and then aggregate their *set counts*. Whereas, in modified approach, we preprocess the log records to first aggregate the *set counts* for the sets containing the same redistribution licenses. Since, in practice many issued licenses may belong to the same set of redistribution licenses therefore scanning the modified log records would take lesser time. Table 1 summarizes the validation time performance. The experiments show that our proposed algorithm enhances the performance at least by 500 times and 10 times as compared to the direct and modified approach respectively. The large performance enhancement as compared to the direct method is mainly due the compact data representation in *validation tree*. However, in modified approach, the main reason for performance enhancement is the efficiency of our proposed subset search algorithm in the *validation tree*. Thus, it can be concluded that the *validation tree* gains efficiency both by compactly representing the log records and efficiently searching the subsets. Fig. 3 compares the performance in terms of memory requirement. The memory requirements for our approach is much less as the records can be stored in a much compact form in the *validation tree*.

5 Conclusion

Rights violation detection is an important security issue in DRM systems. In this paper, we presented a violation detection mechanism for distributors using the

Table 1. Comparison of validation time required(in milliseconds)

n	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Direct	.02	.05	.23	.80	2.12	5.80	15	39.33	93	188
Modified	0	.00031	.00094	.0034	.0122	.04	.11	.30	.74	1.64
Proposed	0	.00016	.00031	.0011	.0035	.01	.03	.06	.14	.33
n	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**
Direct	422	890	1953	4047	9078	21188	47906	108281	239953	524641
Modified	3.828	8.8	21.6	43.5	95	203	438	937	2063	4485
Proposed	.687	1.44	3.07	6.4	17	31	47	110	250	500
n	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
Direct	$1.2*10^6$	$2.6*10^6$	$5.4*10^6$	$1.2*10^7$	$2.6*10^7$	$5.6*10^7$	$1.3*10^8$	$2.9*10^8$	$6.1*10^8$	
Modified	9640	20922	43844	95640	195953	399281	840109	1761812	3810765	7648375
Proposed	1063	2219	4515	9563	19406	39422	79531	162938	328672	669360

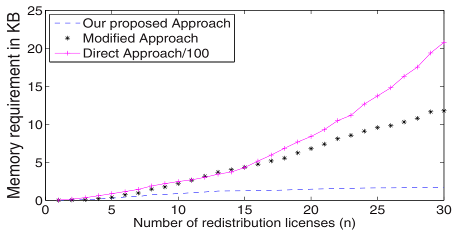


Fig. 3. Comparison of memory requirement

aggregate validation of licenses. However, large number of validation equations make the task difficult. Thus, we proposed '*validation tree*', a prefix tree based data structure to do the validation efficiently. The experiments show that *validation tree* performs better than the direct and a modified approach of validation in terms of validation time and memory requirements.

Acknowledgement. This work is supported by A*STAR Singapore, under project No: 0721010022, title:'Digital Rights Violation Detection for Digital Asset Management'.

References

1. Eavis, T., Zheng, X.: Multi-level frequent pattern mining. In: Database Systems for Advanced Applications (DASFAA), pp. 369–383 (2009)
2. Grahne, G., Zhu, J.: Fast algorithms for frequent itemset mining using FP-trees. IEEE Transactions on Knowledge and Data Engineering 17(10), 1347–1362 (2005)
3. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining and Knowledge Discovery 8(1), 53–87 (2004)
4. Hwang, S.O., Yoon, K.S., Jun, K.P., Lee, K.H.: Modeling and implementation of digital rights. The Journal of Systems and Software 73(3), 533–549 (2004)
5. Sachan, A., Emmanuel, S., Kankanhalli, M.S.: Efficient license validation in MPML DRM architecture. In: Proceedings of the 9th ACM workshop on digital rights management, Chicago, pp. 73–82 (2009)
6. Safavi-Naini, R., Sheppard, N.P., Uehara, T.: Import/export in digital rights management. In: Proceedings of the 4th ACM workshop on digital rights management, pp. 99–110 (2004)

Development Procedure of the Cloud-Based Applications

Masayoshi Hagiwara

Microsoft Company, Limited, Developer & Platform Evangelism, Tokyo Opera City Tower
3-20-2 Nishi-Shinjuku, Tokyo, 163-1445, Japan
masayh@microsoft.com

Abstract. The cloud-based applications are developed based on the loosely coupled, scale-out design, often accessing a key-value store. They work in parallel and asynchronously communicate between the nodes in the cloud, and between the nodes and its clients. By leveraging Service Oriented Architecture using Web or REST services, the queuing service and the service bus enable loosely coupled communications. Combined with this SOA, the object oriented technology, component oriented development and the relational data design can realize the scale-out design and the development of the cloud-based applications. However, the whole development procedure becomes so complicated. In this paper, we will show the development procedure and the analysis method of the cloud-based applications in a phased manner.

Keywords: Cloud computing, scale-out design, key-value store, development method.

1 Constraints of Developing the Cloud-Based Applications

There are 4 typical constraints in the development of cloud-based applications which the mechanism of the cloud causes.

1. Asynchronous communication

To survive partial failures of the system, cloud-based applications do not work in synchronous communications which halt the overall system because of the lifetime synchronization among the transaction participants. While we need to decompose data or transactions for scalability, we should deal with the consistency between decomposed data or sub-transactions by using synchronous communications. However since we cannot practically use synchronous communications in view of partial failures, we need to weaken the consistency conditions of the applications.

2. Constant availability

The cloud-based applications should always accept update requests of data and at the same time cannot block the updates even while they read the same data for scalability reasons. This means the cloud-based applications may not read up-to-date data.

3. CAP (Consistency, Availability, Partition tolerance) theorem

The CAP theorem shows that a cloud can provide availability and partition tolerance by the sacrifice of consistency. Thus, we need the different way of obtaining consistency from existing ACID (Atomicity, Consistency, Isolation, Durability) transactions. Eventual (not immediate) consistency is one way of such weaker consistency requirements.

4. Key-value store

Instead of on-premise mainstream RDBs, the basic data service of the cloud is a key-value store. A key-value store requires different data design, transaction implementations, and data management. It scatters each row of the key-value store on each node in the cloud, and provides only a row-level ACID transaction without supporting a distributed transaction. This leads to the scale-out design.

2 Development Procedure of the Cloud-Based Applications

This is an overview of the development procedure of the cloud-based applications based on scale-out design with data partitioning.

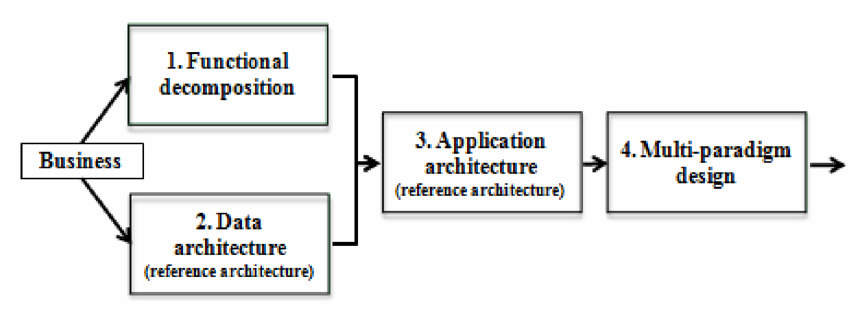

Business
1. Functional decomposition
2. Data architecture (reference architecture)
3. Application architecture (reference architecture)
4. Multi-paradigm design

Fig. 1. The development procedure of the cloud-based applications based on the scale-out design with data partitioning

1. Functional decomposition

To avoid bottlenecks in the data layer by means of a key-value store, we need to decompose the functions of an application to decide what workload or transaction pattern to de-normalize the data for the optimal data access if we adopt the scale-out design with data partitioning.

2. Data architecture

In parallel with 1, we will design the data architecture independent of the application requirements. In doing so, we should identify master and transaction data by category identification of concept. At this point, what matters most is to logically manage master data in one place and transaction data should reference master data unidirectionally. This procedure is basically followed by a data-centric approach.

3. Application architecture

We should adopt the reference application architecture with asynchronous communications and maximizing transaction throughput while we make much of the availability and maintenance. To succeed at the scale-out design with data partitioning, we will decompose master data, and replicate the part of master data to transaction data if necessary. Next we will de-normalize transaction data following the workloads of an application. Then, we will horizontally partition the de-normalized data into rows to deploy them onto the different nodes in the cloud since a key-value store puts the separate rows on different nodes to access them in parallel. For additional scalability, we will sometimes decompose one transaction into sub-transactions. On the other hand, we need to give consistency between the decomposed data, sub-transactions considering the trade-off between scalability and consistency.

4. Multi-paradigm design

We will design an application on a component-oriented basis using the application architecture constructed in 3 and adopt the multi-paradigm design for the change (variability) of application requirements. The multi-paradigm design is a flexible design methodology by choosing the most appropriate paradigm for a domain or a concern in the design phase or in the implementation phase. One example is to use the combination of the object oriented paradigm for the variability of data structures or logic implementations, the functional paradigm for the distributed batch processing, the component oriented paradigm for the deployment of change-sets of the artifacts, and the service oriented paradigm for providing the external reusable functions of an application.

2.1 Analysis Method of the Cloud-Based Applications

To derive the application architecture in 3 from the data architecture in 2, the following analysis method will be effective. Here we will focus on the relationships of

— Data-data
— Data-transaction
— Transaction-transaction

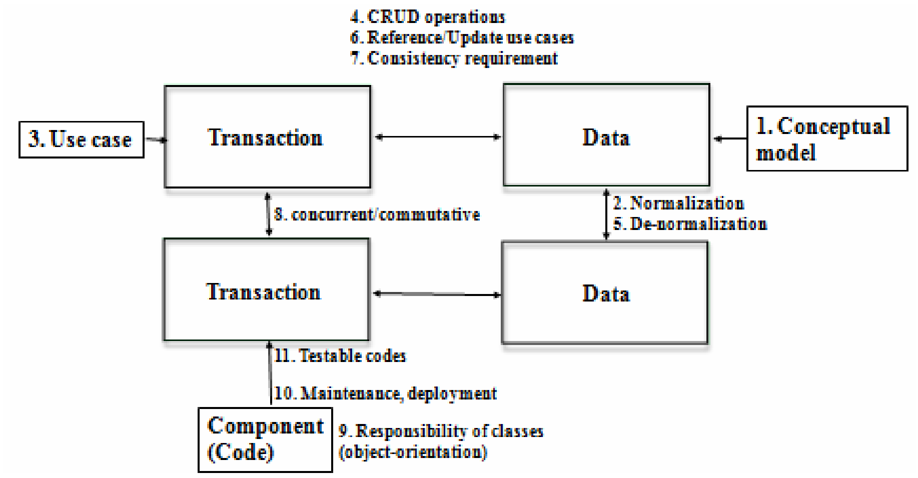

4. CRUD operations
6. Reference/Update use cases
7. Consistency requirement
3. Use case
Transaction
Data
1. Conceptual model
8. concurrent/commutative
2. Normalization
5. De-normalization
Transaction
Data
11. Testable codes
10. Maintenance, deployment
Component (Code)
9. Responsibility of classes (object-orientation)

Fig. 2. A transaction here indicates the application requirements to specify the consistency requirements

We think that we can model the consistency requirements of an application as the transactions of an application. A transaction, which does not mean an ACID transaction here, indicates the application requirements on consistency. This is kind of a non-functional requirement while a use case indicates functional requirements from business viewpoint. The conventional object oriented analysis and design method only adopts a use case to define the application functional requirements. Though we need to specify the consistency requirements as the part of application requirements for the cloud-based applications because we must follow CAP theorem by which the cloud supports the scalability and availability in the sacrifice of the consistency, a use case does not specify the application requirements sufficiently in this sense. This is one of the differences in developing the cloud-based applications from developing usual applications using the object oriented method. This transaction, however, does not yet decide how to implement components and classes. So the transaction is more abstract than the implementation.

The analysis follows the order indicated:

1. Define the logical data from the conceptual models.
2. Normalize the logical data by the functional dependencies.
3. Independent of 1 and 2, define the transaction(s) from a use case.
4. Identify CRUD (Create/Read/Update/Delete) operations of a transaction.
5. De-normalize the normalized data defined in 2 based on the operations of the transaction.
6. Divide the reference use cases (transactions) form the update ones.
7. Specify the consistency requirement of the transaction, and estimate de-normalized data in access frequency.
8. Identify the concurrent, commutative conditions between transactions.
9. Decide the responsibility of the classes implementing the transaction (in case of object-orientation).
10. Define the component(s) implementing the transaction in terms of its maintenance and deployment.
11. Make the source code testable for TDD (Test Driven Development).

For example, the e-commerce site of reviewing the movies has the movie titles, the review data, and the user data in 1. The use cases of the site are to add the user's review to the movie, to query the reviews of the particular movie and to query the reviews of the particular user in 3. The site is mostly for queries. This characteristic workload requires the data to be the optimal design for queries. Thus, we should have both movie-review and user-review de-normalized data to improve two kinds of query use cases in 5. This can be done by duplicating the review data for the movie and the user. As the side effects, when adding the user's review to the movie, we need to add the same review both to the movie-review and user-review data in 7. This will degrade the performance of add operations. However, this degradation is justified by the improvement of queries. This example shows that 3 kinds of relationships in Fig. 2 are complexly intertwined.

The cloud-based applications cannot rely on immediate consistency due to CAP theorem. This implies that an application may not read up-to-date data even after it updates the same data. For this reason, it is preferable to divide the reference/read use cases from the update/write ones. This is also known as a “command query separation” design pattern. In the abstract, the separation of the fact from its recognition is one of the most important design philosophies in software. Similarly there are the separation of the expressions from the meaning, and the separation of the views from the content/model, which we can see in Model-View-Controller architecture style. In principle, the "write" use cases record the fact of events such as - order entry, customer call, the price changed, and the payment. The "read" use cases reference the facts of the "read" use cases and recognize/judge the facts to make some decisions. For example, a decision maker takes an action for the order entry in the form of the sales report after batch processing of the daily orders. In most business cases, delayed business processes using batch processing, or following the business rules are allowed. Thus, weaker consistency requirements on which eventual consistency and other consistency models impose do not prevent business applications from being used on the cloud. The problems are excessive expectation for reliability which an ACID transaction is supposed to give, and the lack of capability of treating weaker consistency models by the frameworks and development environments based on an ACID transaction. In comparison, eventual consistency tolerates partial failures well and improves the concurrent access capability since an ACID transaction blocks the concurrent transactions.

On the other hand, for distributed transactions, we should devise application-level distributed update procedures instead assuming a key-value store does not support distributed transactions in terms of scalability and availability. One way of distributed update procedures is to take advantage of the unique key constraint provided by a key-value store. The unique key constraint enables us to insert a transaction request having the unique identifier only once. This is equivalent to the exact once semantic of the message exchange. And distributed update procedures are possible by updating distributed data among the transaction participants exactly once based on the inserted transaction requests instead of distributed transactions. More formal specification is available when we specify the update procedures by the protocol machine defined by state charts.

2.2 Design Problems Regarding the Use Case

The use cases of a cloud-based application are related with many design problems. Below are 5 big problems of them:

1. The use cases decide the design of de-normalized and horizontally partitioned data.
2. The use cases are classified into reference and update ones to enable eventual consistency.
3. One use case is built by boundary and domain logical models. The boundary model provides a service interface and the domain model provides common core business logics.

4. One use case is executed on the front-end and back-end physical programming models. The front-end model plays roles of Web applications or Web services and the back-end model plays roles of other business logic executors in terms of the programming models which a particular cloud provides. These models are implementation-specific of the cloud.
5. One use case is deployed on two transaction systems, one in the cloud, the other on-premise.

1 and 2 are specified external to the system, and 3, 4 and 5 are specified internal to the system.

2.3 Requirements of Consistency and Its Implementation

The consistency requirement for which one transaction is responsible is specified by the consistency model. The consistency model covers the range of requirements that an application has. The spectrum of the consistency requirements forms a variety of consistency models from weak to strong requirement models. One of the consistency models is eventual consistency.

The concurrency model, on the other hand, is a means of implementing consistency requirements, that is, consistency models. An ACID transaction is one kind of concurrency models. The other examples of concurrency models are transactional replication, queuing, and reader/writer lock. The concurrency model is chosen suitably by the design of application architecture to satisfy the consistency models that the cloud-based applications require.

The implementation of the concurrency models are illustrated by the examples from the concurrency control provided by data stores, for instance, a row-level ACID transaction by a key-value store, to a reader/writer lock design pattern inside the Web service implementation, and the queuing by which Web service requests are serialized at a Web service interface. And a cloud-based application client as the Web service consumer, runs the pipeline of functions which work concurrently and in parallel to consume the Web services provided by the cloud. This example is the hierarchy of the concurrency models in the cloud-based application architecture which affects the decision of an overall scalability design.

3 Conclusion

We face the complication of composing a lot of software technologies and development practices in developing the cloud-based applications. Data-centric approach for data design, service orientation for domain and functional decomposition, object orientation for logic design, component orientation for maintenance and deployment, and use cases to drive the overall development process. In addition, we must consider functional paradigm for parallel executions and asynchronous communications, and multi-paradigm design for the variability of application requirements. Here among others, we should see asynchronism and inconsistency that global large-scale distributed systems inherently have. These natural “features” make synchronous communications and ACID transactions ideal assumptions to simplify the system design. The development of the cloud-based applications breaks these assumptions.

The development procedure in this paper is one of the realistic methods which do not assume these ideal situations.

References

1. Hagiwara, M.: An aesthetic sense of the architects. Shoeisha (2009) (in Japanese)
2. Hagiwara, M.: Architecture and Development methodology using Windows Azure in a cloud era. System Development Journal 10 (2009) (in Japanese)
3. Helland, P.: Life Beyond Distributed Transactions (2007)
4. Pritchett, D.: BASE An ACID Alternative. ACM Queue 6(3) (2008)
5. Stonebraker, M., et al.: The End of an Architectural Era (It's Time for a Complete Rewrite)

Rapid Development of Web Applications by Introducing Database Systems with Web APIs

Takeru Inoue[1], Hiroshi Asakura[2], Yukio Uematsu[2], Hiroshi Sato[1], and Noriyuki Takahashi[1]

[1] NTT Network Innovation Laboratories
takeru.inoue@ieee.org
[2] Innovative IP Architecture Center, NTT Communications Corporation

Abstract. Web APIs are offered in many Web sites for Ajax and mashup, but they have been developed independently since there is no reusable database component matched to Web applications. In this paper, we propose WapDB, a novel database management system for rapid development of Web applications. WapDB is designed on Atom, a set of Web API standards, and provides several features required for Web applications, including efficient access control, an easy extension mechanism, and search and statistics capabilities. By introducing WapDB, developers are freed from the need to implement these features as well as Web API processing. In addition, the design totally follows the REST architectural style, which gives uniformity and scalability to applications. We develop a proof-of-concept application with WapDB, and find that it offers great cost effectiveness with no significant impact on performance; in our experiments, the development cost is reduced to less than half with the overhead (in use) of just a few msec in response times. WapDB is being used to develop new services in NTT Communications.

1 Introduction

Web APIs are a set of interfaces designed to support interoperable machine-to-machine interaction over HTTP. They have been developed for Ajax as well as mashup-based Web applications. Ajax improves the user's perception of interactivity by enabling clients (typically, JavaScript codes) to retrieve data from a server's API without blocking the user's interaction. Mashup-based applications combine data retrieved from multiple Web APIs to create a new service. Mashup has become popular in the last few years by virtue of its rapid integration.

In the early 2000's, developers had to design their own Web APIs. These APIs permitted similar CRUD operations against Web resources, but they differed from each other because of the lack of standards. In the middle of the 2000's, IETF released a set of standards called Atom [1,2], which offers a unified approach to Web APIs for the CRUD operations. Recently, several Web APIs have been designed around Atom.

Unfortunately, developers are still forced to develop their own data store. Some data stores equipped with Atom have been developed (e.g. mod_atom [3] and AtomServer [4]), but they provide neither efficient access control nor easy extension methods. The lack of access control yields inefficient filtering of outside data stores against search results, and the lack of an extension method prevents developers from adding extra functionalities. As a result, Atom stores are rarely used, thereby increasing the development cost of applications with Web APIs. We believe that this continued reinvention of the wheel should (and can) be avoided.

In this paper, we propose a novel database management system called WapDB for the rapid development of Web applications with APIs. Since applications are allowed to interact with WapDB directly via a Web API (an extended Atom protocol), there is no need to translate the API into SQL; developers are released from the need to re-implement the API processes. Moreover, WapDB has the features essential for Web application development including an efficient access control model, an easy extension mechanism, and search and statistics capabilities. It is designed following the REST style [5], which gives uniformity and scalability to applications. We implement this new database management system and develop a proof-of-concept application, a photo-sharing service, with it. Experiments reveal that it offers great cost effectiveness with no significant impact on performance.

The design concept of WapDB is largely oriented towards applications like social networking and photo-sharing services, since they often offer Web APIs. We, however, believe that it will be adopted by a wide variety of other services because of its generality. We focus on applications in which data is accessed mostly through Web APIs. Such an architecture brings design simplicity as well as better interactivity [6].

2 Background

2.1 Web Application Architectures

Figure 1(a) depicts an example of traditional Web sites that follow the multi-tier architecture. Client requests are processed by a logical tier component (e.g. an application server), which makes responses by retrieving resources from a data tier component (e.g. a RDBMS). While the logic tier component implements its own business logic that depends on service, the data tier component provides common CRUD operations through well-defined interfaces (e.g. SQL). The data tier component can be reused, thereby reducing the development cost.

Web applications that provide APIs have a different architecture, as shown in Fig. 1(b). While the whole server-side works as a data tier, business logic is moved to the client-side. Clients (typically, JavaScript codes running on a Web browser) are allowed to dynamically update pages by requesting for resources through the API. Unfortunately, application servers are not reusable despite their similarity.

2.2 Atom Format and Protocol

Atom is often called a RESTful Web API, since it follows the REpresentational State Transfer (REST) architectural style [5]. In RESTful APIs, operations are

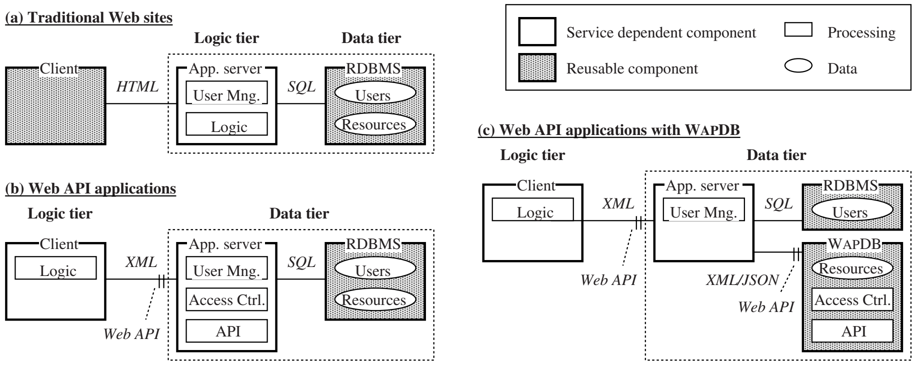


Fig. 1. Examples of Web site architectures. (a) The traditional architecture places business logic in the server-side logic tier. (b) The architecture for Web API applications moves business logic to the client-side, and the application server handles access control and API processing. (c) The architecture of WapDB (discussed in 4) makes access control and API processing reusable, thereby reducing the development cost for the application server.

simply specified by pairs of an HTTP method and a URI. Servers are not allowed to keep any state for clients for parallel processing. RESTful Web APIs have been gaining popularity because of their simplicity, interoperability, and scalability.

Atom is the only RESTful standard for which a protocol as well as a format has been published. The CRUD protocol is called Atom Publishing Protocol (AtomPub) [2], and the format is referred to as Atom XML [1]. Atom is highly extensible, because it is designed as a basic instruction set for building other Web APIs; its extensions include Google Data APIs (GData) and Windows Azure REST API. Figure 2(a) presents the abstract resource model in Atom; data called *members* are organized in containers called *collections*. They are linked as shown in Fig. 2(b); the *service document* has links to collection *feeds*, which link to members in the same collection. For member creation, only a media resource is transferred from a client to the server, which generates the associated media link entry automatically. It is not allowed to create, update, or delete multiple members at a time.

3 Requirements

Before presenting the requirements for WapDB, we briefly discuss the data model and ACID properties in the REST style. Web resources follow a hierarchical model, not the relational model, as Atom follows a fixed hierarchical model (i.e. collection and member). Atomicity (A of ACID) is limited to transactions of a single operation, because RESTful servers are not allowed to keep state between operations.

The following are the requirements faced by WapDB.

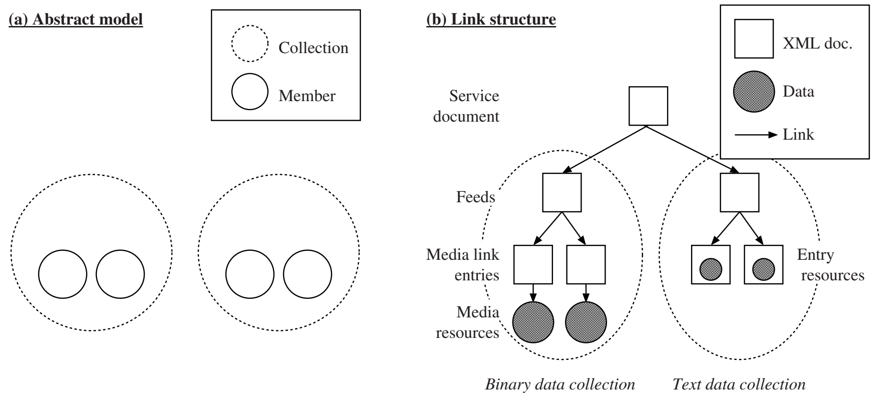


Fig. 2. Resource model in Atom. (a) Members are organized in collections in the abstract resource model. (b) The service document links to collection feeds, which link to members in the same collection. Each member consists of a *media resource* (data itself) and a *media link entry* (metadata written in Atom XML), as depicted in the left-side collection. If the data is a text resource like HTML, it can be embedded in a content element in Atom XML, as shown in the right-side collection.

Reusability. Improving this property is the main force motivating this paper, since it reduces the development cost. As shown in Fig. 1, WapDB is required to offer Web API processing and access control mechanisms. In order to reuse WapDB in actual services, their own requirements such as advertising, search, and statistics, are allowed to be incorporated.

Efficient access control. WapDB cannot be used for account management because of the limitations placed on transactions. This implies that user accounts are managed in a traditional RDBMS, while Web resources are stored in WapDB. Since access control requires account information as well as Web resources, distributed join operations are inevitable between the RDBMS and WapDB. Efficient join operations need a query execution plan that minimizes communication costs (e.g. traffic and round trips) [7, Chap. 7–9]. In addition, the access control model should take advantage of user relationships (e.g. friends).

Easy extension. WapDB is required to provide an easy function extension mechanism. Extension should be possible without source code. Moreover, no restrictions should be placed on the programming languages used to develop extensions. In this paper, we assume that WapDB would be extended mainly for resource modifications in CRUD operations, such as advertising and metadata addition.

Search and statistics. Search and statistics are essential functions in most Web applications. They should be built-in since they require additional indexes and tables, even though WapDB should have an extension mechanism. Furthermore, it is preferable that they be simple and suit Web resources.

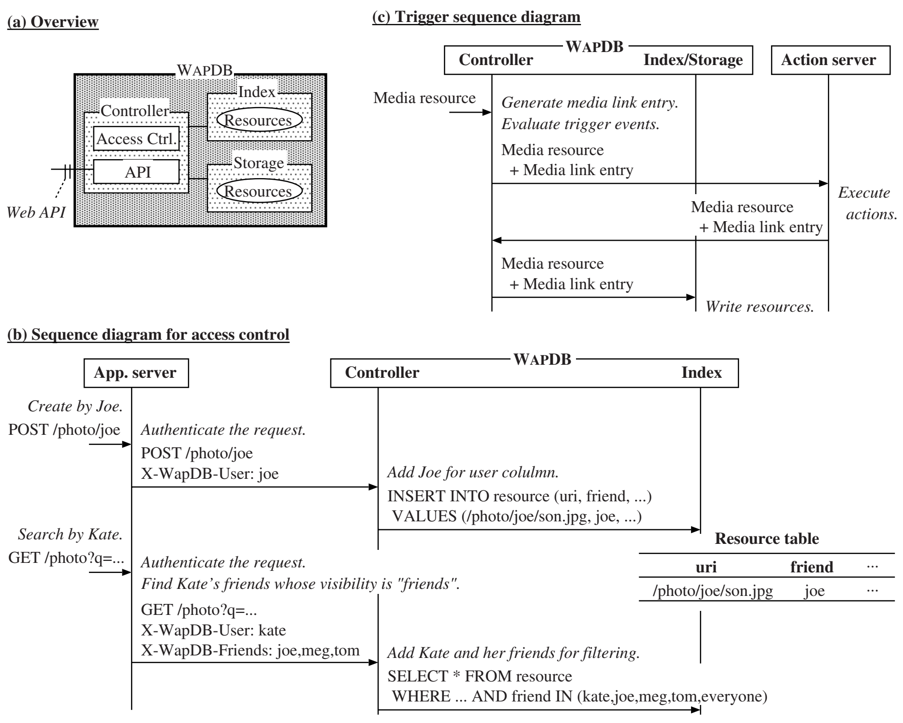


Fig. 3. Design of WapDB. (a) WapDB consists of a controller that handles access control and API processing, and a full-text index and storage that together manage Web resources. (b) Our access control model, which efficiently executes distributed join operations; Joe sets "friends" visibility to his resources, which allows Kate (Joe's friend) to access them in this example. (c) Trigger-like extension mechanism, in which the posted resource is transferred to the action server that provides extensions if the request matches the corresponding event.

4 Design

4.1 Architecture for Reusability

WapDB prevents application servers from handling Web APIs, thereby reducing the development cost. As shown in Fig. 1(c), WapDB is placed behind the application server. Upon receiving a request for Web resources in WapDB, the application server authenticates the request if needed, and forwards it to WapDB with additional headers for access control as described in Sect. 4.2.

Figure 3(a) shows the configuration of WapDB. The controller is an HTTP server that executes access control and API processing. The full-text index (a search engine) and storage are used to manage Web resources. They can be a single component or distributed among multiple machines (fragmentation of the index and/or storage is beyond the scope of this paper). For a write query, the

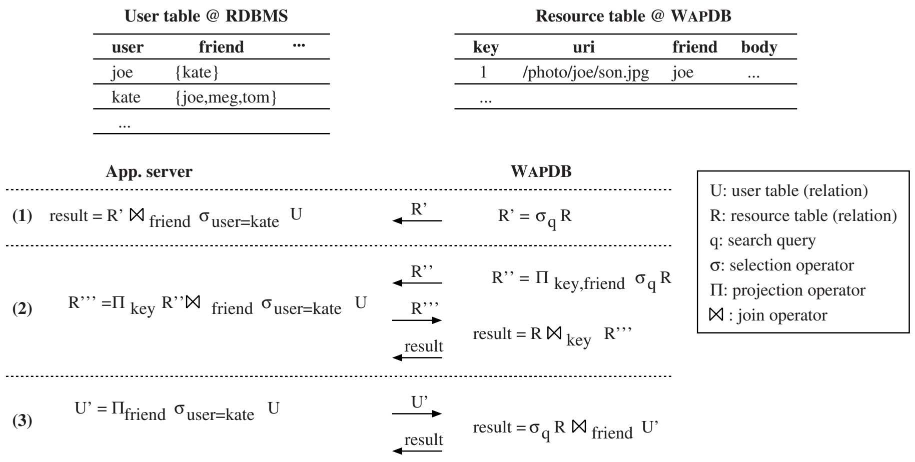


Fig. 4. Three possible execution plans of distributed join operations for access control in the example of Kate's finding her friends' resources. The operations are managed by the application server between the user table in the RDBMS and the resource table in WapDB (the tables are not normalized due to space limitations). (1)st plan creates excessive traffic R'. (2)nd plan requires an extra round trip. (3)rd plan reduces the traffic and requires no extra round trip.

controller updates the index and saves the posted resource into the storage. For a search query, the controller searches the index and retrieves the corresponding resources from the storage.

This architecture (Fig. 1(c) with Fig. 3(a)) degrades latency, since messages must pass through more machines. The impact of this is evaluated in the experiments in Sect. 5.

4.2 Efficient Access Control Model

Assuming that users are allowed to access the resources of their friends, we first discuss a query execution plan for distributed join operations. Account information is managed in a RDBMS, while Web resources are stored in WapDB. A write query poses no significant difficulty, since it involves just a single resource. On the other hand, a search query can involve a large amount of resources, and so an efficient plan is imperative for minimizing the cost of the distributed join operation. Figure 4 shows three possible plans.

1. The first plan is sending the entire set of matched resources to the application server from WapDB, and filtering them at the server. Unfortunately, this creates excessive traffic, R', between the server and WapDB.
2. The second plan utilizes semi-join; it returns only the primary key and friend columns; filtering is done at the server followed by retrieving the body of the desired resources from WapDB. This plan reduces the traffic unlike the first plan, but does require an extra round trip.

3. The third plan is sending user's account information (e.g. friend list) to WapDB and filtering resources in the search process. The account information, U', is much smaller than search results, R', and no extra round trip is required. Moreover, complicated joins do not need to be implemented in the application server.

We choose the third plan, since it yields low communication cost for most queries. The existing Atom stores, such as mod_atom and AtomServer, cannot execute our plan since it requires the join operation shown in Fig. 4(3).

Next, we discuss the detailed behavior of our access control model in Fig. 3(b). In the model, a user has the privilege of setting the visibility of all his/her resources. The visibility has a three-grade scale, "only me", "friends", or "everyone". The visibility is stored in the RDBMS with account information. For a write query, the application server authenticates the query, and forwards it with the X-WapDB-User header containing the authenticated user name. WapDB finally stores the resource into the resource table; the friend column contains the reserved term indicating "everyone", otherwise the user name. For a search query, the server authenticates the query, and forwards it with the X-WapDB-User and the X-WapDB-Friends headers. The latter includes the user's friends whose visibility is "friends". Finally, WapDB executes a search over resources whose friend column contains the user name, one of his/her friends, or the reserved term for "everyone", by adding a filtering predicate to the original query (`friend IN (...)` in Fig. 3(b)).

Our model follows the REST style, since it introduces no state management.

4.3 Trigger-Like Easy Extension

WapDB provides easy extension through a trigger-like mechanism. Triggers are commonly supported in RDBMSs; SQL actions are executed in response to certain events if given conditions are met. They are often used to log changes and execute business rules. Since WapDB follows the REST style, our triggers are provided in the RESTful way; events are specified by a pair of HTTP method and URI, and actions are implemented as HTTP servers. For simplicity, conditions are not specified. Triggers of WapDB are row triggers, not statement triggers, since it is not allowed to edit multiple resources (rows) at a time.

Figure 3(c) is an example of trigger processing upon receipt of a request for creating a new resource. WapDB controller evaluates the posted resource, and generates an associated media link entry if needed. The resources are forwarded to all action servers whose events are matched to the request. The action servers process the resources, which are returned to the controller. The controller finally saves the returned resources into the index and storage.

Since WapDB and action servers are loosely coupled through HTTP, the programming languages used are not restricted. Trigger overheads, extra round-trips and other XML parsing operations, are evaluated in the experiments.

Table 1. Key query parameters in GData

Name	Description
`q`	Full-text query string.
`/-/<category>`	Category filter.
`author`	Entry author.
`updated-min, updated-max`	Bounds on the entry update date.
`start-index`	1-based index of the first result to be retrieved.
`max-results`	Maximum number of results to be retrieved.

4.4 Search Queries and Frequency Distribution

WapDB uses the GData query model because it has a proven track record, has been operational for several years in Google, as well as being simple and Atom-friendly. Table 1 presents some of the GData query parameters (they are expressed as HTTP URIs). Clients are allowed to select resources with full-text search and range queries for key Atom elements. Results are sorted in descending order of last modified time. If the number of results per page is limited, a link to the next page is given with the `start-index` parameter in the results.

WapDB also provides frequency distribution of categories for every collection. Frequency Distribution is a basic statistic; it is often used to create rankings and tag clouds. The distributions are shown in the service document.

5 Experiments

5.1 Implementation

We implemented a prototype of WapDB on the LAMP (Linux, Apache, MySQL, Perl) stack with the search engine named infobee/evangelist [8]. MySQL was used as the storage engine. We also developed a proof-of-concept application, a photo-sharing service (Fig. 5(a)). WapDB manages photos (media resources) and their metadata (media link entries). Two triggers are fired on creating and updating resources. One is for advertising; it inserts an ad matched to the photo's category into the media link entry. The other trigger extracts the location from the photo's Exif data, and adds it to the associated media link entry. The application works well with the Atom client named Windows Live Writer.

The server-side consists of three machines; each of them runs an application server with the RDBMS, three components of WapDB, and two action servers. All machines have a quad-core processor (Intel Xeon 2.13 GHz), and are connected by Gigabit Ethernet.

5.2 Evaluation

We developed the application with and without WapDB, which are based on architectures of Fig. 1(c) and Fig. 1(b), to evaluate the cost effectiveness and performance overhead of our proposal. Figure 5(b) shows the lines of code needed to

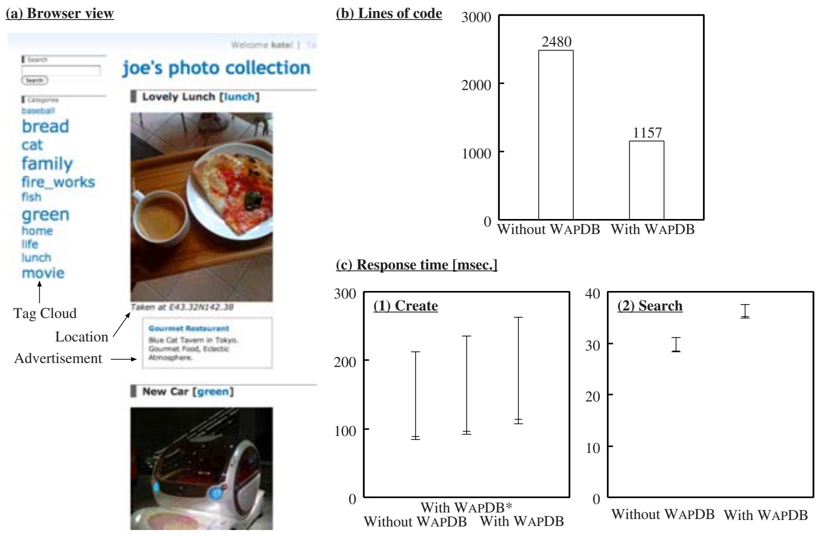


Fig. 5. Experimental results. (a) An image of the proof-of-concept application. Photos and tag cloud are drawn by Ajax through Web API. An advertisement matched to the photo categories and the location at which the photo was taken, are added by triggers. (b) Lines of code for the proof-of-concept application, developed with and without WapDB. (c) Response time in the application with and without WapDB (1st, 50th, and 99th percentiles are plotted). (c-1) Response time for resource creation. (c-2) Response time for a category query.

create the application. The number of lines was reduced to less than half by introducing WapDB in our experiments. This is mainly because WapDB eliminates the need to implement Atom processing, access control, search, and statistics. We implemented only business logic at the client, account management and header manipulations for access control at the application server. Actual development period was reduced by roughly half. We found no difficulty in developing the action servers thanks to recent advances in Web application frameworks (e.g. Ruby on Rails and Catalyst); it required only 168 lines of code.

Next, we observed the response time for evaluating performance overhead. The response time was measured at the client for creating a resource and searching for the resources of a specified category (i.e. `/-/<category>` in Table 1). While the posted resources were photos with average size of 120 KB, the associated media link entries were 4.0 KB in size on average. The client received top 10 entries at most for each category query. We repeated the measurements 1000 times and determined the 1st, 50th, and 99th percentiles.

In addition to implementing the application twice (i.e. with/without WapDB), we introduced another implementation named WapDB* in order to distinguish between the overhead of the new architecture and that of triggers. In this implementation, the application was developed with WapDB based on the architecture of Fig. 1(c), but triggers were not used (advertising and Exif extractor were implemented in the application server). For resource creation, new architecture overhead is the difference between response time of "without WapDB" and that of "with WapDB*", while trigger overhead is the difference between "with WapDB*" and "with WapDB". In terms of searching, new architecture overhead is the difference between "without WapDB" and "with WapDB".

Figure 5(c) depicts the response time for each implementation. The figure shows that the overhead of WapDB is not significant; on average, new architecture overhead is 7.5 and 6.7 msec for creation and search, respectively, while trigger overhead is 22.0 msec. The impact of searching can be considered even smaller in practice, because some results would be retrieved from cache servers. Although trigger overhead is slightly larger, it does not matter because write queries are much less common than read ones.

6 Conclusions

This paper introduced WapDB, a novel database management system with Web APIs for the rapid development of Web applications. We designed WapDB as a highly reusable database component for reducing the development cost. WapDB removes from developers the need to reinvent Web API processes. In addition, WapDB provides common features in Web applications, such as efficient access control, an extension mechanism, and search and statistics capabilities. We developed a photo-sharing service by using WapDB. Experiments revealed the great cost effectiveness provided by WapDB; the number of lines was reduced to less than half. In addition, the performance overhead was no more than ten msec for searching, and several tens of msec for creating a resource even if triggers were used. We are currently developing new services with WapDB at NTT Communications. Finally, we would like to thank Dr. Onizuka, Mr. Okabe, and Mr. Sawamura for their kind support.

References

1. Nottingham, M., Sayre, R.: The Atom Syndication Format. RFC 4287 (2005)
2. Gregorio, J., de hOra, B.: The atom publishing protocol. RFC 5023 (2007)
3. Bray, T.: mod_atom. ongoing (2007), http://www.tbray.org/ongoing/When/200x/2007/06/25/mod_atom
4. Jacob, B., Berry, C.: AtomServer – the power of publishing for data distribution. InfoQ (2008), http://www.infoq.com/articles/atomserver
5. Fielding, R.T., Taylor, R.N.: Principled design of the modern Web architecture. ACM Trans. Internet Technol. 2(2), 115–150 (2002)
6. Kuuskeri, J., Mikkonen, T.: Partitioning Web applications between the server and the client. In: Proceedings of the 2009 ACM symposium on Applied Computing, pp. 647–652 (2009)
7. Özsu, M., Valduriez, P.: Principles of Distributed Database Systems, 2nd edn. Prentice-Hall, Englewood Cliffs (1999)
8. Inagaki, H., Mori, D., Sugizaki, M., Takeno, H.: Japanese Internet portal-site www.goo.ne.jp powered by InfoBee technology. In: Proceedings of the International Conference on Digital Libraries: Research and Practice, pp. 197–202 (2000)

A General Maturity Model and Reference Architecture for SaaS Service

Seungseok Kang[1], Jaeseok Myung[1], Jongheum Yeon[1],
Seong-wook Ha[2], Taehyung Cho[2], Ji-man Chung[2], and Sang-goo Lee[1]

[1] Department of Computer Science and Engineering, Seoul National Univ., Seoul, Korea
{pyxis81,jsmyung,jonghm,sglee}@europa.snu.ac.kr
[2] Q.N.SOLV Corporation, Seoul, Korea
{suha,thcho,comgen27}@qnsolv.com

Abstract. In today's dynamic IT environment with increased global competition, enterprises must achieve greater business agility and decrease the TCO (Total Cost of Ownership) of their system for service. As the need for innovative software circulation process emerges, SaaS (Software as a Service) is introduced for integrating software service framework. But most of current ASP (Application Service Provider) Players have difficulties to migrate their systems to SaaS Platform for the lack of maturity model and process. In this paper, we have surveyed several cases of SaaS service, and we identified the common key functions of SaaS service. We contend that the practical maturity model is a key enabler for achieving migration to innovative SaaS service platform. To assist in building our SaaS maturity model, we defined two important axes of maturity model and introduced the detailed components of each phase with the reference architecture which contains the essential activities according to the common functions of SaaS service.

Keywords: SaaS Service, ASP, Maturity Model, Software Business.

1 Introduction

The economical, social, and technological development in the computing environment triggers big changes in the software business. The emergence of Web and its infrastructure has become an integral part of enterprise computing and continues to give both new opportunities and challenges to the software providing vendors. Intuitively, the development of IT technology and business environment gives a significant influence on software business as well as software itself. Traditional business paradigm on software is evolved due to the rapid change of IT technology, and it also brings the change of the type of service and technology. This paradigm shift in software business is categorized into four key parts of core service component such as data, system, service, and business.

The emerging of SaaS is one of the results which are derived by paradigm shift. Based on the changes of software business environment, a number of current ASP players have tried to advance to new paradigm with SaaS service component. However, there are still some challenging issues due to the lack of adequate growth strategy and proper guideline for adapting the characteristics of SaaS service in their current service model. In this respect, we believe that a SaaS service architecture based on a general SaaS maturity model is an enabler of a myriad of current ASP players who want to adapt the concept of SaaS service.

In this paper, we survey several cases of current SaaS service, and identify the common key functions of SaaS service to build successful SaaS service from their service model. Second, we define our general SaaS maturity model and the detailed components of each phase of our model with two important axes. Finally, we propose the SaaS service architecture which contains the essential activities according to the common functions of SaaS service.

2 Related Work

It's a common experience among software companies today with increasing cost pressure, high customer expectation, and global competition all placing new demands on development and delivery processes. SaaS is a software delivery model where instead of purchasing the software and implementing, users can rent the software on a monthly cost-per-user or usage basis and can scale up or down as needed [1]. There are some representative SaaS vendors including Microsoft, Google, Salesforce.com, and Amazon which provide successful service cases. They have different strategies to catch up the paradigm shift toward SaaS respectively.

However, nowadays, it seems that software vendors have reached a consensus that SaaS model can be achieved in an incremental way. Several research groups have reported characteristics and maturity model for SaaS service platform. Microsoft characterizes configurability, multi-tenancy, scalability as key criteria of SaaS maturity model [2]. Forrester group also presented 6 levels of maturity model from traditional ASP to SaaS service model [3]. In addition, SaaS has been intensively studied with the academic point of view. A number of academic researchers have analyzed principal aspects of SaaS model, and also, have reported several features and advantages with both customer's and enterprise's perspectives [4-6]. Actually, SaaS is a complex business model which has many role players such as customer, developer, vendors, and so on. Therefore, some researches argued processes and interactions between them [7-8], and also, some researchers introduced lesson learned of implementing SaaS [9-11]. There also have been discussions about the maturity of SaaS model [12].

The rest of this paper is organized as follow. In section 3, we examine current SaaS service cases and identify common features of SaaS service according to the research methodology to find out key enablers for SaaS service model. In section 4, current SaaS maturity models are introduced and compared with our maturity model. A detail activities and architecture based on the maturity model is presented in section 5, and finally we draw our conclusion in section 6.

2.1 Research Methodology

Our research methodology consists of three steps - Deriving SaaS Service Functions, Defining SaaS Maturity Model, and Establishing SaaS Service Architecture.

Generally SaaS service consists of core functions which represent the properties of SaaS service. In this phase, we derived some common key functions of SaaS service through the survey of several cases of SaaS vendors. Moreover, we have analyzed the current maturity model for SaaS service such as Microsoft's and Forrester's model. Considering these approaches, we defined the important axis and our maturity model based on these axes according to the incremental development phase. The last step is establishing SaaS Service Architecture according to maturity model. In order to make practical methodology to build SaaS Service, we categorized the technical activities which constitute the role of each layer on the maturity levels.

3 An Analysis of SaaS Service

Since definition of SaaS is shown as various meanings according to researchers, we adapted the meaning of SaaS service through the Gartner Group as follows: *"software that is owned, delivered and managed remotely by one or more providers. The provider delivers an application based on a single set of common code and data definitions, which are consumed in a one-to-many model by all contracted customers, at anytime on a pay-for-use basis, or as a subscription based on usage metrics."* [8]. In other words, SaaS is a service-oriented framework with high deployment efficiency and supportable platform where ASP focused on architecture-oriented solutions with low deployment efficiency relatively. In this section, we discuss several cases of current large SaaS vendors which have their own characteristics of functionalities of SaaS service, and derive the essential common functions to build successful SaaS service.

3.1 Case Study: In Case of Current SaaS Service Vendors

Nowadays, the marketplace of SaaS service is mostly lead by international large software vendors and traditional solution provider for enterprise. We selected four major vendors as the successful cases of current SaaS service and surveyed the characteristics based on our conceptual SaaS service characteristics. Table 1 shows a summarized result of present condition of four SaaS service vendors.

3.1.1 Amazon: Service Infrastructure

Amazon provides SaaS service in terms of *Amazon Web Services.* It mainly focus on providing computing resources to users rather than a separated web-based application so that it gives customers various business application on their service infrastructure that is based on cloud computing paradigm. The goal of Amazon's SaaS service can be summarized by these terms: *Cost-effective*, *Dependable*, *Flexible*, and *Comprehensive*. In order to achieve the goals of Amazon Web Services, they settled various service types from business infrastructure to Web search and on-demand workforce. A distinct characteristic of Amazon Web Service is to give the opportunity to ISVs (Independent Software Provider), where the main targeted user of other vendors is end-users themselves who use the applications on the SaaS framework.

3.1.2 Salesforce.com: Platform as a Service

The SaaS service of Salesforce.com can be summarized as Force.com platform. It is a multi-tenant on-demand business platform which consists of service component and process. The biggest difference of the strategic directions of salesforce.com is to be a solution provider to enterprise with multi-tenant support platform. The whole service process of Force.com service aimed to achieving the next level of current SaaS, which is called PaaS (Platform as a Service). It means that PaaS should be able to provide the tools for developing on-demand applications easily on the Web-based infrastructure as well as using and distributing the solutions.

3.1.3 Microsoft: Software+Service

The main target users of Microsoft are customers who have used Microsoft's package software such as *Windows* and *Microsoft Office*. They try to add the service strategy based on the web to existing software in comparison with the other vendors who provide their service through network by using Web browser. This strategy is called *Software+Service*. The strategic directions of Microsoft are categorized into four parts: *Unified Experience*, *Server and Cloud*, *Tightly Coupled System*, and *Multiple Business Model*. By adapting these strategies, Microsoft tries to get flexibility and availability on service process from building service with software to software distribution.

3.1.4 Google: Service on the Web

Google provide SaaS service as the set of Google application named *GoogleApps*. It provides communicate and connect service through Web browser, and they are interlinked by collaboration process of Google Application such as *Google Docs* and *Google Sites*. In order to use their infrastructure and ability to search on the Web, Google tries to organize user's service via Web application development environment named Google Apps Engine. Most of Google SaaS service is supposed in the form of distributed APIs to guarantee effectiveness, flexibility, and easiness of application usage.

Table 1. Summarization of SaaS Service Vendors

Vendor	Service Description	Business Model	Origin	Strategy
Amazon	Computing Resource Providing	Amazon Web Services	Web Service	Service Infrastructure
Salesforce	Web-based CRM	Force.com	Web service / CRM	Platform as a Service
Microsoft	Personal / Office Tools	Microsoft Office Live	Package Software	Software +Service
Google	Web Office Tools	Google Apps	Web-based Service	Service on the Web

3.2 Common Functions of SaaS Service

By surveying representative cases of current SaaS service, the common functions are established which could be used for building standardized SaaS platform. We classified each function into technical and business functions as follows.

Table 2. The Common Functions of SaaS Service

Technical Function	
Multi-tenant Support	Shared Database and Service
	Predefined Database Extension
	Distributed Database Schema
Configuration	User Interface
	Workflow and Business Rule
	Customizable Data Model
	Metadata Set
Scalability	Scaling The Application
	Scaling The Data
Standard Support	Standard Business Data Model
	Business Standard Platform
	Standard Development API Set
Integration	Mash-up API
	Web Service
	Service Connector
	Multi-Platform Support
Security	Authentication
	Authorization
	Security Proof
	Tailored Security Policy

Business Function	
Market	Open Marketplace
	Application Selling Business Model
	SLA Adaptation and Support
	Service Billing Policy
Scalability	Guaranteed Performance
	Monitoring Tools for Availability and Performance
Development	Development Toolkit Providing
	Application Release Process
	Integrated Development Platform
Communica-tion	Supporting Community for Users
	Partnership Policy

3.2.1 Technical Functions

It means a set of functions which are related to technical issues of SaaS service such as database management, configurable user interface and business workflow, and integration technology. We expand the concept of the feature in their maturity level with *Standard Support*, *Integration*, and *Security* based on the features of Microsoft's maturity model such as *Multi-Tenant Support*, *Configuration*, and *Scalability*.

3.2.2 Business Functions

Business function is another important issue because software vendors should provide proper business model to customers and guarantee the continuity of business activity on their service platform with performance. We divide the business functions into four categories such as *Market*, *Scalability*, *Development*, and *Communication* area. Each category also contains list of detailed functions, which can support availability of core business competency.

4 SaaS Maturity Model

The strategic positioning of current SaaS services are promoted in different ways according to platform openness to partnership and their original service type such as packaged software or Web-based service. Nevertheless, current maturity models have not been able to present concrete procedures due to ambiguous definition of maturity levels. In this section, we introduce two maturity models from Microsoft and Forrester research group as examples of current maturity model for SaaS service.

Microsoft's maturity model is a sort of incremental development model through integration between functional features of SaaS. There are four level of maturity level such as *Ad hoc/custom*, *Configurable*, *Multi-Tenant*, and *Scalable*. Ad hoc level represents traditional ASP model with customizing to each individual users. At configurable level, users do not need to modify the application in the code level where multi-tenant architecture and tenant load balancer support the adaptation of various customers' requirements. Despite of the well-defined incremental structure, it is still ambiguous to measure ASP vendors' availability due to the lack of detailed concepts.

Forrester's model is similar to Microsoft's maturity model, but it contains six degrees of incremental development. At level 0 and 1, it can be mapped into current ASP players who handle their business manually with single or similar application to multi clients. At level 2 and 3, it can be called SaaS service because most of the vendors provide configured solutions with multi-tenant environment via packaged or Web-based distributed application. Level 5 and 6 contains custom extensions and dynamic composition so that service provider can compose user-specific applications with custom extensions in a multi-tenant environment. However, it also does not suggest the detailed incremental process of evaluation.

4.1 The Axis of Maturity Model

We decide the axis of maturity model as the core criteria for measuring the degrees of evaluation; *service component axis* and *maturity level axis*. We defined the service component as the core features of structuring software business, where each component corresponds to higher or lower layers of components. Moreover, we categorized our maturity levels which stand for the current situation of availability for ensuring SaaS service into four levels. In general, each level represents the fundamental foundation which is needed to evolve to next level in the model. Fig. 1 shows the summary of our service component and maturity level axis.

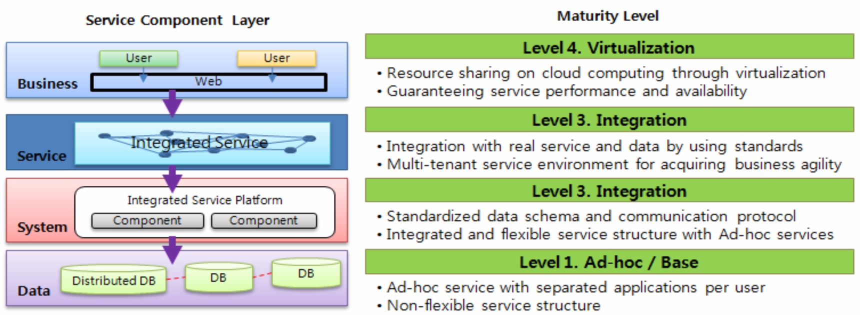

Service Component Layer
Business
User
User
Web
Service
Integrated Service
System
Integrated Service Platform
Component
Component
Data
Distributed DB
DB
DB
Maturity Level
Level 4. Virtualization
• Resource sharing on cloud computing through virtualization
• Guaranteeing service performance and availability
Level 3. Integration
• Integration with real service and data by using standards
• Multi-tenant service environment for acquiring business agility
Level 3. Integration
• Standardized data schema and communication protocol
• Integrated and flexible service structure with Ad-hoc services
Level 1. Ad-hoc / Base
• Ad-hoc service with separated applications per user
• Non-flexible service structure

Fig. 1. The two axes of maturity model: Service Component and Maturity Level

4.2 Maturity Model

Combining with two axes we discussed above, we build a general SaaS maturity model in Fig. 2 as a set of keywords that represent the core requirements. In detail, the components on the maturity model have technical activities which consist of the structure of service component. The following sections describe the concrete figures which are mainly discussed as essential component of our SaaS maturity model.

4.2.1 Ad Hoc Level

Ad hoc level is similar to current Ad hoc system with simple ASP business model. In this level, most of ASP players are concentrating on dedicated database and schema in data layer and schema without respect to content sharing and multi-tenant environment. System layer also consist of Ad hoc multi instances which use different applications by users. The service layer has separated system integration on Web interface mainly, where the detailed functions of the services are materialized by various requirements of customers. Finally, the business process such as SLA on Ad hoc level mostly depends on the simple contraction that reflects the necessity of separated users without any concrete policies. Many service providers have failed to find the proper ways to build SaaS service model and still have stayed in this level.

4.2.2 Standardization Level

Standardization level aims to provide shared service with the discrete instances of user's application and configurability. We define other features of data and business layer as sharing contents and standardized service policy. In this phase, customers use shared and publicized database with dedicated data schema. System layer support configurable single instance and single tenant so that users build their service model within predefined instance that is given by service provider. With respect to customize applications, service layer also has configurable options in service software. In addition, the standardized methods are needed in the last business level of our maturity model. Despite of the conspicuous features which show the typical characteristics of SaaS service, it still does not support multi-tenant environment, one of the most important issues of SaaS platform.

4.2.3 Integration Level

The third level of our model is integration. It contains the entire feature of former two levels while it focuses on actualizing multi-tenant environment. In other words, database schema is shared as well as database itself in the data layer in order to accommodate multi tenant simultaneously. In the system layer, multi tenant supporting platform is provided with single instance. It means that user's systems should use predefined common instance with simple configurable options where the various user functions are achieved by service combination such as Web service and mash-up. Lastly, business layer focuses on realizing measurable SLA adaptation with standardized scheme. In general, most of current SaaS vendors concentrate their ability to build multi tenant service process with service connection as the key factors of successful SaaS service we described above.

4.2.4 Virtualization Level

We believe that the ideal approach to reach real SaaS service is virtualization. For achieving virtualization on the data layer, entire database and its schema should be constructed upon distributed computing power such as cloud computing. It aims to realize optimized multi-tenant environment through well-defined set of metadata. In the system layer, the system space is transformed into virtual concept with load-balancing system. With the measurement of a quantity of service used, service providers can allocate the computing power dynamically to the users' systems on the virtual space. Similar to integration level, a set of function is given by interlinked service combination whereas business process covers the requirements of users instead of customization in code levels. It is generally considered that the service architecture on full-SOA can achieve the goals of service layer on virtualization level. Finally, business layer on this level is represented by optimized SLA adaptation. Service provider can use flexible and dynamic methods for measuring the amount of service used so that they ensure the optimization of SLA policy to their customers. In conclusion, virtualization level mainly focuses on maximization of practical use of resources via service modulation and encapsulation.

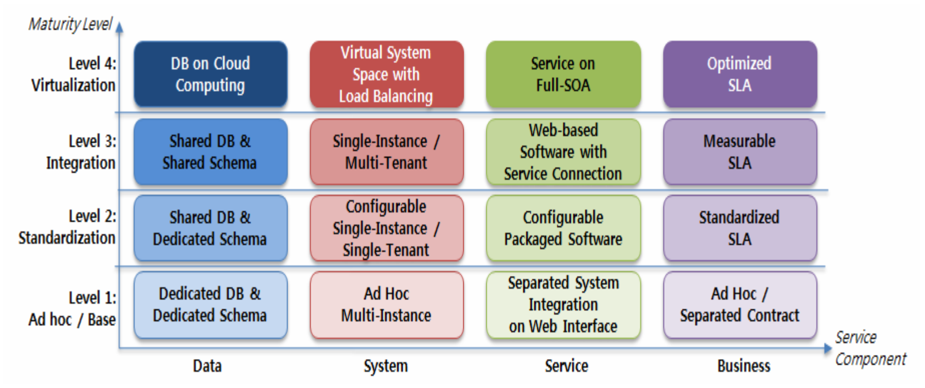

Maturity Level
Level 4: Virtualization
Level 3: Integration
Level 2: Standardization
Level 1: Ad hoc / Base
DB on Cloud Computing
Virtual System Space with Load Balancing
Service on Full-SOA
Optimized SLA
Shared DB & Shared Schema
Single-Instance / Multi-Tenant
Web-based Software with Service Connection
Measurable SLA
Shared DB & Dedicated Schema
Configurable Single-Instance / Single-Tenant
Configurable Packaged Software
Standardized SLA
Dedicated DB & Dedicated Schema
Ad Hoc Multi-Instance
Separated System Integration on Web Interface
Ad Hoc / Separated Contract
Data
System
Service
Business
Service Component

Fig. 2. General SaaS Maturity Model

5 SaaS Service Architecture

Considering the gap between the ideal figure of SaaS service and general business model of ASP players, referenceable SaaS service architecture is able to help service providers decide the service model and its development process in practical ways. We will present our SaaS service architecture roughly in terms of general service process and the major activities of SaaS service.

5.1 Major Activities of Maturity Model

Assuming the role of SaaS maturity model, the abstract definition of each level is converted into technical activities with hierarchical structure which constitute the structure of service architecture. As the same ways, second level activities can be described with lower level activities. Table 3 summarizes a part of the major activities on our maturity model.

Table 3. A Part of Major Activities on SaaS Maturity Model

	Data	System
Virtualization	Metadata Management Data Structure Optimization	System Provisioning Resource Optimization
Integration	Data Structure Management Data Model Management	Resource Management Resource Measurement
Standardization	Database Performance Management Database Security Management	System Composition Management
	Service	**Business**
Virtualization	Service Optimization On-Demand based Business Process	SLA KPI Optimization
Integration	Standard Service Distribution Service Reuse Measurement	SLA KPI Measurement
Standardization	Software Quality Management User Requirement Management	Standard SLA template production

5.2 Reference Architecture

Based on the service activities described above, we design SaaS service architecture and its service process. Our service architecture is a middleware between service users and service vendors, where the SaaS service platform in the architecture consist of three parts such as data, system, and service. Each part of service platform guarantees the core competencies of SaaS service. Fig. 3 describes a big picture of SaaS service architecture with the flow of business process.

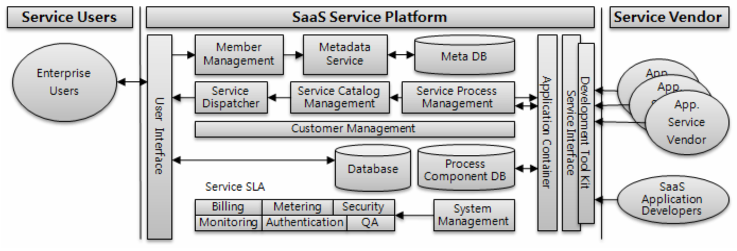


Fig. 3. SaaS service architecture

6 Conclusion and Future Work

This paper reviewed the SaaS service model and offered a practical SaaS maturity model and the service architecture with core functionalities of SaaS service. From our analysis described in previous sections, we concluded that the challenging issues of migrating current ASP service to SaaS can be solved incrementally by adapting the concept of our maturity model with two important axes.

Our maturity model only considers 2-dimension axes such as service component and maturity level. We believe that expanding the maturity model with n-dimension axes according to the organizations or characteristics of enterprise can contribute for designing future SaaS platform. In addition, we also aim at implementing real SaaS service platform with our maturity model and architecture with service process.

It is essential for enterprises to provide a strong and steady vision of SaaS service. However, existing approaches are still too general to design the detailed business process. We believe that the maturity model should give a concrete way of constructing a strategy to build practical SaaS service. Based on this research, we will begin organizing the nature of SaaS service and refining our maturity model with core competencies and more detailed requirements considering the current situation of industry as a whole. We also expect that our research will contribute to spread the concept of SaaS service within the service architecture as well as unique strategies within some practical semantic technology in the near future.

Acknowledgments. This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency). (grant number NIPA-2009-C1090-0902-0031).

References

1. Upadhyay, S.: Software as a Service (SaaS). Oracle (2008)
2. Chong, F., Carraro, G.: Architecture Strategies for Catching the Long Tail, MSDN Library. Microsoft Corporation (2006)
3. Ried, S., Rymer, J.R., Iqbal, R.: Forrester's SaaS Maturity Model. Forrester Research (2008)
4. Ma, D.: The Business Model of Software-As-A-Service. In: IEEE International Conference on Services Computing (SCC 2007), pp. 701–702 (2007)
5. Waters, B.: Software as a service: A look at the customer benefits. Journal of Digital Asset Management 1(1), 32–39 (2005)
6. Xin, M., Levina, N.: Software-as-a-Service Model: Elaborating Client-Side Adoption Factors. In: 29th International Conference on Information Systems, Paris, France (2008)
7. Sääksjärvi, M., et al.: Evaluating the Software as a Service Business Model: From CPU Time-Sharing to Online Innovation Sharing. In: IADIS International Conference e-Society, Qawra, Malta, pp. 177–186 (2005)
8. Choudhary, V.: Software as a Service: Implications for Investment in Software Development. In: 40th Annual Hawaii International Conference on System Sciences, HICSS 2007 (2007)
9. Greschler, D., Mangan, T.: Networking lessons in delivering 'Software as a Service' - part I. International Journal of Network Management 12(5), 317–321 (2007)
10. Guo, C.J., et al.: A Framework for Native Multi-Tenancy Application Development and Management. In: 9th IEEE International Conference on E-Commerce Technology and (CEC 2007), Tokyo, Japan (2007)
11. Chou, D.C., et al.: Analysis of a new information systems outsourcing practice: software-as-a-service business model. International Journal of Information Systems and Change Management 2(4), 392–405 (2007)
12. Goth, G.: Software-as-a-Service: The Spark That Will Change Software Engineering? IEEE Distributed Systems Online 9(7), Art.No. 0807-o7003 (2008)

Lazy View Maintenance for Social Networking Applications

Keita Mikami, Shinji Morishita, and Makoto Onizuka

NTT CyberSpace Laboratories, NTT Corporation, 1-1 Hikari-no-oka Yokosuka, 239-0847 Japan
{mikami.keita,morishita.shinji,onizuka.makoto}@lab.ntt.co.jp

Abstract. We introduce CAMEL, a lazy view maintenance system for social networking applications on a database server with a distributed memory cache. System administrators can control the throughput of the system by tuning the level of freshness of materialized views. CAMEL employs the existing view maintenance techniques of incremental maintenance, lazy maintenance, and control table. In addition, CAMEL optimizes view maintenance performance by pushing the top-k operation down to before join operations and by constructing a reverse index. We evaluate CAMEL using real data from a mini-blog service. The results show that CAMEL is 6.13 and 11.2 times faster than the method of eager view maintenance while keeping the freshness of materialized views at 66.2% and 38.0%, respectively.

1 Introduction

Many social networking websites have been emerged and they are becoming very popular on the Internet, such as MySpace, Facebook, and Twitter. According to article [1], MySpace became the most popular social networking site in the United States in June 2006 and was the destination of 4.46% of US visits. Facebook is also a popular social networking website and there are more than 250 million active users in 2009 [2]. Twitter has 200 thousand active users per week [3]. Those social networking websites not only share some common features with web applications but also have additional features that are typical to social network applications.

First, we consider the common features of web applications: 1) high read throughput is required, and 2) durability is necessary. To obtain high read throughput, a distributed memory cache is widely used for web applications. A distributed memory cache is installed on multiple machines and they are placed between the application server and the database server. The distributed memory cache achieves high read throughput by caching frequently accessed data on the database server; applications can access the cached data without accessing the database server. Durability is achieved by sending write operations directly to the database server, at the same time cached data is invalidated by the write operations to ensure rigid consistency. However, if we consider the features of social networking applications, the above approach to distributed memory is not effective. The features of social networking applications are: 3) high write (append) throughput is required, and 4) eventual consistency is enough, rigid consistency is not required, since it is acceptable for users to read data with certain level of staleness. Indeed, according to Facebook statistics [2], more than 1 billion pieces of content (web links, news stories, blog posts, notes, photos, etc.) are shared each week. Speculation about Twitter [3] shows that there are 3 million new messages per day. Therefore, those systems must provide high write throughput. The high write throughput invalidates the cache so frequently that cache effectiveness decreases. Fig.1[1] shows a result of preliminary experiment that examined how cache hit rate is affected by the write/read ratio of query workload. Cache hit rate is decreased to around 60% when the ratio is 1 and this raises the number of accesses to database server by a factor four compared to the case when the cache hit rate is 90% (write/read ratio is 0.05).

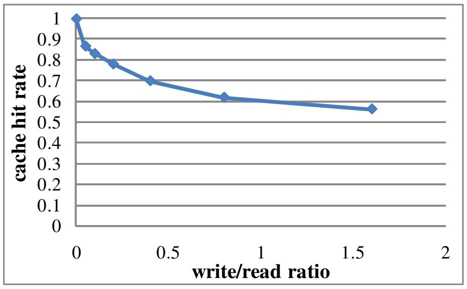


Fig. 1. Cache hit rate

Against this problem, we propose CAMEL, a view maintenance system that efficiently maintains materialized views stored in distributed memory cache, given that users accept certain level of staleness of views, therefore staleness of query results. There is a tradeoff between the staleness or freshness of materialized views and write/read throughput of the system; if users need more fresh data to be returned, the throughput decreases. Therefore, CAMEL provides system administrators a function to set the level of freshness of materialized views. We focus on social networking applications in which each user appends a new message and reads the latest k messages of his/her friends. This query is expressed with joining message table and friend table so as to locate all messages of the user's friends and then perform the top-k operation over the located messages sorted by publication date.

CAMEL consists of three major techniques as follows. First, as described above, it provides system administrators a function to set the level of freshness of materialized views, so that he/she can control the throughput of the system. As long as freshness is being satisfied, CAMEL lazily maintains views stored in distributed memory cache, so the the overhead of view maintenance is reduced by packing multiple maintenance tasks into one task. Second, we employ incremental view maintenance [4] for efficient cache maintenance, since the views for the above query are in the equivalent class of select-projection-join(-sort) views. There are two types of update operations that trigger the incremental view maintenance, 1) update of base tables of the views, such as an user appends a new message or a new friend, and 2) update of control table of views [5]. A control table is a table that controls which records of the views should be dynamically materialized. Cache replacement policies are used for the dynamic update of the

[1] Fig.1 used the mixed workload of message posting and *friend timeline* in Section 4.

control tables for keeping the cache hit rate high. According to [5], we could get 90% of the benefit of the materialized view by materializing only 0.5% of the rows and this reduces both the overhead of view maintenance and storage space. Third, CAMEL improves view maintenance performance by pushing the top-k operation down to before join operations and by constructing a reverse index. A reverse index is an index that improves the performance of view maintenance whose plan is executed in reverse join order of the view/query processing.

We evaluated CAMEL using real data from a mini-blog service. The results show that CAMEL is 6.13 and 11.2 times faster than the method of eager view maintenance while keeping the freshness values of materialized views at 66.2% and 38.0%, respectively.

2 Preliminary

Incremental view maintenance: Incremental maintenance for views in select-projection-join class is based on the differentiation step [4]. Consider the following example of a join operation among two tables. Let $\Delta\mathbf{R}, \Delta\mathbf{S}$ be inserted records for base tables $\mathbf{R}, \mathbf{S}$ and $\mathbf{R}', \mathbf{S}'$ be the after images of $\mathbf{R}, \mathbf{S}$ to which $\Delta\mathbf{R}$, $\Delta\mathbf{S}$ have been applied, respectively. The following equation can be obtained by applying the differentiation step:

$$\begin{aligned}
\mathbf{R}' \bowtie \mathbf{S}' &= \mathbf{R} \bowtie \mathbf{S} \cup \mathbf{R} \bowtie \Delta\mathbf{S} \cup \Delta\mathbf{R} \bowtie \mathbf{S} \cup \Delta\mathbf{R} \bowtie \Delta\mathbf{S} && (1)\\
&= \mathbf{R} \bowtie \mathbf{S} \cup \mathbf{R}' \bowtie \Delta\mathbf{S} \cup \Delta\mathbf{R} \bowtie \mathbf{S}' - \Delta\mathbf{R} \bowtie \Delta\mathbf{S} && (2)\\
&\neq \mathbf{R} \bowtie \mathbf{S} \cup \mathbf{R}' \bowtie \Delta\mathbf{S} \cup \Delta\mathbf{R} \bowtie \mathbf{S}' \cup \Delta\mathbf{R} \bowtie \Delta\mathbf{S} && (3)
\end{aligned}$$

Since the first term, $\mathbf{R} \bowtie \mathbf{S}$, in Equation (1) or (2) is assumed to be materialized already, the reminder part of the equation needs to be processed for incremental view maintenance. Except for the first term, notice that Equation (1) refers only to the before images of the tables, whereas (2) refers only to the after images of the tables. We will return to these equations in Section 3.3.

State bug problem: Either equation is applicable to eager view maintenance, however Equation (1) is not applicable to lazy view maintenance. Since views are maintained lazily, update operations are made to the base tables, so the before images of the two tables, $\mathbf{R}, \mathbf{S}$, do not exist anymore. The state bug problem [6], which produces an incorrect result, is caused by applying the equation of before images to the after images of tables as Equation (3) shows.

3 CAMEL

3.1 Architecture

Fig.2 depicts a system architecture on which CAMEL is implemented. It consists of an application server, a distributed memory cache running on multiple machines, and a database server. The view maintenance algorithm, CAMEL, is

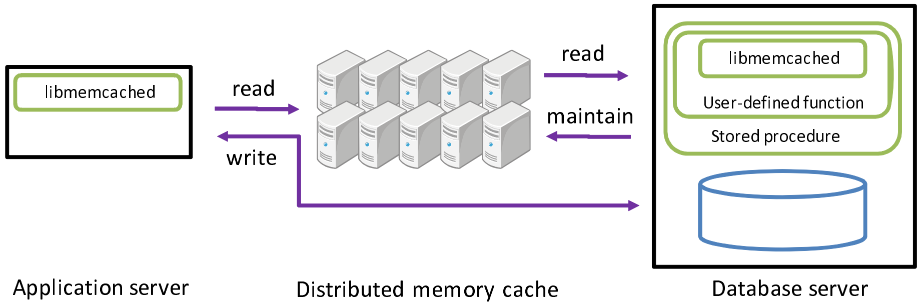


Fig. 2. System Architecture

implemented by stored procedures with user-defined functions working on top of the database server.

A query is processed as follows. It is submitted from the application server to a certain partition of the distributed memory cache that may store the corresponding materialized view of the query. The partition is located by hashing the record key used in the WHERE clause in the query. If the materialized view is stored there, the query is processed without accessing the database server. Otherwise, the query is sent to the database server and the result is returned and is copied in the memory cache. The materialized view is lazily and incrementally maintained for the update of the base tables or control table (See 3.3 for details).

3.2 Application

In typical social networking applications, each user uploads messages and shares them with his/her friends. *friend timeline* denotes the user's view that lists the latest k (10 or 20 in general) messages of his/her friends. *friend timeline* is usually shown in the top page of every user in social networking applications, so it requires high throughput and fast response time. In terms of schema design, there need two tables, 1) **message(<u>id</u>, user, date, body)** table to store users' messages, and 2) **friend(<u>user, friend</u>)** table to store friend relation of users: **friend** is a friend of **user**'s. The *friend timeline* of user u is expressed in relational algebra as follows:

$$timeline(u) = \mathrm{topk}_{date}(\pi_{friend}(\sigma_{user=u}(\mathbf{friend})) \bowtie_{friend=user} \mathbf{message}) \quad (4)$$

where $\mathrm{topk}_{atr}(R)$ is an unary operation and the result is defined as first k records sorted in descending order by attribute atr in relation R. For efficient processing of the above query, we materialize the result of the above view (query) only for users that are in relation **hotspot(<u>hotuser</u>)**, which is a control table. The idea of the control table [5] is that we can reduce the overhead of view maintenance and the storage space by materializing only frequent view records referred from queries. The view is defined by letting user u in the above query be any record in **hotspot** as follows:

$$\forall u \in \mathbf{hotspot}.timeline(u). \quad (5)$$

The distributed memory cache partitions the materialized view by the partition key obtained by hashing the record key in the view. In the above view definition, **hotuser** attribute of **hotspot** table is used as the input to the hash function.

The features of social networking applications are 1) users frequently append messages, 2) most users have around 100 friends on average[2], and 3) users accept a certain level of data staleness; we assume that they do not care so much if the delay in updating their friends' messages is around several minutes, because social networking applications are not real-time applications.

3.3 Technical Advantages

Incremental view maintenance: Since the top-k operation can be interpreted as a combination of sort and selection operations, the view defined above is in the class of select-projection-join(-sort) views. Thus, it can be incrementally maintained by existing techniques [4], which is more efficient than being maintained from scratch. There are two types of update operations that trigger incremental view maintenance, 1) update of base tables of views, such as a user appending a new message or a new friend, and 2) update of control table of views. We explain how CAMEL incrementally maintains views with reference to Alg. 1, which is the pseudo-code of incremental view maintenance of CAMEL.

First, we investigate the maintenance of base tables. As an example, the incremental maintenance of 100 message appends is dramatically more efficient than the full maintenance of the message table which may contain one million records (See the real data size in Section 4). Let **Δmessage** be the appended messages. The friend timeline view is incrementally maintained according to the following equation obtained by applying the differentiation step [4]:

$$\begin{aligned}\mathrm{topk}_{date}(\pi_{friend}(\sigma_{user=u}(\mathbf{friend})) \bowtie_{friend=user} \mathbf{message} \cup \mathbf{\Delta message}) = \\ \mathrm{topk}_{date}(\mathrm{topk}_{date}(\pi_{friend}(\sigma_{user=u}(\mathbf{friend})) \bowtie_{friend=user} \mathbf{message}) \cup \\ \mathrm{topk}_{date}(\pi_{friend}(\sigma_{user=u}(\mathbf{friend})) \bowtie_{friend=user} \mathbf{\Delta message})) \quad (6)\end{aligned}$$

Note the out-most topk_{date} operation on the right hand in Equation (6) is required, since its parameter is a union of two ordered sets. When u is not in the control table **hotuser**, the corresponding view of u is not being materialized, so there is no need of view maintenance. Otherwise, $\mathrm{topk}_{date}(\pi_{friend}(\sigma_{user=u}(\mathbf{friend})) \bowtie_{friend=user}$ **message**) is being materialized, so CAMEL maintains the view. Logically, CAMEL has to incrementally maintain friend timeline view for each user in **hotuser**, as described in Equation (5). However, the optimum plan of incremental view maintenance is not always the same plan with view processing. CAMEL chooses an optimum execution plan as follows. According to Equation (5), which corresponds to joining **hotspot** with *timeline*(u) that includes joining **friend** and **message**, CAMEL has to join three tables, **hotspot**, **friend**, ***Δ*message** and the order of joining those tables strongly impacts its performance. CAMEL relies on the query optimizer of the database server to choose the optimum

[2] 80 friends in Twitter [7] and about 150 social relationships in real world [8].

Algorithm 1. Incremental View Maintenance

```
Require: updated records $\Delta R$, target table $R$
Ensure: updated target table $R'$ and distributed memory cache
 1: if ($R$ is message table) then
 2:    $R' = R \cup \Delta R$;
 3:    submit query in Fig.3;
 4:    for all value $u \in$ hotspot.hotuser do
 5:       locate partition in the query result by $u$;
 6:       send the partition to the corresponding memory cache located by hash($u$);
 7:       append the partition to the cached data specified by $u$, and then apply topk$_{date}$
          operation to the cached data;
 8:    end for
 9: else if ($R$ is friend table) then
10:    {Do incremental view maintenance in similar way when $R$ is message table}
11: else {$R$ is control table}
12:    if $\Delta R$ is a type of insert then
13:       $R' = R \cup \Delta R$;
14:       for all $u \in \Delta R$ do
15:          submit query of $timeline(u)$ defined in Equation (4);
16:          send the query result to the corresponding memory cache located by
             hash($u$);
17:          copy the query result as cached data of $u$;
18:       end for
19:    else {$\Delta R$ is a type of eviction}
20:       {Only this part is triggered from the distributed memory cache when $\Delta R$ has
          already been evicted from the cache}
21:       $R' = R$ - $\Delta R$;
22:    end if
23: end if
```

plan by submitting an SQL query, see Fig3. The view maintenance works as follows (lines 3-7 in Alg.1): 1) submit query in Fig.3 and obtain the query result partitioned by each value of **hotspot**.**hotuser**, 2) send each partition to the corresponding memory cache located by hashing the corresponding value of **hotspot**.**hotuser**, and 3) at the memory cache, append the specified partition to the cached data located by **hotspot**.**hotuser**, and then apply topk$_{date}$ operation to the cached data. In addition, the third step is simplified to append the query result to the head of the cached data without topk$_{date}$ operation, since the **date** of any record in **Δmessage** is newer than that stored in the memory cache.

```
SELECT hotspot.hotuser,Δmessage.*
FROM   hotspot, friend, Δmessage
WHERE  hotspot.hotuser=friend.user
AND    friend.friend=Δmessage.user
GROUP BY hotspot.hotuser;
```

Fig. 3. View maintenance query

Second, the materialized view is maintained by not only the update of the base tables, but also the update of the control table, **hotspot**, that are updated according to cache replacement policies. Notice that record eviction is triggered by the distributed memory cache, whereas record insertion to the control table is triggered by the database server. When a query of *timeline*(u) is processed, CAMEL inserts u into **hotspot** table and the query result is copied in the distributed memory cache (lines 14-18 in Alg.1). For record eviction, CAMEL relies on a cache replacement policy of distributed memory cache. In CAMEL, the distributed memory cache notifies the evicted records to the database server, so that CAMEL can determine which update of base tables may affect the materialized views (lines 20-21 in Alg.1).

Lazy view maintenance: We introduce a function to configure the level of freshness of materialized views, so that system administrators can control the throughput of the system. By lazily maintaining views as long as freshness is being satisfied, CAMEL reduces the overhead of view maintenance by packing multiple maintenance tasks (for example, 100 message appends) into one task. This reduces the number of database accesses needed for maintenance, the number of communications between the database server and the distributed memory cache, and the number of view maintenance operations on the distributed memory cache. However, lazy view maintenance approaches can suffer from the state bug problem as described in Section 2. In general, lazy view maintenance must use Equation (2) to avoid triggering the state bug. CAMEL creates for each updatable table a differential table [9,6] storing updated records of base tables that are not propagated to the materialized views. CAMEL periodically propagates the updated records stored in the differential tables as indicated by either time constraint or space constraint of the differential tables.

Optimization: CAMEL introduces two optimizations: pushing down the top-k operation and creating reverse index.

The inefficiency of the query processing of *timeline*(u) is due to the fact that it must join all the messages of the friends of user u, although most of them are not needed in the query result. CAMEL optimizes this process by pushing the top-k operation down to inside the join operation according to the following equation:

$$\begin{aligned}\mathrm{topk}_{date}(\pi_{friend}(\sigma_{user=u}(\mathbf{friend})) \bowtie_{friend=user} \mathbf{message}) =\\ \mathrm{topk}_{date}(\bigcup_{f\in\pi_{friend}(\sigma_{user=u}(\mathbf{friend}))} \mathrm{topk}_{date}(\sigma_{f=user}(\mathbf{message})))\end{aligned} \qquad (7)$$

Intuitively, Equation (7) indicates that the top-k messages of all friends of user u are obtained by 1) extract top-k messages of each friend of user u, 2) make union of the extracted messages of all friends, and 3) extract top-k messages of the obtained messages. It is also possible to omit the outer top-k operation by using sorted-merge join, since it keeps the order of the records.

Remember that the optimum plan of incremental view maintenance is not always the same as the plan for view/query processing. As an example, there are multiple plans for SQL query in Fig.3: 1st plan is joining **friend** and Δ**message** first, and then joining the result with **hotspot**, and 2nd plan is joining three tables in reverse order, joining **hotspot** and **friend** first, and then joining the result with Δ**message**. The number of records in **hotspot** or Δ**message** is assumed to be less than that in **friend**. Then, for the 1st plan, an index should be constructed on **friend.friend**, whereas for the 2nd plan, an index should be constructed on **friend.user**. Not only do we build the latter index, which is already built for view/query processing, but also the former index, namely reverse index, to improve view maintenance performance.

3.4 Handling Failure of Distributed Memory Cache

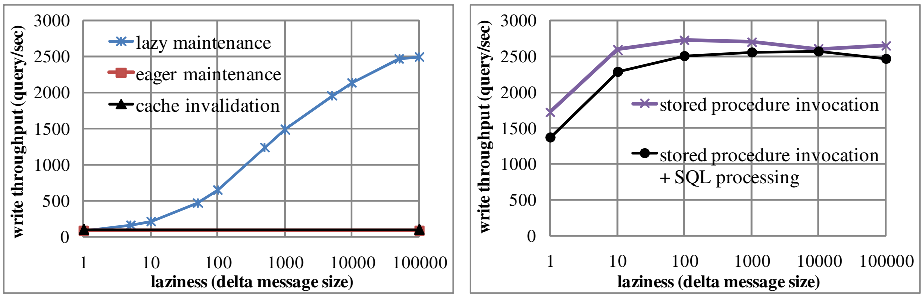
Some machine that consists the distributed memory cache may fail due to system failure or power supply failure. In such case, an inconsistency occurs between control tables and cached data in the distributed memory cache: although the control tables store records, there are no corresponding materialized views in the distributed memory cache. There are two approaches to fix this inconsistency: to immediately detect the restart of the partition and re-build the corresponding materialized views, or to detect the inconsistency afterward when CAMEL maintains the materialized views and finds the cache lost.

4 Experiments

We made experiments to evaluate the efficiency of lazy view maintenance and tradeoff between system throughput and freshness of views. We omit the use of the control table in the experiments, since its effectiveness has been validated in [5]. This simplifies query and view maintenance processing, however the techniques described in Section 3 still have a valid impact on view maintenance.

Setting: Our execution environment consisted of a single database server (MySQL 5.0) on Xeon X5355(2.66GHz) with 16GB RAM, ten sets of distributed memory cache servers (memcached 1.2.6) on Celeron 430(1.8GHz) with 2GB RAM, and a single application server (memcached client, libmemcached 0.23 running on ruby 1.8.5) on Xeon X5470(3.33GHz) with 32GB RAM. Linux 2.6.18 was run on all 12 computers. Each distributed memory cache server was configured to have 1GB cache pool.

Data and Workload: We used real data from a mini-blog service, Goo home (http://home.goo.ne.jp), posted from May to November 2008 with the schema definition described in Section 3.2. The number of records of **friend**, **message** were 70.8K(thousand), 1128K, respectively. We stored all records in **friend** table and first 1000K records in **message** table. They were MyISAM tables. We constructed a message posting workload by using the reminding 128K message records. We used different sizes of Δ**message** from 1 to 100K records, to validate the effectiveness of lazy view maintenance. For another workload of *friend timeline* processing, we generated a synthetic workload by assuming that a user who posts more messages or has more friends reads the friend timeline view more frequently according to a Zipf distribution.



(a) Comparisons of three algorithms (b) Micro-benchmark for lazy maintenance

Fig. 4. Performance of Lazy view maintenance

Results: (a) Fig.4 shows the system throughputs of message posting workload by comparing three algorithms, cache invalidation, eager view maintenance, and lazy view maintenance. Cache invalidation, which is a typical usage of distributed memory cache, invalidates the records of the view that are affected by the update of the base tables. Eager view maintenance maintains materialized views synchronized with every record update in the base tables, whereas lazy view maintenance asynchronously maintains materialized views for every Δ**message**-size update. The X-axis value is the size of Δ**message** for lazy view maintenance. The throughputs of the other two methods are fixed against the X-axis, since both are triggered by each record update of **message** table, so their throughputs are independent of the size of Δ**message**. It is interesting to observe that cache invalidation is as expensive as eager view maintenance, because not only eager view maintenance, but cache invalidation also requires query processing of Fig.3 to locate the materialized records in the view affected by the update of the base tables. We also observe that lazy view maintenance improves the throughput of message posting workload. As the size of Δ**message** increases, the throughput is improved more, because the overhead of view maintenance is reduced by packing Δ**message**-size maintenance tasks into one task. For example, lazy view maintenance by CAMEL is 6.13 and 11.2 times faster than the other two when Δ**message** size is 100 and 1000, respectively.

(b) Fig.4, which uses the same scale as (a) Fig.4 in X- and Y-axis, shows the result of a micro-benchmark of lazy view maintenance. **stored procedure invocation** in (b) Fig.4 indicates the throughput of lazy view maintenance only with the invocation of stored procedures, i.e., without SQL processing and view maintenance on the distributed memory cache. **stored procedure invocation + SQL processing** indicates the throughput with the invocation of stored procedures plus SQL processing, but without view maintenance. Compared with the performance of full lazy view maintenance in (a) Fig.4, we found that view maintenance on the distributed memory cache is the major bottleneck in performance.

Finally, Fig.5 depicts how the freshness of views varies when laziness of views increases, while the write/read ratio of the mixed workload of message posting and *friend timeline* is fixed to 1. The freshness of views is defined as the ratio of the number of the queries with fresh (not stale) results to the number of all queries. As expected, freshness of views decreases as laziness of views increases. For example, freshness of views are 66.2% and 38.0% when *Δ***message** size is 100 and 1000, respectively. By taking the result in (a) Fig.4 into account, we conclude that lazy view maintenance by CAMEL is 6.13 and 11.2 times faster than the method of eager view maintenance while keeping the freshness of materialized views at 66.2% and 38.0%, respectively.

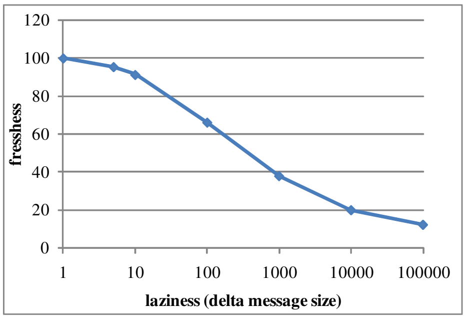


Fig. 5. Freshness of views

5 Related Work

There are a number of related studies on incremental maintenance for materialized views: [4] examines the view maintenance problem by categorizing maintenance techniques from three viewpoints: expressiveness of view definition language, amount of available information, and type of modification.

Control table is introduced in [5] to control which records of views should be dynamically materialized. Cache replacement policies (LRU, LRU-k, 2Q [10]) are used for the dynamic update policies of the control tables.

The state bug problem was pointed out by [6]. As we described in Section 2, it is caused by applying equations of before table images to the after table images, since only after images are available in case of lazy view maintenance. The approach illustrated in [9] solves this problem by keeping the base tables unchanged by update operations. Instead, it uses an optimized *hypothetical relation* [11] to store the updated records of base tables. One problem with this approach is that it slows down the processing of all queries over base tables. Deferred or asynchronous view maintenance was proposed in [6][12] but the focus was on minimizing view downtime. Both of them avoid the state bug problem by using compensation. Intuitively, compensation enables us to obtain the before image of base tables from the after image by applying base log in a reverse way, such as $\mathbf{R} = \mathbf{R}' - \Delta\mathbf{R}$. The lazy maintenance proposed in [13] achieves efficient maintenance while ensuring that queries see only up-to-date views. It avoids the state bug by introducing row versioning, so both before and after images are available. [14] also uses multi-versioning to reduce contention between view maintenance and read operations. A deferred view maintenance for data cloud [15] proposes two types of materialized views: remote and local view tables. Remote view table is separated from the base tables, whereas local view table stores view records on the same server as the corresponding base records. For incremental maintenance of remote views, [15] uses a log that contains both before and after values of the records.

6 Conclusion

We developed CAMEL, a lazy view maintenance system for social networking applications on a database server interacting with a distributed memory cache. We evaluated CAMEL using real data from a mini-blog service to show that CAMEL is 6.13 and 11.2 times faster than the method of eager view maintenance while keeping 66.2% and 38.0% of freshness of materialized views, respectively.

We expect that social networking applications will expand rapidly in size and type. One interesting application is nearmiss (http://twitter.com/nearmiss) on Twitter, which inputs the user's GPS data and notifies other users who are close to him/her geographically. This example leads to an open problem: social networking engines can act as a framework on which different applications with different priority levels are run. This feature is similar to the multi-tenant function of data cloud and so the social networking engine must schedule those different applications properly, if the system as a whole is to achieve high throughput.

References

1. Cashmore, P.: MySpace, America's Number One (2006), `http://mashable.com/2006/07/11/myspace-americas-number-one`
2. Facebook: Press Room (2009), `http://www.facebook.com/press/info.php?statistics`
3. Arrington, M.: End Of Speculation: The Real Twitter Usage Numbers (2008), `http://www.techcrunch.com/2008/04/29/end-of-speculation-the-real-twitter-usage-numbers/`
4. Gupta, A., Mumick, I.S.: Maintenance of materialized views: Problems, techniques and applications. IEEE Data Engineering Bulletin 18(2), 3–18 (1995)
5. Zhou, J., Larson, P.Å., Goldstein, J., Ding, L.: Dynamic materialized views. In: Proceedings of ICDE, pp. 526–535 (2007)
6. Colby, L.S., Griffin, T., Libkin, L., Mumick, I.S., Trickey, H.: Algorithms for deferred view maintenance. SIGMOD Rec. 25(2), 469–480 (1996)
7. Owyang, J.: Understanding HP Lab's Twitter Research research (2008), `http://www.web-strategist.com/blog/2008/12/08/understanding-hp-labs-twitter-research`
8. Howard, B.: Analyzing online social networks. Commun. ACM 51(11), 14–16 (2008)
9. Hanson, E.N.: A performance analysis of view materialization strategies. SIGMOD Rec. 16(3), 440–453 (1987)

10. Johnson, T., Shasha, D.: 2Q: A low overhead high performance buffer management replacement algorithm. In: Proceedings of VLDB, pp. 439–450 (1994)
11. Agrawal, R., Dewitt, D.J.: Updating Hypothetical Data Bases. Information Processang Letters 16, 145–146 (1983)
12. Salem, K., Beyer, K., Lindsay, B., Cochrane, R.: How to roll a join: asynchronous incremental view maintenance. SIGMOD Rec. 29(2), 129–140 (2000)
13. Zhou, J., Larson, P.A., Elmongui, H.G.: Lazy maintenance of materialized views. In: Proceedings of VLDB, pp. 231–242 (2007)
14. Quass, D., Widom, J.: On-line warehouse view maintenance. SIGMOD Rec. 26(2), 393–404 (1997)
15. Agrawal, P., Silberstein, A., Cooper, B.F., Srivastava, U., Ramakrishnan, R.: Asynchronous view maintenance for VLSD databases. In: Proceedings of SIGMOD, pp. 179–192. ACM, New York (2009)

Birds Bring Flues? Mining Frequent and High Weighted Cliques from Birds Migration Networks

MingJie Tang[1,3,*], Weihang Wang[1,3], Yexi Jiang[5], Yuanchun Zhou[1], Jinyan Li[4], Peng Cui[2,3], Ying Liu[3], and Baoping Yan[1]

[1] Computer Network Information Center, Chinese Academy of Sciences
[2] Institute of Zoology, Chinese Academy of Sciences
[3] Graduate University of Chinese Academy of Sciences
[4] School of Computer Engineering, Nanyang Technological University
[5] School of Computer Science, Sichuan University, Chengdu
{tangrock,supercat0325,yexijiang}@gmail.com,
zyc@.cnic.cn, jyli@ntu.edu.sg, cuipeng@ioz.ac.cn,
yingliu@gucas.ac.cn, ybp@cnic.cn

Abstract. Recent advances in satellite tracking technologies can provide huge amount of data for biologists to understand continuous long movement patterns of wild bird species. In particular, highly correlated habitat areas are of great biological interests. Biologists can use this information to strive potential ways for controlling highly pathogenic avian influenza. We convert these biological problems into graph mining problems. Traditional models for frequent graph mining assign each vertex label with equal weight. However, the weight difference between vertexes can make strong impact on decision making by biologists. In this paper, by considering different weights of individual vertex in the graph, we develop a new algorithm, Helen, which focuses on identifying cliques with high weights. We introduce "graph-weighted support framework" to reduce clique candidates, and then filter out the low weighted cliques. We evaluate our algorithm on real life birds' migration data sets, and show that graph mining can be very helpful for ecologists to discover unanticipated bird migration relationships.

Keywords: Graph Mining, Birds Migration, Birds Flues H5N1, Scientific data, Qinghai Lake.

1 Introduction

The Asian outbreak of highly pathogenic avian influenza H5N1disease in poultry in 2003, 2004 and 2009 was unprecedented in its geographical extent. Its transmission to human beings showed an ominous sign of life-threatening infection [1]. Research findings indicate that the domestic ducks in southern China played critic role in virus reproduction and maintenance [9]. The major question is arising to understand the

[*] This work was supported by Special Project of Informatization of Chinese Academy of Sciences in" the Eleventh Five-Year Plan", No.INFO-115-D02.

highly correlated species' habitat. It is critical for us to find the roots of the answers, such as: how the wild life and domestic poultry intersect together to translate virus to related places? How is the possibility of H5N1 that spilled over from the poultry sector into some wild bird species among habitats?

Clique is the most coherent and dense substructure among all kinds of subgraphs based on the assumption that there is at most one edge between any two vertices [3]. Discovering cliques from graph transaction database can provide insights about the underlying structure or relationships among different objects in graph transaction [5]. Meanwhile, researchers found that group of birds tended to move in ways that resembled weighted graphs, especially when a flock was active during their migration. For this matter, graph mining from birds' migration data does bring some aspiration for answering biological problem.

Traditional frequent cliques mining reflect information about the frequency of the presence or absence of a specified vertex label. However, in many cases it is possible that frequent cliques only contribute to a small portion of the overall "profit", whereas non-frequent cliques produce a large portion of the "profit". "Profit" could be defined as the object interest in different ways. In our study, biologists need to know active area of the birds by looking at the weights of the habitat combinations in order to consider the possibility of the bird interaction virus with poultry sector. The weight can be deemed as bird migration time on one habitat or density of bird satellite tracking location points. For instance, a clique, C_1, may be a frequent subgraph with frequency 60%, contributing 1% of the whole bird migration time. Another clique,C_2, may be a non-frequent clique with frequency 8%, contributing 20% of the whole migration time. Empirical experience from other ecological studies [10, 11] suggests that clique C_2 would be much interest to biologists to track the bird flues transferring.

In this work, this new approach called ***Helen*** (it stands for **H**igh w**E**ighted c**L**osed cliqu**E** mi**N**ing) was proposed to mine high weighted closed clique from graph transaction databases. We first introduce graph-weighted support framework which adopts "downward close closure" to reduce the clique candidate sets. And then we prune the overestimated weighted cliques. The main contributions of our work are summarized as follows: (1) Convert the biological research problem associated with bird flues translation way into graph mining problem, (2) Present new birds' migration clique mining mode to find highly correlated habitat areas. (3) Provide important and hided clues about the relationship of bird migration and H5N1 according to the results of our experiment.

The rest of this paper is organized as follows. Section 2 overviews the related work. Section 3 points out the desired results in which biologists interested and the practical challenges, then we introduce the preliminary concepts for discussion. Section 4 proposes the birds' migration clique mining model to find frequent and high weighted closed clique. Section 5 presents the experiment results and discuss the results for biological research ways. Finally, we summarize our works and point out the future research direction in the section 6.

2 Related Works

The applications of satellite tracking to bird migration studies have enabled considerable progress to be made with regard to elucidating the migration routes and stay sites of various migratory bird species, with important implications, for example, for conservation [11]. Our previous works [2] use clustering and association rules to discover bird migration habitats, site connectedness and migration routes. However, Biologists found that bird migration routes in small range of area usually are graph patterns rather than simple sequence.

In the previous published literature, we are unable to find any work on discovering weighted cliques from graph database, however, lots of works deal with mining the frequent clique and quasi-clique from multiple graphs. Pei et al. [5] proposed an algorithm called Crochet to mine cross quasi-cliques from a set of graphs. Later, Wang et al. [3] studied the problem of mining frequent closed cliques from graph databases. We adopt their clique enumeration and pruning idea to find frequent clique (in the section 4), and extend this by using the graph-weighted and rechecking to get high weighted closed cliques (see section 4.2). Later, Zeng et al. [4] studied a more general problem formulation, that is, mining frequent closed quasi-cliques from graph databases, and proposed an efficient algorithm called Cocain to solve the problem.

Meanwhile, an effort was done on assessing weighted association rules mining in the last decade using either the average weight value of the items comprising this itemset, or utilizing weighted framework to evaluate the weighed association rules [7]. Presumably the most relevant work to our current study was done by Liu [6], this paper proposed two phase algorithm to deal with "utility mining" problem. We use their two phase principle, but instead of using utility principle we use weighted-support framework which is easier for interpretation (see section 3.2)

3 Problem Formulation

3.1 Desired Results and Challenges

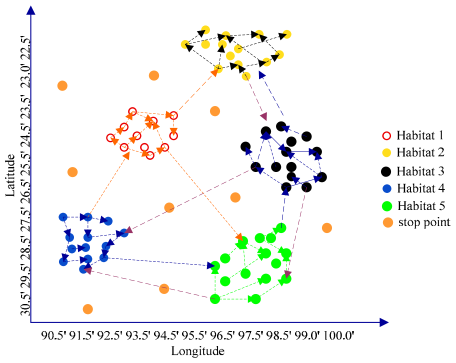
Supposed in the Figure 1(a), migration routes of birds can be regarded as one kind of graph, where habitat can be deemed as vertex nodes and the migration routes can be treated as edges. And birds' migration routes can be made of one graph database. Analyzing cliques from this graph database would give important knowledge about the possibility of birds spillover H5N1 among habitats in the same clique. There are two kinds of cliques that we hope to discover: frequent clique and high weighted clique.

Frequent Clique mining is such a process: given a graph transaction database D and a minimum support threshold min_sup, identify the complete set of cliques in D that are both frequent and closed. Several frequent cliques mining methods [3, 4, 5] could be deployed to solve frequent clique mining.

In some scenarios, high weighted cliques can provide useful information if we pay attention to the graph vertex weight. For example, in Figure 1 (a) and its related table Figure 1(b), several factors such as the number of migration points or the time spending on a particular habitat can be deemed as weight. If we dismiss this kind of information which would influence biologists to judge the possibility of birds transfer avian virus to domestic ducks or other birds [9], the mining results would not be "interest".

3.2 Problem Definition

We start with the introduction of a set of terms that leads to the formal definition of high weighted cliques mining problem. The same terms are given in [3,4,5].



(a) Birds Migration illustration

	Time (Day)	Location point
Habitat1	10	4000
Habitat2	30	6000
Habitat3	61	9000
Habitat4	21	4800
Habitat5	5	2000

(b) Weight of Migration Habitats

Fig. 1. Bird migration Space defined by Longitude and Latitude in the left. Point with different color and lines stand for tracked bird's migration coordinates and migration routes, separately. In the right table is birds' active information associate with their migration habitat.

Notations Description

- *V :* V = {v_1; v_2;…; v_k}, the set of vertices
- *E:* $E \subseteq V \times V$, the set of edges
- *L:* the set of vertex labels
- *F :* F :V⇒L, the mapping function from labels to vertices.
- *G:* G = (V;E; L; F), an undirected vertex-labeled graph
- |G| : |G| = |V|, the cardinality of G
- G(S): G(S): the induced subgraph on S from G, S $\subseteq$ V (G)

In this paper, we consider simple graph only, which does not contain self-loops, multi-edges, and edge labels. A clique is a fully connected graph and each pair of vertices in V there exist an edge in E. The size of a clique is defined by the number of vertices it contains, i.e.,|V|. A clique with n vertices is called an n-clique and the number of edges in the n-clique is n*(n-1)/2. For instance, given the graph database in Figure 2 and min_support=2, frequent 3-clique G_1 and 4-clique G_2 are illustrated at the right side. We use canonical code to present each clique, canonical code representation is defined as the minimum string among all its possible strings such as [3]. For example, Graph 3 in Figure 2(b) is a clique with 4 vertexes, the canonical form CF is represented as the string "ABDE", which is the smallest string with the combination of four letters "A", "B", "D" and "E". In the rest of paper, clique will be mentioned by their canonical form directly. Depend on the canonical form of clique; the sub-clique relationship can be changed to subsequence relationship. If clique C_1 and C_2 with canonical form CF_1 and CF_2 respectively, $C_1 \subseteq C_2$ iff CF_1 is a sub string of CF_2.

Frequent clique mining has been introduced in [3,4]. In the rest of paper, weights for the graphs vertex are considered, and high weighted clique mining are discussed mainly. Before discussion, some important definitions are given below.

DEFINITION 2.1 (Vertex weighted): The weight value w(L) means the significance for certain vertex label L. A graph is a set of weighted labels, each of which may appear in multiple graphs with same weight. For instance, in the Table 2, weight of the graph vertex in the Figure 2 are described: w(A)=7 , w(B)=6 ... □

DEFINITION 2.2 (Weighted graph): G_w = (V;E; L; F;W), an undirected vertex-labeled Weighted graph. The weighted graph is only considered with the weighted for the vertices label. □

DEFINITION 2.3 (Weight of graph): Weight (G^w) is the one graph weight. It could be sum up weight of all vertex labels in one graph simply.

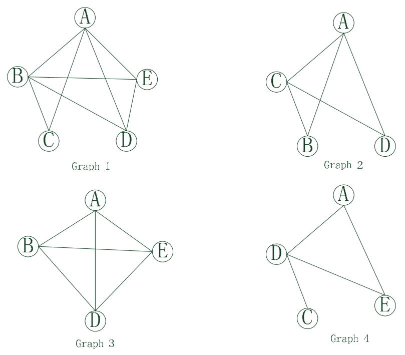
$$\text{Weight(G)} = \sum_{i=1}^{|V|} w_i \tag{1}$$

Where the |V| is the number of vertex label, the w_i is the vertex weight. For example, in the Figure 2 (a) and related weight Table 1:
Weight(G1)={w(A)+w(B)+w(C)+w(D)+w(E)}=(7+6+2+14+20)=49

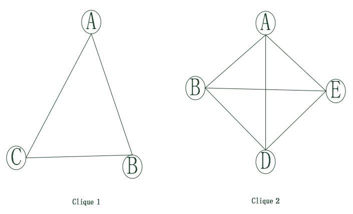
DEFINITION 2.4 (High Weighted Clique): The high weighted clique can be defined as the sum of the weight of the graph and the weight of the fraction of transactions that the graph occurs in. Thus, one clique is high weighted clique if:

$$\text{WSP}(C) = \frac{\sum_{C \subseteq G_i} Weight(C)}{\sum_{i=1}^{|D|} Weight(G_i)} > \varepsilon \tag{2}$$

Where the Weight (G) and Weight(C) is the weight of one graph defined in the DEFINITION 2.3. |D| is the number of graphs in one graph database. ε is viewed as user's interest. In addition, one clique C is a **High Weighted Closed Clique (HWCC):** If there does not exist another clique C' such that C $\subseteq$ C', WSP(C')= WSP(C) and WSP(C)> ε. □



(a) An example of graph database D



(b) Two frequent k-clique from D

Fig. 2. A graph database and parts of its sub graph

Table 1. The weight table for each vertical label in the Figure 2 (a)

Label	A	B	C	D	E
Weight	7	6	2	14	20

Problem Statement: Given one Graph database D and the related vertex weighted table WT, weighted clique mining is to find all high weighted closed clique. For example, in Figure 2 and Table 1, WSP(Clique 1)= WSP(ABC) = (15 × 2) / (49 + 29 + 47 + 43) = 0.17, WSP(Clique 2)=WSP(ABDE) = (47 × 2) / (49 + 29 + 47 + 43) = 0.55. If ε=0.5, clique(ABC) is a low weighted clique and should not be considered.

The graph database in the Figure 2(a), and graph database related weighted table (Table 1) will be our running examples in the rest of our paper**.**

4 Birds Migration Closed Clique Mining Model

We utilized a data mining framework to discover the frequent and high weighted cliques as in the Figure.3. A clustering algorithm in [2] is developed in this system to find sub-areas with a dense location points relative to the entire area. Then we adopted clique mining to discover the frequent and high weighted clique between the discovered habitats. Because of space limitation, details of frequent closed clique mining approach CLAN can be reached from paper [3].

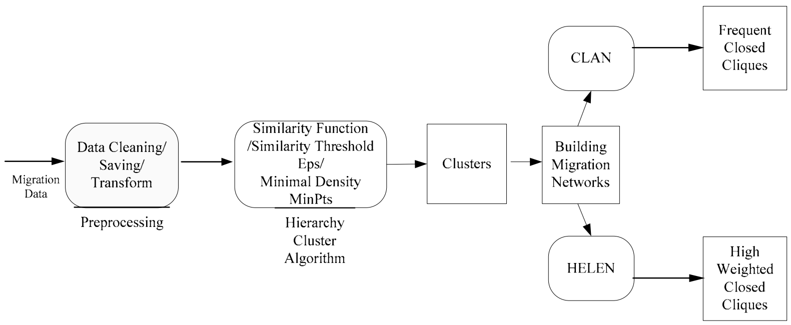

Migration Data
Data Cleaning/ Saving/ Transform
Preprocessing
Similarity Function /Similarity Threshold Eps/ Minimal Density MinPts
Hierarchy Cluster Algorithm
Clusters
Building Migration Networks
CLAN
Frequent Closed Cliques
HELEN
High Weighted Closed Cliques

Fig. 3. System Framework of Cliques Mining Toward Birds Migration

4.1 HELEN: High Weighted Closed Cliques Mining

Motivated by discovering high weighted cliques from birds' migration graph, we intend to extend traditional frequent graph mining method to meet our requirements. However, the "downward closure property" in Apriori-based approach cannot apply to the weighted clique mining directly due to weight bias support. Considering the challenges, “high graph-weighted” that owns downward closure property is deployed to reduce clique candidates. In the second place, rechecking is explored to filter out the high graph-weighted cliques that are indeed low high weighted clique. This approach was called HELEN.

4.1.1 High Graph-Weighted Support Framework to Prune Candidates

DEFINITION 4.1. Graph-Weighted Clique: The graph-weighted of a clique C, denoted as tw(C), is the sum of the graph weight of all the graph that C embedded in::

$$\mathrm{tw}(C) = \sum_{C \subseteq G_w \subseteq D} \mathrm{Weight}(G_w)$$

For example clique(ABC) in Figure 2, tw(clique ABC) =(W(G1)+W(G2))=(49+29)=78.

DEFINITION 4.2. High Graph-Weighted Clique: For a given clique C, C is a high graph-weighted supported clique if

$$GWSP(C)=\{\mathrm{tw(C)}/(\sum_{i=1}^{|D|} Weight(G_i))\} > \varepsilon',$$

Theorem 1. The graph-weighted Downward Closure Property indicates that when K-1 Clique is not high graph-weighted significant clique, the K clique can never be the high graph-weighted significant clique as well. In the process of enumerating clique, only high graph-weighted (K-1)-clique can be added as the candidates to extend K-clique.

Proof: Let *T(K)* be the collection of the graph transactions containing K-Clique and *T(K-1)* be the collection of transactions containing (K-1)-Clique. Although (K-1)Clique $\subseteq$ K-Clique, the graph support K-clique will decrease as the K increasing, thus the *T(K-1)* is a superset of *T(K).*

$$tw((K-1)clique) = \sum_{(k-1)clique \subseteq G_m} Weight(G_m) > tw((K)clique) = \sum_{(k)clique \subseteq G_n} Weight(G_n)$$

Then:

$$\{tw((K-1)clique) / \sum_{i=1}^{|D|} Weight(G_i))\} > \{tw((K)clique) / \sum_{i=1}^{|D|} Weight(G_i)\} > \varepsilon'$$

Theorem 2. Let HGWCC be the collection of all high graph-weighted closed cliques in a transaction database D, and HWCC be the collection of high weighted closed cliques in D. if $\varepsilon' = \varepsilon$, HWCC $\subseteq$ HGWCC

Proof

$\forall C \subseteq$ HWCC ,if C is a high weighted clique, and $C \subseteq$ HGWCC

$$\varepsilon' = \varepsilon \le WSP(C) = \frac{\sum_{C \subseteq G_i} Weight(C)}{\sum_{i=1}^{|D|} Weight(G_i)} < \frac{\sum_{C \subseteq G_i} Weight(G_i)}{\sum_{i=1}^{|D|} Weight(G_i)} = GWSP(C)$$

Thus, C is high graph-weighted clique and $C \subseteq$ HGWCC .

To illustrated the process of clique enumeration: one lattice in the Figure.4 is built from the graph database in our running example (in section 3.2).The sub-clique relationship between two cliques can be represented by their canonical forms and they can be conceptually organized into lattice like structure in depth search first order. Each box represents one clique and its canonical form, edge between two boxes

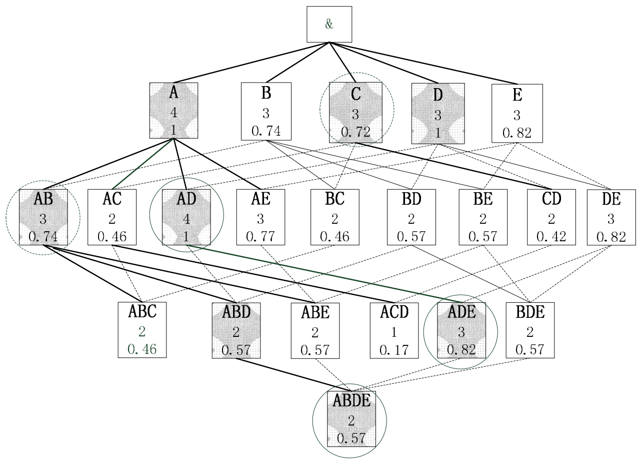


Fig. 4. Clique traverse-lattice tree related to the example in section 3.2. Canonical form boxes covered by circle (solid and dashed) are high graph-weighted closed cliques when Weighted Support $\mathcal{E}$ is 0.5. Gray-shaded boxes denote the search space. The number in the middle and bottom of canonical box is occurrence and graph-weighted support, separately. Clique with solid circle are the high weighted cliques after pruned.

means the sub-clique relationship. One such example in the lattice tree is one of cliques: ABE. It is supported by graph 1 and 3, and its frequent support is 2. Its graph-weighted support is (49+47)/(49+29+47+43)=0.82.

4.1.2 Rechecking Procedure

From phase 1 one have generated a set of High Graph-Weighted Closed Clique (HGWCC), but it may contain some false-positive results, we call such sub-clique **Pseudo High Weighted Closed Clique (PHWCC)**. We calculate its Weighted Support for each HGWCC. If it is a PHWCC, it should be pruned, otherwise it is kept. For example, in the Figure.4 Clique(AB) with graph-weighted support 0.74 should be pruned, because its weighted support is 0.29 and is lower to ε=0.5. What is more, in order to reduce the computation cost, the weight of the entire graph and the result clique should be calculated beforehand and stored in a data structure. Since the number of graph and result clique is not huge, and the weight of each graph or clique is represented as a single value, the space cost is not high.

5 Experiments

In this section, we present empirical results. At first, the real application of birds' migration database is introduced. The birds migration location data sets are converted into a graph database contain 59 graphs. On average, there are 314 edges and 37 vertices in each graph. The maxim one is the one graph from bar headed goose data sets, which owns 540 edges and 67 vertices. Moreover the algorithm CLAN and HELEN is discussed how to discover some highly related birds' migration habitats, and present some empirical results in the section 5.1. In addition, we would evaluate the HELN algorithm's efficiency and scalability (see section 5.2). Finally, we discuss the relationship between highly closed birds' migration habitats with H5N1 incidents in 5.3. The experiments are preformed on a 1.83 GHZ Inter(R) core(TM) CPU with 2G memory and Windows XP platform. The program is implemented in Java. All of our results are embedded into the Google Map.

5.1 Frequent and High Weighted Closed Cliques from Birds Migration Network

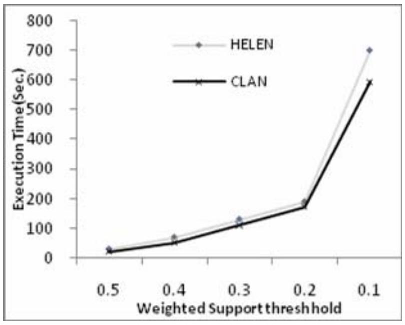
Table 2 shows the number of HGWCC and HWCC. As the decrease in the closed large clique would bring an increase on the small closed cliques for compensate, the total amounts of high weighted closed clique will not change totally as a result. In order to evaluate if the high weighted mining results is useful to the biologist research, we compare the high weighted cliques with the frequent cliques mined by traditional methods. We do observe a number of interesting cliques. For example, a clique in Fig.6(a) is a not frequent item (its frequent is 3), however, its contribution the total time of birds spring migration is more than 5.2%.

Table 2. Experiment summary of birds' migration data

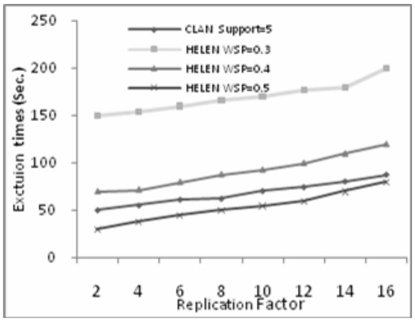
Minimum Weighted support Threshold	Run Times (Seconds)	#Candidates Cliques (Size>2)	#High Graph-weighted closed clique	#High Weighted closed clique
0.5	20	27	15	8
0.4	50	56	34	14
0.3	109	94	53	31
0.2	172	122	79	44
0.1	694	145	91	70

5.2 Efficiency and Scalability Test of HELEN

To evaluate the efficiency of the algorithm, the proposed algorithm HELEN was compared with CLAN. The desired results sets of HELEN and CLAN are different, and their performance is not suitable for comparable. However, our purpose here is show that our algorithm could handle high weighted graph mining problem without increase time consuming greatly. At first, we compare two algorithms with birds' migration data sets. Figure.5(a) shows the runtime of our algorithm by varying the weighted support threshold from 0.1 to 0.5. The results illustrated that since the number of candidate cliques decreases as the minimum weighted support increases, the execution time decreases, correspondingly. Time cost of rechecking would not increase greatly comparing to CLAN since we have saved parts of results before (see section 4.1.2). We also evaluate HELEN's scalability using several real databases in terms of the base size. In Figure.5(b) we replicated the birds' migration graphs from 2 to 16 times. It is evident that HELEN shows a linear scalability in runtime against the number of graphs in the database.



(a) Efficiency

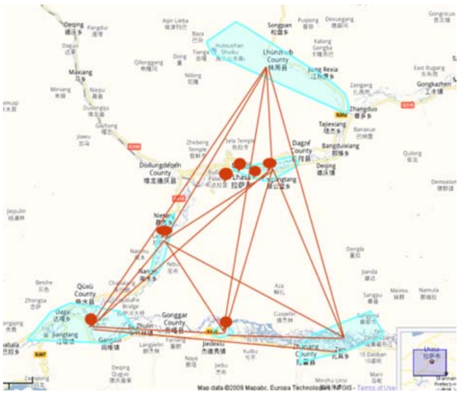


(b) Scalability

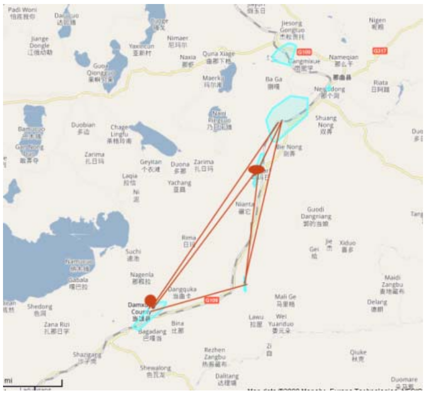
Fig. 5. Efficiency and Scalability Test of HELEN

5.3 Waterbirds Movements in Relation to High Related Habitats and H5N1 Outbreaks

Information about H5N1 outbreaks were obtained from the Ministry of Agriculture of the People's Republic of China Database and OIE Database for the period 16 February 2004–18 May 2009. From our experiment results in the Figure 6, the correlation between birds' migration action and the timing of H5N1 incidents in waterbirds region around Lasha, China was very high. This place is one of the most important areas for waterbirds to overwinter, and with high density of population and poultry. The high weighted clique in the Figure 6 means that waterbirds incline to stay those habitats in a longer time, since both of cliques have high weighted support 5.2% and



(a) One clique with frequent 3 and the Weighted support is 5.2%



(b) One clique with frequent 2 and the Weighted support is 3.1%

Fig. 6. High Weighted Cliques related to Birds Migration Habitats and H5N1 outbreak locations including wild and domestic birds. Circle in blue (habitats), line in red (migration routes), and diamond circle (H5N1 once outbreak location). Vertex weight is time of birds' spring migration, which lasts from 2008 Oct 10th to 2008 Nov 21th.

3.1%. For this matter, we can see that H5N1 outbreaks involving waterbirds occurred during winter, when the potential for interaction with poultry and probability of direct transmission from poultry to migratory waterbirds was predicted to be highest. The highly closed habitats also could be consider follow: waterbirds are directly infected from poultry (i.e., spillover), and they may be responsible for local movement of virus regionally, followed by the potential to transmit virus back to poultry (i.e., spillback)[10,11].

6 Conclusion

In this paper, we suggest to explore the field by using the location data information as a supplement data mining process which can provide an alternative approach for traditional bird telemetry data analysis: visual observation from the location points. In order to discover high weighted cliques, we develop new algorithm HELEN. Our experiment shows that frequent and high weighted clique mining do provide an effective assistance for biologists to discover new correlated relationship between habitats. In the future, we plan to extend our current work to address several unresolved issues. Specifically, we intend to extend the techniques proposed in this paper to mine high weighted closed quasi-cliques.

References

1. Liu, J., et al.: Highly pathogenic H5N1 influenza virus infection in migratory birds. Science 309, 1206 (2005)
2. Tang, M., Zhou, Y., Cui, P., Wang, W., Li, J., Zhang, H., Hou, Y., Yan, B.: Discovery of Migration Habitats and Routes of Wild Bird Species by Clustering and Association Analysis. In: Huang, R., Yang, Q., Pei, J., Gama, J., Meng, X., Li, X. (eds.) ADMA2009. LNCS, vol. 5678, pp. 288–301. Springer, Heidelberg (2009)
3. Wang, J., Zeng, Z., Zhou, L.: CLAN: An Algorithm for Mining Closed Cliques From Large Dense Graph Databases. In: ICDE 2006 (2006)
4. Zeng, Z., Wang, J., Zhou, L., Karypis, G.: Coherent closed quasi-clique discovery from large dense graph databases. In: SIGKDD 2006 (2006)
5. Pei, J., et al.: Mining Cross-graph Quasi-cliques in Gene Expression and Protein Interaction Data. In: ICDE 2005 (2005)
6. Ying, L., Liao, Choudhary, A.: A Fast High Utility Itemsets Mining Algorithm. In: UBDM 2005 (2005)
7. Tao, F., et al.: Weighted Association Rule Mining using Weighted Support and Significance Framework. In: SIGKDD 2003 (2003)
8. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. In: Proc. of the 20th VLDB Conference (1994)
9. Li, et al.: Numbers and distribution of waterbirds and wetlands in the Asia-Pacific region: results of the Asian Waterbird Census Wetlands International (2007)
10. Newman, S.H., Iverson, S.A., et al.: Migration of Whooper Swans and Outbreaks of Highly Pathogenic Avian Influenza H5N1 Virus in Eastern Asia. PLos ONE 4(5) (May 2009)
11. Sturm-Ramirez, K.M., Hulse-Post, D.J., Govorkova, E.A., Humberd, J., Seiler, P., et al.: Are ducks contributing to the endemicity of highly pathogenic H5N1influenza virus in Asia? J. Virol. 79, 11269–11279 (2005)

Performance Improvement of OpenJPA by Query Dependency Analysis

Miki Enoki, Yosuke Ozawa, and Tamiya Onodera

IBM Research – Tokyo, Japan
{enomiki,ozawaysk,tonodera}@jp.ibm.com

Abstract. OpenJPA is an implementation of the Java persistence API (JPA) for Apache, with a caching layer for databases queries to share cached objects among multiple client sessions. This is a critical component for high performance, since the caching layer can handle many database requests. However the performance is limited when an application includes write transactions, because the current OpenJPA cache invalidation mechanism is course-grained and this results in a low cache hit rate. We have implemented two kinds of finer-grained invalidation mechanisms by using query dependency analysis and integrated them into OpenJPA. In our experiments with TPC-W, the OpenJPA with the finer-grained invalidation mechanisms outperformed the current OpenJPA. In addition, we created many more mixes TPC-W, and found that the finer mechanism is not necessarily the better, that is, the best mechanism varies depending on the mixes.

Keywords: EJB3.0, JPA, OpenJPA, Query dependency analysis, Cache invalidation.

1 Introduction

Enterprise JavaBeans (EJB) [1] is the server-side component architecture for component-based distributed applications using Java. EJB is extensively used for application development since it permits the secure and portable handling of persistent data in the form of distributed objects. The EJB 3.0 specification of 2006 made it easier to develop applications by replacing many of the complicated techniques used in EJB 2.1. The Java Persistence API (JPA) that was defined as a part of the EJB 3.0 specification standardizes the Object-Relational (O/R) mapping to enhance the capabilities of EJB. JPA simplifies the difficult task of using object-oriented programming with a relational database by replacing direct persistence-related database accesses with high-level object handling functions. In addition, JPA provides the Java Persistence Query Language (JPQL) which is an SQL-like language to retrieve and manipulate data.

There are several JPA implementations such as Apache OpenJPA [3], Hibernate [4], and TopLink Essentials [5]. Each of them has a caching layer for database queries. For example, OpenJPA has two types of cache, a DataCache and a QueryCache. The DataCache stores entities loaded from a database and each instance corresponds to a tuple in a table. The QueryCache stores lists of the object ids returned by queries.

However the performance of a cache is limited when an application includes write transactions, because the current OpenJPA cache invalidation mechanism is course-grained and this results in a low cache hit rate.

In this paper, we introduce two invalidation mechanisms that use query dependency analysis for finer-grained invalidations in the OpenJPA caching layer and evaluate their effectiveness on TPC-W benchmark. The contributions of our paper are as follows.

- Implementation of two finer-grained invalidation mechanisms for OpenJPA. We implemented two invalidation mechanisms, column-based and value-based, and integrated them into OpenJPA.
- Evaluation with an industry standard benchmark. We empirically studied the effectiveness of the invalidation mechanisms with TPC-W, and found that the fine-grained mechanisms perform better in all the three mixes (scenarios) than the original mechanism. In addition, we created many more mixes TPC-W, and found that the finer mechanism is not necessarily the better, that is, the best mechanism varies depending on the mixes.

The outline of the rest of this paper is as follows. Section 2 describes the cache mechanism of OpenJPA. Section 3 provides explanations of query dependency analysis for our two new invalidation mechanisms. Section 4 presents our experimental environment and performance evaluations. Related work is presented in Section 5. Section 6 concludes the paper.

2 Apache OpenJPA and Cache Mechanism

OpenJPA is one of the implementations of JPA by Apache Software Foundation [3]. OpenJPA also provides a caching layer to store the results of queries from a database on the application server side [2]. This caching layer is designed to significantly increase performance while remaining in full compliance with JPA standard. In this section, we describe the cache mechanism of OpenJPA in detail.

2.1 DataCache and QueryCache

A table of a database corresponds to a Java class called an entity in OpenJPA. The client program processes select, update, insert, and delete operations through entity objects. A row of data in a table can also correspond to an instance of an entity. The OpenJPA caching layer consists of two key-value stores.

The first store is called DataCache, which stores data loaded from the database. The key for DataCache is an entity id (primary key of the table) and the value is the entity instance. The second store is called QueryCache, which stores the results of queries. Each key for QueryCache is the combination of a JPQL statement and its instantiated parameters. The value is the list of entity ids returned as the result of that query.

Figure 1 shows the relationships between the OpenJPA caches and a database. When a select query is issued, OpenJPA checks for cached query results in QueryCache before retrieving the data from the database. If cached results are found, the entity ids in the QueryCache are looked up, and that data is returned from DataCache instead of accessing the database. Otherwise, the query is executed against the database, and the response data is loaded into the cache.

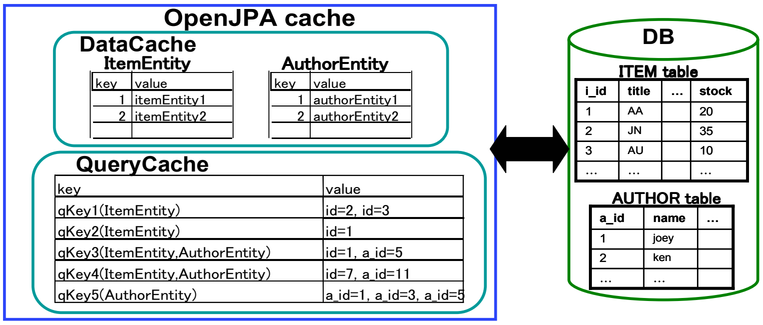


Fig. 1. DataCache and QueryCache

2.2 QueryCache Invalidation Mechanism: Table-Based Invalidation

When an update request is issued, OpenJPA has to maintain consistency with the data in the database. OpenJPA has two update mechanisms: update through entity and update by JPQL. Update through entity means using JPA method for update. When the field data of an entity is modified by JPA method, the corresponded data in the database is updated transparently. Update by JPQL means using JPQL for update. In both cases, OpenJPA invalidates all cache entries which are related to the updated table in QueryCache. That is, the invalidation is table-based.

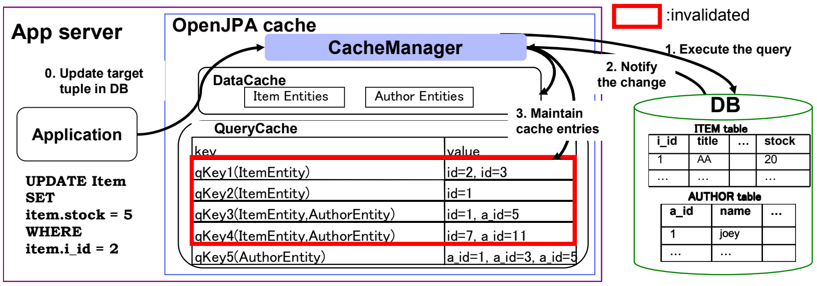

:invalidated
App server
OpenJPA cache
CacheManager
0. Update target tuple in DB
Application
UPDATE Item SET item.stock = 5 WHERE item.i_id = 2
DataCache
Item Entities
Author Entities
QueryCache
key | value
qKey1(ItemEntity) | id=2, id=3
qKey2(ItemEntity) | id=1
qKey3(ItemEntity,AuthorEntity) | id=1, a_id=5
qKey4(ItemEntity,AuthorEntity) | id=7, a_id=11
qKey5(AuthorEntity) | a_id=1, a_id=3, a_id=5
1. Execute the query
2. Notify the change
3. Maintain cache entries
DB
ITEM table
i_id | title | … | stock
1 | AA | | 20
AUTHOR table
a_id | name | …
1 | joey

Fig. 2. Table-based invalidation

Figure 2 illustrates how the table-based invalidation works. This example shows updating the quantity of stock in the entity with id = 2 in the ITEM table. When the entity is updated, OpenJPA invalidates all entries related to ItemEntity in QueryCache. As expected, the table-based invalidation results in low cache hit rates.

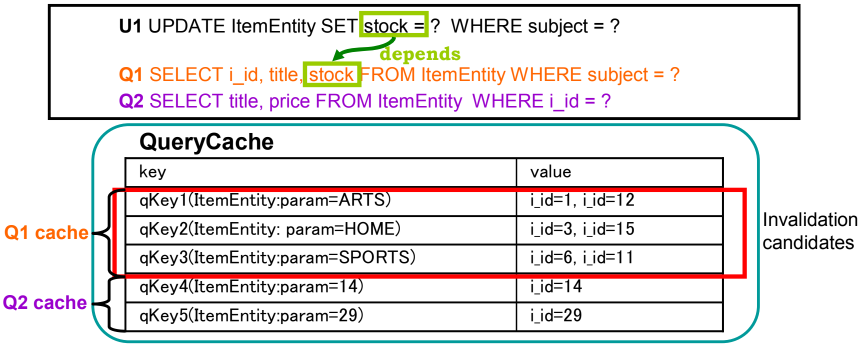
3 Query Dependency Analysis for Finer-Grained Cache Invalidation

We now present two finer-grained invalidation mechanisms, column-based and value-based. We employed query dependency analysis to enable their mechanisms.

3.1 Column-Based Invalidation

In column-based invalidation, the dependencies of the select and update queries are determined by the column-level analysis of queries. Each column of select queries is checked to see if it is affected by any update queries.

Figure 3 shows an example of column-based invalidation. The update query U1 modifies the number in the stock column of entities for which the subject matches the parameter of executed query. Select query Q1 retrieves the i_id, title and stock data from the ITEM table, while Q2 gets the title and price from the ITEM table. Through query dependency analysis, we can determine that U1 affects only Q1 because Q1 selects the same column as U1 and only need to invalidate the cache entries for Q1.Then Q1 is put into an invalidation candidates group associated with U1 and this group is used for the invalidation of U1 at runtime.



```
U1 UPDATE ItemEntity SET stock = ? WHERE subject = ?
Q1 SELECT i_id, title, stock FROM ItemEntity WHERE subject = ?
Q2 SELECT title, price FROM ItemEntity WHERE i_id = ?
```

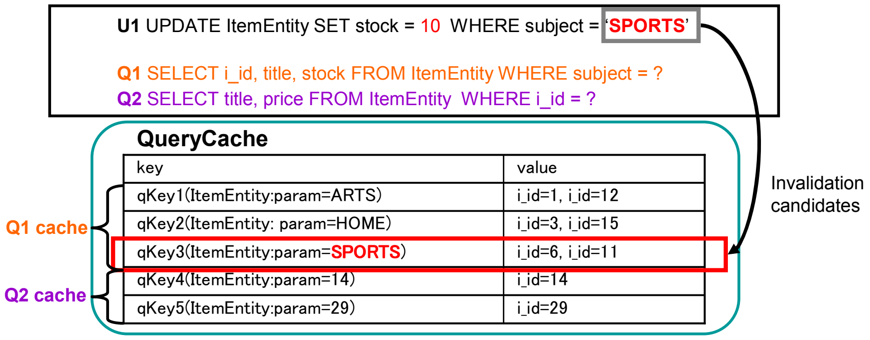
key	value
qKey1(ItemEntity:param=ARTS)	i_id=1, i_id=12
qKey2(ItemEntity: param=HOME)	i_id=3, i_id=15
qKey3(ItemEntity:param=SPORTS)	i_id=6, i_id=11
qKey4(ItemEntity:param=14)	i_id=14
qKey5(ItemEntity:param=29)	i_id=29

Fig. 3. Column-based invalidation

3.2 Value-Based Invalidation

In value-based invalidation, the dependencies of the select and update queries are determined by a value-level analysis of queries. This means that the parameters of the update queries are checked to determine whether or not they affect each cached results at runtime.

Figure 4 shows an example of value-based invalidation. At execution time for the queries, we can obtain the values of the parameters of each query. For example, the U1 can issue a query with the value “SPORTS”, and only the entries which are cached with the parameter value of “SPORTS” will be invalidated.



U1 UPDATE ItemEntity SET stock = 10 WHERE subject = 'SPORTS'

Q1 SELECT i_id, title, stock FROM ItemEntity WHERE subject = ?
Q2 SELECT title, price FROM ItemEntity WHERE i_id = ?

key	value
qKey1(ItemEntity:param=ARTS)	i_id=1, i_id=12
qKey2(ItemEntity: param=HOME)	i_id=3, i_id=15
qKey3(ItemEntity:param=SPORTS)	i_id=6, i_id=11
qKey4(ItemEntity:param=14)	i_id=14
qKey5(ItemEntity:param=29)	i_id=29

Fig. 4. Value-based invalidation

3.3 Procedure for Column-Based and Value-Based Invalidation

Figure 5 represents the procedure for column-based and value-based invalidation.

When an update query is issued, its dependency with cache entries in QueryCache is checked. In case of value-based invalidation, it is additionally analyzed in query parameter level. Our query dependency analysis is based on query-update independent analysis [16].

```
invalidate_QueryCache(updateQuery)
//iterate until all query pattern in QueryCache are checked
 while(selectQueries.hasNext()){
   selectQuery = selectQueries.getNext();
   //check dependency by the column-level analysis
   isdepend = doColumnBasedDependencyAnalysis (updateQuery, selectQuery);
     if(isdepend){
     //add keys to list for invalidation
      invalidationList.add(getQueryKeys(selectQuery));
     }
 }
//invalidate only cache entries related in update Query
 if(column_based_invalidation){
   invalidate(invalidationList);
 }else if(value_based_invalidation){
   //vlue-level analysis
   while(invalidationList.hasNext()){
     selectQuery = invalidationList.getNext();
      //check dependency by the value-level analysis
     isdepend = doValueBasedDependencyAnalysis(updateQuery,uValues,selectQuery,sValues);
     if(isdepend){
          valueBasedInvalidationList.add(getQueryKeys(selectQuery));
     }
    invalidate(valueBasedInvalidationList);
 }
}
```

Fig. 5. Procedure of column-based and value-based invalidation

3.4 Usage with OpenJPA

There are two ways to update in OpenJPA as described in Section 2.2, update through entity and update by JPQL. Hence, we have to apply query dependency analysis regardless of how the update request is issued. This process is illustrated in Figure 6. First, the update query is executed using both update methods, and then extract the information to apply query dependency analysis for column-based or value-based invalidation. Next, query dependency analysis module analyzes their dependency, and generates the invalidation candidates.

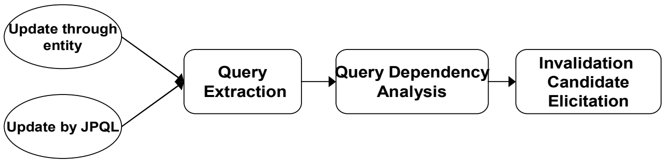


Fig. 6. Query dependency analysis in OpenJPA

3.5 Invalidation Cost Models

Ideally, when any update query issued, only the affected data in cache should be invalidated so as to keep high cache hit rate. However, strict cache maintenance sometimes brings overhead, thus the finer mechanisms may not necessarily get the better performance.

To calculate the invalidation cost at runtime, the cost has two components, the cost of removing cache entries C_r and the cost of analyzing query dependencies C_a to specify the cache entries to be invalidated. Therefore the total invalidation cost is defined as $C_i = C_r + C_a$. The values I_p, I_q and I_r are the numbers of cache entries to be invalidated in table-based, column-based, and value-based invalidation, respectively. R is the cost of removing one cache entry. The cost of value-based query dependency analysis is defined as Q_v.

$Table-based\colon C_r = I_p \times R,\ C_a = 0\ Column-based\colon C_r = I_q \times R,\ C_a = 0\ Value-based\colon C_r = I_r \times R,\ C_a = I_q \times Q_V$

C_a is estimated as zero for the table-based and column-based invalidations since the dependency analysis can be done off-line if we can collect all of the query templates in advance. Also these analyses only have to be conducted for the first time for each query templates, even if we cannot collect all of the query templates. In contrast, we have to use on-line analysis for the value-based invalidation since the query parameters are determined only at runtime. The key challenge in efficient maintenance of cache consistency is to determine an appropriate cache invalidation mechanism for each application.

4 Performance Evaluation

In this section, we evaluate the performance of three cache invalidation mechanisms, table-based invalidation (as in the original OpenJPA), column-based invalidation and value-based invalidation. This section evaluates of the performance on various cache invalidation mechanisms with the TPC-W workload.

4.1 TPC-W

The TPC-W benchmark is defined by Transaction Processing Council[12]. It is a transactional web benchmark designed for evaluating e-commerce systems. In TPC-W, web interactions can be classified as either "Browse" or "Order" interactions. TPC-W provides three mixes that have different rate of Browse and Order web interactions, the Browsing mix consists of 95% and 5%, the Shopping mix does 80% and 20%, and the Ordering mix does 50% and 50% respectively. The Browsing mix has a high percentage of read-only interactions, whereas the ordering mix has a high percentage of database modifications (insert, update and delete).

The maximum throughput in TPC-W is defined as the maximum number of web interactions per second (WIPS) sustained over a measurement period while responding to 90% of each type of interaction within given response time thresholds. We used the TPC-W java implementation [11] and set the number of items to 100K and the number of customers to 2.8 million.

4.2 Evaluation Platform

Our environment for this evaluation consists of three servers, a 64-bit Dual-Core AMD Opteron 2218 x2 with 4 GB RAM and Linux 4.2 for the client server, two 64-bit Dual-Core AMD Opteron 2222 x2 with 8GB RAM and Linux 4.2 for the application server and the database server. We installed DB2 UDB v9.129 as the database server and WebSphere Application Server v7.0 as the application server. We used OpenJPA v1.2.0 and WebSphere eXtreme Scale as the cache pug-in. No matter what cache plug-in of OpenJPA is used, the invalidation mechanism isn't different since its methods are defined by OpenJPA. The client server sends requests using EBs (Emulated Browsers). In this experiment, we modified the data access part of the TPC-W application to use OpenJPA.

4.3 The Performance Comparison of Each Cache Invalidation

We compared the OpenJPA cache invalidation performance using TPC-W for four approaches: (1) NoCache: OpenJPA with no cache, (2) Base: the original OpenJPA cache with table-based invalidation, (3) OpenJPA with a cache using column-based invalidation, and (4) OpenJPA with a cache using value-based invalidation. Figure 7 shows the maximum throughput of TPC-W for each approach with each mix and Table 1 represents the cache hit rate for QueryCache.

The throughput of the column-based and value-based invalidation mechanisms is better than Base for all mixes. The value-based invalidation has about two times better throughput than NoCache and the cache hit rate for the QueryCache was increased to 92% for the Browsing mix. However, the throughputs of the cache topologies decreased as the ratio of the orders increased. It seems that cache invalidation cost harms caching effectiveness. The performance differences between column-based invalidation and value-based invalidation also decreased even though the cache hit rates of value-based invalidation remained high.

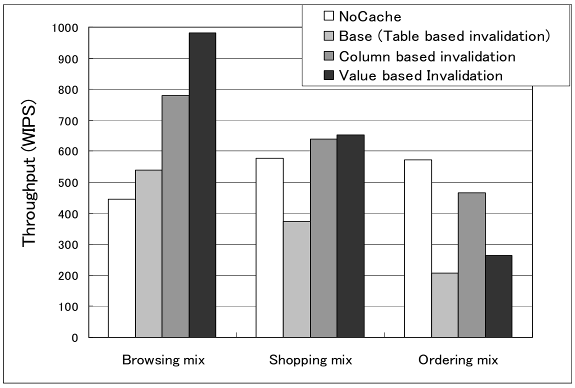


Fig. 7. Performance comparison in TPC-W

Table 1. Cache hit rate of QueryCache and DataCache

	Browsing mix			Shopping mix			Ordering mix		
	Base	Column	Value	Base	Column	Value	Base	Column	Value
QueryCache	17%	24%	92%	8%	14%	82%	4%	21%	71%

The CPU utilization of the database server reached more than 90% while the application server remained under 30% in the NoCache test, while the cache-using topologies kept the database server below 50% while the application server was above 90% for all of the mixes. This indicates that NoCache causes the database server to become the bottleneck. In contrast, the cache maintenance costs affect the application server performance. However, since the cache topologies greatly reduce the load on the database, the throughput can be increased by scaling out the application servers.

In the next experiment, we evaluated the advantages of the cache topologies by modifying the ratios of Browse and Order between the Browsing mix and Ordering mix ratios. The results are shown in Figure 8. The x-axis represents the ratio of Browse and Order and the y-axis is the relative value of the three cache invalidation mechanisms based on NoCache. The value 1 on the y-axis shows the performance of NoCache for each ratio of Browse and Order.

From these experiments, we see that whereas the performance of base falls below NoCache at the ratio of orders 10%, column-based and value-based invalidation can keep the effect of cache to the ratio of orders 25%.

The performance of column-based and value-based invalidation are almost the same at the ratio of orders is about 20%, and the performance of column-based invalidation is marginally better than value-based invalidation. This is because the cost of value-based invalidation is the largest of the three mechanisms, as described in Section 3.5. We found that the most suitable cache invalidation mechanisms depend on the ratio of update queries, which reflects the characteristics of each application.

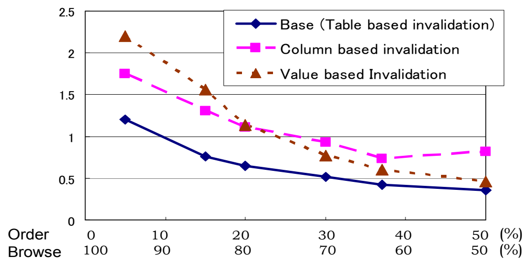


Fig. 8. Performance comparison of each cache invalidation

5 Related Work

There has been previous work on the performance of various early versions of EJB. Paul et al. [6] explored the impact on performance of three commit options which are defined in the EJB 1.1 specification. Emmanuel et al. [7] experimented with several EJB 2.0 implementations by using different application implementation methods, container designs and communication layers. Avraham et al. [8] compared the performance among JDBC direct using application, the EJB application with and without a cache enabled. In these experiments, the EJB application with the cache showed the advantage of scale out. For EJB 3.0, Ben et al. [9] presented a prototype Java compiler with query extraction so as to optimize the amount of data loaded in each select query. Zachary et al. [10] added JPQL type checking since the JPQL queries are not analyzed in compilation.

Much previous work about database query caching exists. For example, DBCache [14] caches only the data which is frequently retrieved. DBProxy [15] caches the data as a materialized view. These cache storing techniques are different with which of OpenJPA. OpenJPA stores all query results and data respectively, hence we uniquely defined cache maintenance mechanism. Ferdinand[13] stores database query cache and use offline analysis of queries and updates to know the dependency for making fewer multicast groups for update notification. Our work is different from their focus. We evaluated the effectiveness of some cache invalidation mechanism.

6 Conclusions and Future Work

OpenJPA is an implementation of the Java persistence API (JPA), with a caching layer for databases queries. However, the performance is limited because of the current course-grained OpenJPA cache invalidation mechanism.

To improve the cache hit rates for high performance, we created two kinds of finer grained invalidation mechanisms by using query dependency analysis, column-based and value-based invalidation. In our experiments with TPC-W, these new invalidation mechanisms outperformed the current OpenJPA. In addition, we created many more mixes TPC-W, and found that the finer mechanism is not necessarily the better, that is, the best mechanism varies depending on the mixes.

Since the current OpenJPA implementation for cache maintenance is based on eager maintenance, we focused on improving the eager model. However, lazy maintenance is also used in many scenarios. In our future research we may compare the performance including lazy maintenance approaches. In the future, we generalize how to determine the suitable invalidation mechanism depending on each application characteristic, and create and apply more various grained invalidation mechanism to each query. We also add application servers for scale out to get better performance.

References

1. EJB 3.0 Expert Group, JSR 220: Enterprise JavaBeans Version 3.0 Java Persistence API, Sun Microsystems, Santa Clara, CA (2006)
2. Patrick, L., Mark, P.: An in-depth look at the architecture of an object/relational mapper. In: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pp. 889–894 (2007)
3. Apache OpenJPA, `http://openjpa.apache.org/`
4. Hibernate, `http://www.hibernate.org/`
5. TopLink Essentials, `https://glassfish.dev.java.net/javaee5/persistence/`
6. Paul, B., Shuping, R.: Entity Bean A, B, C's: Enterprise Java Beans Commit Options and Caching. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 36–55. Springer, Heidelberg (2001)
7. Emmanuel, C., Julie, M., Willy, Z.: Performance and scalability of EJB applications. In: Proc. 17th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2002), pp. 246–261 (2002)
8. Avraham, L., James, T.: Improving Application Throughput With Enterprise JavaBeans Caching. In: Proceedings of the 23rd International Conference on Distributed Computing Systems (ICDCS 2003), pp. 244–251 (2003)
9. Ben, W., Ali, I., William, R.: Interprocedural query extraction for transparent persistence. In: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications (OOPSLA 2008), pp. 19–36 (2008)
10. Zachary, T., Chris, T., David, S., Ranjit, J., Sorin, L.: Deep typechecking and refactoring. In: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications (OOPSLA 2008), pp. 37–52 (2008)
11. Harold, W., Ravi, R., Mowwis, M., Mikko, H.: An Architectural Evaluation of Java TPC-W. In: Seventh International Symposium on High-Performance Computer Architecture (HPCA 2001), p. 229 (2001)
12. Transaction Processing Council. TPC-W specification, `http://www.tpc.org.tpcw`
13. Charles, G., Amit, M., Anastasia, A., Bruce, M., Todd, M., Christopher, O., Anthony, T.: Scalable Query Result Caching four Web Applications. In: Proceedings of the 34th Very Large Databases (VLDB 2008), pp. 550–561 (2008)
14. Luo, Q., Krishnamurthy, S., Mohan, C., Pirahesh, H., Woo, H., Lindsay, B.G., Naughton, J.F.: Middle-tier database caching for e-business. In: Proc. ACM SIGMOD International Conference on Management of Data (2002)
15. Amiri, K., Park, S., Tewari, R., Padmanabhan, S.: DBProxy: A dynamic data cache for Web applications. In: Proc. International Conference on Data Engineering, ICDE 2003 (2003)
16. Levy, A.Y., Sagiv, Y.: Queries independent of updates. In: Proc. International Conference on Very Large Data Bases, VLDB 1993 (1993)

Chimera: Stream-Oriented XML Filtering/Querying Engine

Tatsuya Asai[1], Shin-ichiro Tago[1], Hiroya Inakoshi[1], Seishi Okamoto[1], and Masayuki Takeda[2]

[1] Fujitsu Laboratories Ltd., Kawasaki 211–8588, Japan
{asai.tatsuya@jp,s-tago@jp,inakoshi.hiroya@jp,seishi@labs}.fujitsu.com
[2] Kyushu University, Fukuoka 819–0395, Japan
takeda@inf.kyushu-u.ac.jp

Abstract. In this paper, we study the problem of filtering and querying massive XML data against a large set of XPath patterns in Univariate XPath. Based on an efficient matching engine XSIGMA for linear XPath patterns with Boolean expression over keywords and a twig evaluator over event streams, we propose an XPath filtering/querying engine *Chimera*, which runs fast and stably for any XPath patterns without heavy pre- processing techniques for queried data often used by existing native XMLDBs and RDBs. Chimera also runs much faster than those engines against thousands of XPath patterns. We implemented Chimera and showed its effectiveness by several experiments on artificial and real datasets.

1 Introduction

In the present age called the *Info-plosion era*, efficient processing for massive XML data has been more important. Thus, there have been increasing demands for efficient processing engines for standard XML querying languages such as XPath [7], XQuery [4], and XSLT [6].

Existing XML querying engines are mainly divided into the following two categories: (i) Stored data processing, and (ii) Stream processing. For stored data processing, there are various commercial engines such as native XMLDB engines and RDB engines with features for processing XML. Those engines enable us to achieve higher performance for a tuned set of queries by using indices and normalizing queried data. However, the burden of developing large and complex systems with those engines has been increasing due to the large computational and human costs for the heavy pre-processing that is required, as described above. Furthermore, the response time of those engines strongly depends on the characteristics and amount of input queries. For example, response time becomes much worse when there are queries with many wildcards or when a number of queries are input at once. Therefore, it is difficult to achieve fast and stable performance for any input set of queries with those engines.

Stream processing technologies for XML data such as XML filtering [2,8,9,10] and stream XML querying [5,11,13] have been widely studied and a lot of prototyping systems have been proposed since around 2000. Most of those studies achieved scalable performance against both data and query sizes to filter or to query infinite XML streams continuously, while they restrict classes of queries for fast and light-weight stream processing.

The purpose of this study is to develop an XPath querying engine that works quickly and stably for massive XML datasets stored and multiple queries input with light-weight pre-processing of data, instead of requiring heavy pre-processing and human operations like traditional stored data processing engines.

In this paper, we consider stream-oriented filtering and querying methods for stored large XML data and propose a fast and scalable querying engine *Chimera* against a large set of XPath patterns included in Univariate XPath [3,9]. Chimera's core engine, named XSIGMA, filters thousands of primitive XPath patterns efficiently by a light-weight pre-processing technique for XML data [10] and a DFA detecting together both occurrences of paths and keywords. Chimera also includes evaluating engines for output event streams from XSIGMA. By adopting such a stream-oriented architecture, Chimera achieves a fast and stable performance against a large set of XPath patterns, in contrast with traditional processing engines for stored XML data.

We show the effectiveness of Chimera by comparing it with several stream XPath filtering/querying engines and a commercial RDB engine. The experimental results show that Chimera runs fast and stably for any XPath patterns without the need for heavy pre-processing techniques often used by existing native XMLDBs and RDBs. Furthermore, Chimera runs much faster than those engines against thousands of XPath patterns.

1.1 Related Work

Several XPath streaming engines have been proposed. Yfilter [8] builds an NFA for the queries by merging common prefixes of the query paths. XMLTK [2] adopted the Lazy-DFA technique that converts the NFA for the queries into a DFA lazily. TwigList [13] constructs tree-like structures for matching a twig pattern from XML streams. SPEX [11] processes regular path expressions including reverse axes [12]. The most related research is XAXEN [10], which is a stream-oriented XML filtering engine using a light-weight pre-processing technique for XML data.

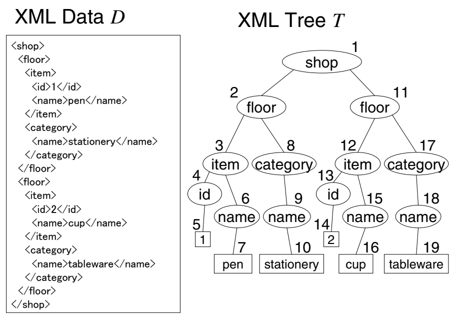
1.2 Organization

This paper is organized as follows. In Section 2, we prepare basic notions and definitions. In Section 3, we present the architecture of Chimera. In Section 4, we report results from experiments on artificial and real datasets, and finally, we conclude this paper in Section 5.

2 Preliminaries

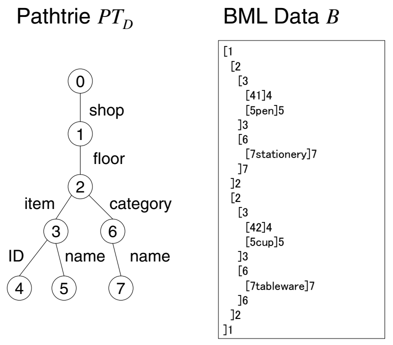
2.1 XML Trees and Pathtrie

First, we define a model of XML data. Let Σ be an alphabet and Σ^* denote the set of strings over Σ. Let $\mathcal{T}$ be a set of *tagnames*. Then, an *XML tree* T is an



```
<shop>
 <floor>
  <item>
   <id>1</id>
   <name>pen</name>
  </item>
  <category>
   <name>stationery</name>
  </category>
 </floor>
 <floor>
  <item>
   <id>2</id>
   <name>cup</name>
  </item>
  <category>
   <name>tableware</name>
  </category>
 </floor>
</shop>
```

Fig. 1. XML data D and its tree representation T



```
[1
 [2
  [3
   [41]4
   [5pen]5
  ]3
  [6
   [7stationery]7
  ]7
 ]2
 [2
  [3
   [42]4
   [5cup]5
  ]3
  [6
   [7tableware]7
  ]6
 ]2
]1
```

Fig. 2. Pathtrie PT_D and BML data B for the XML data D

ordered tree such that the interior nodes, called the *element nodes*, are labeled by tagnames in $\mathcal{T}$ and the leaves, called the *text nodes*, are labeled by strings in Σ^*. Fig. 1 shows an XML data D and its tree representation T, where the circles and the squares represent element and text nodes, respectively. The strings in element and text nodes in T represent tagnames and text values, respectively. The numbers adjacent to nodes of T mean their IDs.

Let T be an XML tree. Then, V_T and E_T denote the node set and the edge set of T, respectively. Let $k \geq 0$ be a non-negative integer. Then, a *path* $p = (v_0, \ldots, v_k) \in V_T^*$ in D is defined as a sequence from the root $r = v_0$ to an element node v_k, where $(v_i, v_{i+1}) \in E_T$ holds for every $i = 0, \ldots, k-1$. Let $p = (v_0, \ldots, v_k)$ be a path in T and $\text{tag}(p) = (\text{tag}(v_0), \ldots, \text{tag}(v_k)) \in \mathcal{T}^*$ be the sequence of tagnames associated with each node v_i. Let D be an XML data and T be its tree representation. Then, the *pathtrie* [10] PT_D for D is defined as a trie with alphabet $\mathcal{T}$ representing the set of tagname sequences for all the paths in T. We associate each node in PT_D with a non-negative integer $k = 0, 1, \ldots, |PT_D| - 1$ called a *path ID*, where $|PT_D|$ is the size of PT_D. The figure on the left in Fig. 2 shows the pathtrie for the XML data D in Fig. 1, where the strings associated with the edges mean tagnames and numbers in the circular nodes mean path IDs.

2.2 Fragment of XPath Patterns

The fragment of XPath that Chimera supports is *Univariate XPath* [3,9], the grammar of which is shown in Fig. 3. In what follows, *XPath patterns* mean any strings generated by the grammar. If an XPath pattern has no predicates, we call it a *linear XPath pattern* (or *linear path* for short).

An XPath pattern specifies a *query tree* Q to navigate an XML tree T. Now, we will give a definition of query trees and their occurrences in XML trees. A query tree Q is an unordered tree with a special node *Out*, called the *output node*, such that the interior nodes are labeled by tagnames in $\mathcal{T}$, the leaves are labeled by Boolean expression over strings in Σ^*, and the edges are labeled by '/' (child) or '//' (descendant).

```
Path := Step Path Step
Step := Axis NodeTest | Axis NodeTest '[' Predicate ']'
Axis := '/' | '//'
NodeTest := Tagname | '*'
Predicate := Path | Contains(Path, String) | Path CompOp Constant
   | Predicate 'and' Predicate | Predicate 'or' Predicate | 'not'
   Predicate
CompOp := '=' | '!=' | '>' | '>=' | '<' | '<='
```

Fig. 3. Univariate XPath

For an XML tree T and a query tree Q, we say that Q *occurs in* T if there exists a *matching function* $\varphi : V_T \rightarrow V_Q$ satisfying the following conditions for any $v, v_1, v_2 \in V_Q$:

– $(\varphi(v_1), \varphi(v_2)) \in E_T$ holds if the edge-label of $(v_1, v_2) \in E_Q$ is '/' and $(\varphi(v_1), \varphi(v_2)) \in E_T^+$ holds if the edge-label of $(v_1, v_2) \in E_Q$ is '//'
– $label_Q(v) = label_T(\varphi(v))$ holds for every internal node $v \in V_Q$
– $label_Q(v)$ matches $label_T(\varphi(v))$ in the sense of string pattern matching for every leaf $v \in V_Q$

where $label_Q(v)$ and $label_T(w)$ denote the node-labels assigned to $v \in V_Q$ and $w \in V_T$, respectively. An *occurrence* of Q in T is defined as a node $\varphi(Out) \in V_T$ such that $Out \in V_Q$ is the output node of Q and φ is a matching function from Q in T. If $v \in V_T$ is an occurrence of Q in T, we say that Q *occurs at* $v \in V_T$. We denote by $Occ(Q, T) \subseteq V_T$ the set of all occurrences of Q in T.

2.3 Problem Statement

We give a formal definition of two basic problems we address in this paper as follows:

Definition 1 (XPath filtering problem). For a given XML data D and a given set $X = \{x_1, \ldots, x_m\}$ of XPath patterns, return the set $Occ(X, T) := Occ(x_1, T) \cup \cdots \cup Occ(x_m, T) \subseteq V_T$, where T is the tree representation of D.

Definition 2 (XPath querying problem). For a given XML data D and a given set $X = \{x_1, \ldots, x_m\}$ of XPath patterns, return each set $Occ(x_i, T) \subseteq V_T$ for every $i = 1, \ldots, m$, where T is the tree representation of D.

3 Architecture

3.1 Overview of Chimera

In Fig. 4, we show the architecture of Chimera. Given a set $X = \{x_1, \ldots, x_m\}$ of XPath patterns, Chimera first decomposes them into a set of primitive patterns described in Section 3.3. The most significant component in the system is

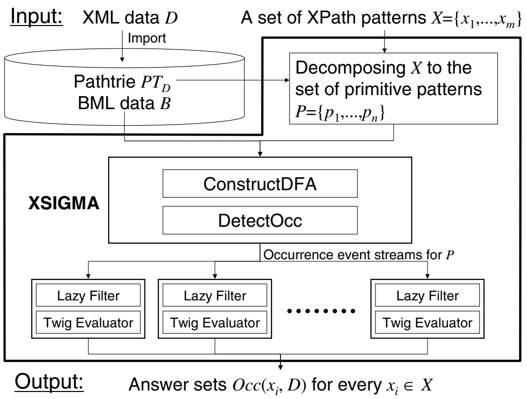

Input:
XML data D
A set of XPath patterns $X=\{x_1,\ldots,x_m\}$
Import
Pathtrie PT_D
BML data B
Decomposing X to the set of primitive patterns $P=\{p_1,\ldots,p_n\}$
XSIGMA
ConstructDFA
DetectOcc
Occurrence event streams for P
Lazy Filter
Twig Evaluator
Output:
Answer sets $Occ(x_i, D)$ for every $x_i \in X$

Fig. 4. Architecture of Chimera

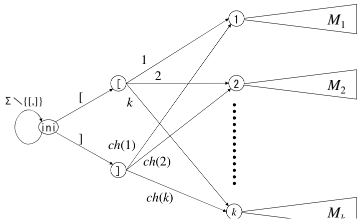


Fig. 5. DFA for matching primitive patterns, where each M_j is an AC machine for matching string expressions under π of path ID j. (Backward transitions are omitted to display)

XSIGMA, which is a fast matching engine for primitive patterns described in Section 3.4. XSIGMA generates occurrence event streams for all primitive patterns in P. To achieve faster matching by XSIGMA, we assume that the input XML data is transformed into its binary representation called BML data described in Section 3.2. Finally, Chimera applies Lazy Filter and Twig Evaluator described in Section 3.5 to the event streams for evaluating all XPath patterns in X. In Chimera, the last evaluating processes can be executed in parallel.

3.2 Binary Transformation of XML Data

For a given XML data D and its pathtrie PT_D, Chimera transforms D into its binary representation B [10], called *BML data for D*, as follows. All occurrences of start tags and end tags in D are, respectively, replaced with '[' and ']' followed by their corresponding path IDs $i \in \{0, 1, \ldots, |PT_D| - 1\}$. We assume that '[' and ']' are reserved characters not included in Σ. This transformation process takes $O(|D| \cdot \log |\mathcal{T}|)$ time [10], where $|\mathcal{T}|$ is the number of distinct tagnames in D. The BML data B transformed from the XML data D of Fig. 1 is shown on the right of Fig. 2, where the path IDs are given by the pathtrie PT_D displayed on the left.

3.3 Decomposing XPath Patterns

Let π be a linear XPath pattern not including descendant axes (//) and *expr* be a Boolean expression over keywords in Σ^*. Then, *primitive patterns* are denoted as $p = \pi\{expr\}$, which means that p occurs at $v \in V_D$ if and only if π occurs at v and *expr* is evaluated to be true under v. For a primitive pattern $p = \pi\{expr\}$, *expr* is called the *string expression under* π.

In the Chimera system, each input XPath pattern x_i is decomposed into a set of primitive patterns. The decomposition procedure is rather straightforward: It first replaces the descendant axes (//) in x_i with their possible tagname sequences with consulting the pathtrie, and then decomposes the resulting tree-shaped patterns into their paths.

Now, we show small examples of pattern decomposition. Suppose that we have the following XPath patterns for the XML data D in Fig. 1:

x_1: `/shop/floor//name[./ = "pen"]`

x_2: `/shop/floor/item[name = "cup"]/ID`

Then, x_1 is decomposed into $p_{11} = /shop/floor/item/name\{"pen"\}$ and $p_{12} = /shop/floor/category/name\{"pen"\}$ by the pathtrie PT in Fig. 2. Also, x_2 is decomposed into $p_{21} = /shop/floor/item\{\varepsilon\}$, $p_{22} = /shop/floor/item/name\{"cup"\}$, and $p_{23} = /shop/floor/item/ID\{\varepsilon\}$.

3.4 XSIGMA: The Core Engine

Let D be an XML data and B be its binary representation. For a given set $P = \{p_1, \ldots, p_n\}$ of primitive patterns and the pathtrie PT_D, XSIGMA first constructs a DFA M for matching the primitive patterns in P. Then, XSIGMA scans B from left to right to find all occurrences of the input primitive patterns and returns sequences $Occ(p_1), \ldots, Occ(p_n)$ of *occurrence events* for every primitive pattern in P. We also call $Occ(p_i)$ an *event stream for* p_i for every $i = 1, \ldots, n$.

In Fig. 6, we show the algorithm ConstructDFA. The algorithm builds a machine M for matching all primitive patterns in P, consisting of Aho-Corasick's pattern matching machines [1] (AC machines, for short) M_π for matching all string expressions under π. Fig. 5 shows a machine M, where the circles and the arrows represent states and transitions, respectively. Moreover, M_j represents an AC machine for matching all string expressions under π where j is π's path ID, for every $j = 1, \ldots, |PT_D| - 1$. Note that $ch(j)$ denotes the set of child nodes of j in the pathtrie PT_D.

Then, we show the algorithm DetectOcc in Fig. 7. Receiving a BML data B and a DFA M constructed by ConstructDFA, the algorithm scans B by one character and make a transition in M. If the algorithm detects an occurrence of a primitive pattern $p_i \in P$, then immediately it generates an *occurrence event* for p_i. After finishing scanning B, the generation of event streams $Occ(p_1), \ldots, Occ(p_n)$ is complete. As the result, DetectOcc finds all occurrences of every primitive pattern in P in $O(|B| + ||Occ||)$ time while it uses $O(|PT_D| \cdot ||Str||)$ space in the worst case, where $||Occ||$ is the total size of the event streams and $||Str||$ is the total length of strings included in P.

3.5 Evaluation for XPath Patterns

In this subsection, we describe the last two modules of Chimera, namely, Twig Evaluator and Lazy Filter. Twig Evaluator constructs tree structures like TwigList [13] from the event streams $Occ(p_1), \ldots, Occ(p_n)$ for evaluating every input XPath pattern in X. Twig Evaluator runs in $O(c \cdot ||Occ||)$ time, where $c = \max_{x_i \in X} |x_i|$ and $||Occ||$ denotes the total size of the event streams. Therefore, Twig Evaluator will be a bottleneck of Chimera when $||Occ||$ is quite large.

```
Algorithm ConstructDFA:
Input: A set $P = \{p_1, \ldots, p_n\}$ of primitive patterns and a pathtrie $PT$.
Output: A DFA $M$.
1. For every linear path $\pi$ in $PT$, build an AC machine $M_\pi$ for matching all string
   expressions under $\pi$ in $P$.
2. Connect all machines $M_\pi$ as displayed in Fig. 5 and let $M$ be the resultant machine.
3. Return $M$.
```

Fig. 6. An algorithm for constructing a DFA

```
Algorithm DetectOcc:
Input: A BML data $B$ and a DFA $M$ constructed by ConstructDFA.
Output: An event stream $Occ$.
1. Set $i := 0$ and $s$ be the initial state of $M$.
2. While $B[i] \neq$ EOF, do the followings: /* scanning by one character */
     – $s := goto(s, B[i])$. /* making a transition */
     – If an occurrence of $p_i \in P$ is found, then return an occurrence event for $p_i$.
     – $i := i + 1$.
```

Fig. 7. An algorithm for finding occurrences of primitive patterns

To overcome this difficulty, we have devised Lazy Filter, which is a fast and light-weight filtering module for event streams, and arranged it in front of Twig Evaluator. Lazy Filter recognizes and removes such events in advance to be obviously evaluated as false in Twig Evaluator by checking lazy logical conditions. For example, suppose that we have the pattern $/A_1[B_1]/\ldots/A_n[B_n]$. Then, Lazy Filter evaluates the logical condition $B_1 \wedge \ldots \wedge B_n$ on the event streams. If the condition is false, the corresponding events are removed. A complex twig pattern tends to cause large event streams since the number of decomposed primitive patterns becomes large and they match data frequently. Thus, Lazy Filter plays an important role in Chimera for achieving fast and stable performance.

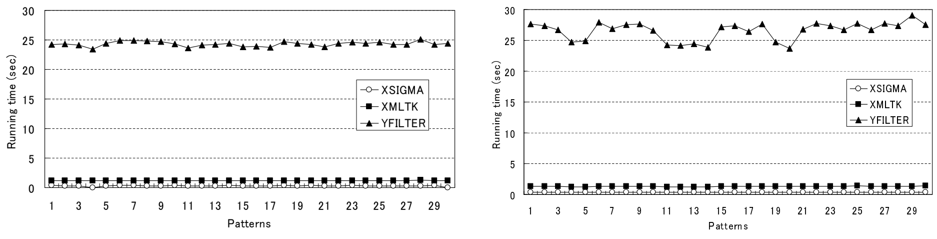
4 Experimental Results

In this section, we will present experimental results on artificial and real datasets to evaluate the performance of Chimera. We implemented the prototyping system in C. The experiments were run on a PC (Intel 2.66GHz dual core, CentOS 5.2) with 4GB of main memory. In Table 1, we show datasets prepared for the experiments. The data XMark1, XMark5, and XMark10 are randomly generated by xmlgen[1] with scaling factors 1, 5, and 10, respectively. The data dblp[2] is bibliographic information on major computer science. We also randomly generated

[1] http://monetdb.cwi.nl/xml/downloads.html
[2] http://www.cs.washington.edu/research/xmldatasets/www/repository.html#dblp

Table 1. Datasets for the experiments

	XMark1	XMark5	XMark10	dblp
Data size (MB)	111	558	1,118	128
BML size (MB)	120	603	1,208	129
# tagnames	74	74	74	35
# pathtrie nodes	515	515	515	126



(a) Patterns without wildcards (b) Patterns with wildcards (50%)

Fig. 8. Running time comparison for filtering single primitive patterns

test sets of XPath patterns by using pathgenerator[3] with DTDs of XMark and dblp.

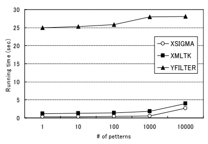
In what follows, we assume that a *wildcard* means // or *, and that *patterns with wildcards (c%)* mean that their occurrence probabilities of // and * are both $c\%$ ($0 \leq c \leq 100$). We call patterns with wildcards (0%) as *patterns without wildcards*. We also define *simple patterns* and *complex patterns* as XPath patterns with at most one predicate and ones with more than two predicates, respectively.

4.1 Comparing XSIGMA with Other XML Filtering Engines

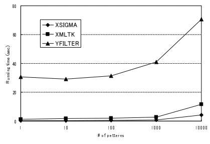
First, we compared the performance of XSIGMA with those of yfilter [8] and XMLTK [2] in filtering a single primitive pattern. Fig. 8 shows the running times of those engines over XMark1 data for 30 primitive patterns with/without wildcards. XSIGMA runs about 3.4 times faster than XMLTK and about 70 times faster than yfilter.

Then, we studied performance of XSIGMA for filtering a set of primitive patterns. In Fig. 9 and 10, we show the running time and the memory usage of XSIGMA, yfilter, and XMLTK against sets of primitive patterns of size $1, 10, \ldots, 10000$. From the results, XSIGMA runs faster than yfilter and XMLTK against large sets of primitive patterns while it uses more computing space than XMLTK. This tradeoff between XSIGMA and XMLTK comes from the difference of their strategies for matching primitive patterns using DFA. XSIGMA adopts

[3] http://yfilter.cs.berkeley.edu/code_release.htm

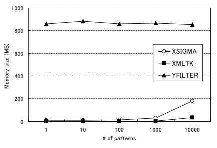


(a) Patterns without wildcards

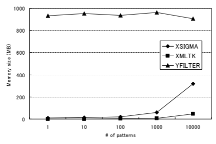


(b) Patterns with wildcards (50%)

Fig. 9. Running time comparison of XSIGMA against a set of primitive patterns



(a) Patterns without wildcards

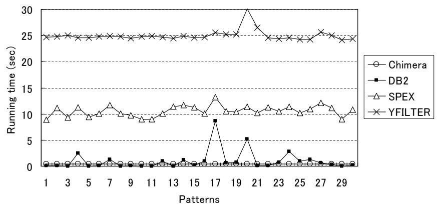


(b) Patterns with wildcards (50%)

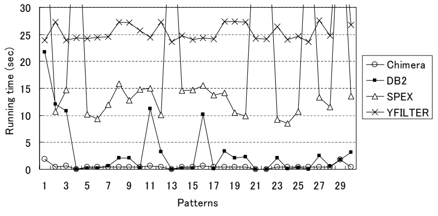
Fig. 10. Space comparison of XSIGMA against a set of primitive patterns

Table 2. Import time comparison

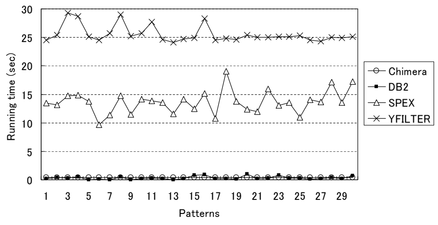
Running time (sec)	XMark1	XMark5	XMark10	dblp
Chimera	4	23	43	8
DB2	30	732	932	40



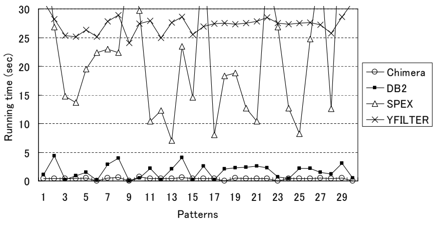
(a) Simple patterns without wildcards



(b) Simple patterns with wildcards (50%)



(c) Complex patterns without wildcards



(d) Complex patterns with wildcards (50%)

Fig. 11. Running time comparison for querying single XPath patterns on XMark

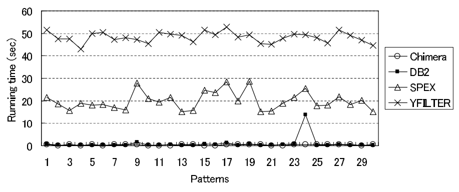
the DFA matching primitive patterns directly for faster processing and XMLTK adopts the Lazy-DFA technique for using less memory.

4.2 Comparing Chimera with Other XPath Querying Engines and a Commercial RDB

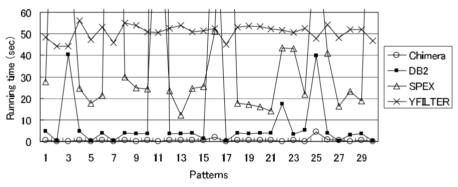
We also compared Chimera to yfilter, SPEX [11], and IBM's DB2 (Enterprise Server Edition Version 9.7). Note that XMLTK does not allow both simple and complex patterns completely. Table 2 compares the import time of Chimera and DB2, where the import time of Chimera we measured is the time for pathtrie construction and binary XML data transformation. Chimera is 5-30 times faster than DB2.

Then, we executed performance comparison for querying the four kinds of single XPath patterns, which are (a) simple patterns without wildcards, (b) simple patterns with wildcards, (c) complex patterns without wildcards, and (d) complex patterns with wildcards. Fig. 11 and 12 show the results against each kind of single patterns on XMark1 and dblp, respectively. From the results, Chimera runs fast and stably against each kind of patterns. DB2 also runs fast while its performance sometimes becomes worse especially in case of querying a pattern with wildcards. This shows the potential of stream-oriented processing for large XML data stored.

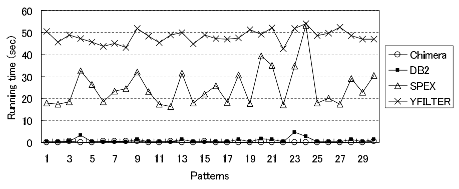
Finally, we examined the scalability of Chimera. In Fig. 13, we show the running time of Chimera, yfilter, and DB2 for querying a set of patterns of sizes 100, 1000, . . . , 5000 on XMark1 and dblp data. In Fig. 14, we also show the running time of Chimera on XMark1, XMark5, and XMark10 data against a set



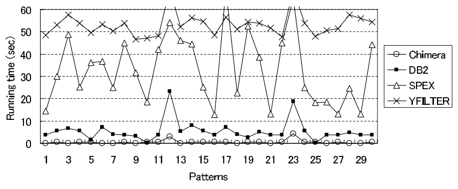
(a) Simple patterns without wildcards



(b) Simple patterns with wildcards (50%)

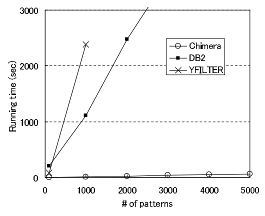


(c) Complex patterns without wildcards

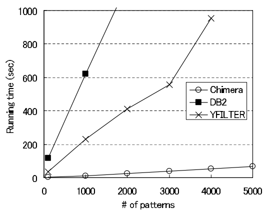


(d) Complex patterns with wildcards (50%)

Fig. 12. Running time comparison for querying single XPath patterns on dblp



(a) XMark1



(b) dblp

Fig. 13. Running time comparison for querying a set of XPath patterns

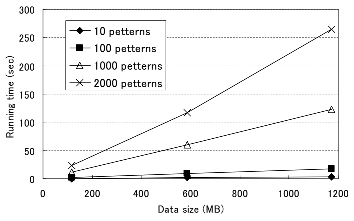


Fig. 14. Scalability of Chimera

of patterns of sizes 10, 100, 1000, 2000. The pattern sets used in those experiments include both simple and complex patterns with wildcards (1%). From the results, we can conclude that Chimera runs fast in real time and scales against the size of both data and patterns.

5 Conclusion

In this paper, we proposed an XPath filtering/querying engine Chimera. Based on an efficient matching engine XSIGMA for primitive patterns and a twig evaluator over event streams, Chimera runs fast and stably against a large set of XPath patterns in Univariate XPath. The experimental results show that Chimera runs fast and stably for any XPath patterns and scales against thousands of XPath patterns without the need for any heavy pre-processing techniques.

References

1. Aho, A.V., Corasick, M.: Efficient String Matching: An Aid to Bibliographic Search. Comm. ACM 18(6), 333–340 (1975)
2. Avila-Campillo, I., Green, T.J., Gupta, A., Onizuka, M., Raven, D., Suciu, D.: XMLTK: An XML toolkit for scalable XML processing. In: Proc. PLANX 2002 (2002)
3. Bar-Yossef, Z., Fontoura, M., Josifovski, V.: On the Memory Requirements of XPath Evaluation over XML Streams. Journal of Computer and System Sciences 73(3), 391–441 (2007)
4. Boag, S., Chamberlin, D., Ferandez, M.F., Florescu, D., Robie, J., Simeon, J.: XQuery 1.0: An XML Query Language. W3C (2003), http://www.w3.org/TR/xquery
5. Chen, Y., Davidson, S., Zheng, Y.: An Efficient XPath Query Processor for XML Streams. In: Proc. ICDE 2006 (2006)
6. Clerk, J.: XML Transformations (XSLT) Version 1.0. W3C (1999), http://www.w3.org/TR/xslt
7. Clerk, J., DeRose, R.: XML Path Language (XPath) Version 1.0. W3C (1999), http://www.w3.org/TR/xpath

8. Diao, Y., Altinel, H., Franklin, M.J., Zhang, H., Fischer, P.M.: Path Sharing and Predicate Evaluation for High-performance XPath Filtering. In: Proc. ACMTOD (2003)
9. Gou, G., Chirkova, R.: Efficient Algorithms for Evaluating XPath over Streams. In: Proc. SIGMOD 2007, pp. 269–280 (2007)
10. Mitarai, S., Ishino, A., Takeda, M.: Light-weight Acceleration for Streaming XML Document Filtering. In: Proc. SWOD 2007, pp. 37–42 (2007)
11. Olteanu, D.: SPEX: Streamed and Progressive Evaluation of XPath. TKDE 19(7), 934–949 (2007)
12. Olteanu, D., Meuss, M., Furche, T., Bry, F.: XPath: Looking Forward. In: Chaudhri, A.B., Unland, R., Djeraba, C., Lindner, W. (eds.) EDBT 2002. LNCS, vol. 2490, pp. 109–127. Springer, Heidelberg (2002)
13. Qin, L., Yu, J.X., Ding, B.: TwigList: Make Twig Pattern Matching Fast. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 850–862. Springer, Heidelberg (2007)

Privacy and Anonymization as a Service: PASS[*]

(Demonstration proposal)

Ghasem Heyrani-Nobari[1], Omar Boucelma[2], and Stéphane Bressan[1]

[1] School of Computing, National University of Singapore
ghasem@comp.nus.edu.sg, steph@nus.edu.sg
[2] LSIS, Aix-Marseille University
omar.boucelma@univ-cezanne.fr

Abstract. The Internet and the World Wide Web democratized the means to publish and share corporate and personal data. Many anecdotes occurred over the last decades that well illustrate the danger for privacy and confidentiality. The advent of Cloud computing infrastructures is likely, if successful, to further encourage this trend. The analysis, diagnosis and prevention of privacy risk within a Cloud computing infrastructure are therefore important services to provide to users. In recent years, several algorithms such as K-anonymity, L-diversity and Anatomy, have been proposed to address the issue of data anonymization and diversification. They transform original data sets into modified data sets ensuring some privacy while minimizing the information loss incurred during the transformation. Shared and published data can remain meaningful without jeopardizing privacy.

We propose an integrated collection of privacy management services together with an interface to orchestrate their execution and assess their evaluation. The system consists of Web services and Cloud architecture. Cloud users can explore and apply privacy management services as Cloud services. This proposal is a first but significant step towards the general concept of a Cloud of data services and data transformation processes for data privacy, anonymity, security, quality, mining, management, publishing and sharing of data.

Keywords: web services, data privacy, anonymization, online database tools, data quality, cloud services.

1 Introduction

Organizations are producing larger and larger amounts of electronic information. It is often a necessity --for various reasons related to business and marketing practices -- to share and publish data. These organizations need however to make sure that sensitive information is not disclosed. They need to often resort to ad hoc transformations with little guarantee of privacy. In [3] such risk has been strikingly illustrated: the medical

[*] This project is partially supported by a university research grant R-252-000-328-112 and SERC Grant 0421120028.

record of the governor of Massachusetts has been identified among publicly available medical data.

The problem is further exacerbated by the heterogeneity of hardware, software and data. Cloud computing among its various objectives also aims at offering to users a flexible solution to the problem of interoperability by providing a unique integration platform: the Cloud itself.

The system we propose to demonstrate, PASS, advocates the concept of a Cloud and service oriented solution for the management of data privacy. It consists of a (i) web based user interface for the analysis, diagnosis [2] and elimination of the privacy risk and the exploration of the possible solutions, and (ii) web services for the realization of an integrated solution.

PASS offers a range of data anonymization and diversification algorithms in a one stop integrated environment together with all the utility services to sample, test, upload, process and save data to be published.

2 Designing Data Processes

Processing in PASS is organized in projects. Each project is a collection of objects: datasets (imported data, remote data sources and results) and processes that apply to the data. Objects can be shared among several projects.

The processes can be simple (the application of a single algorithm) or complex (a workflow). Processes can be executed stepwise in order to allow exploration and debugging.

Consider this as a simple usage scenario. A user registers the connection to her remote database. She creates the process for investigating the anonymization of her data. She creates a new process to do so. She decides to import a sample of the data using the sampling service. She wants to try the Anatomy Algorithm on her sample dataset. She specifies the schema and the sensitive attributes. She can customize the ontology for non numeric attributes generalization. Once the process is designed, she can execute it. The results are represented as two new Datasets in the current project folder.

The user can further check whether the output is a 3-Anonymized dataset or decide to change the process parameters and run it again. Once satisfied, she can run the same process on her entire data set and export the results as an XML document to her DB2 Server.

3 System Architecture

The main design goal of PASS is to make the data processing as flexible and simple as possible while providing efficiency. PASS consists of three main groups of Services: data services, user services and control services (see Fig 1). For implementation purposes, we used Microsoft .Net Framework, together with WCF (Windows Communication Foundation) [9] which supports both synchronous and asynchronous communications. Services can be invoked through different standards and protocols such as SOAP and REST.

Data services can be used for various data processing jobs such as anonymization, privatization and sampling methods.

User services offer authentication, authorization and accounting together with VPN, SSH or SSL connections.

Control services are responsible for system stability, accessibility and extendibility. For instance, they allow tracing and monitoring, and the control of resources or running processes.

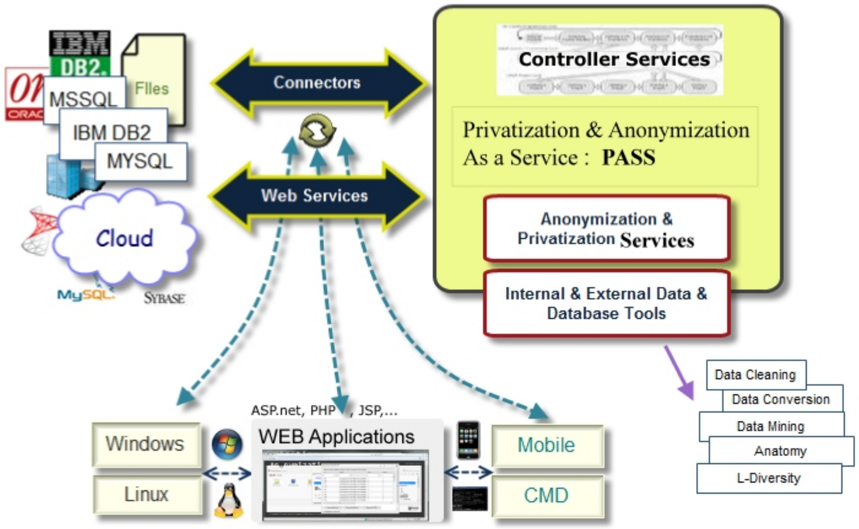


Fig. 1. Privacy & Anonymization as a Service (PASS)

Web services can be used interactively through an interface or programmatically to develop new applications in a variety of environment and on diverse platforms.

The interactive interface is a Web-based platform. It follows the design concept of a cross browser WebOS based on Web 2.0 standards. PASS users recognize a familiar and flexible working environment similar to a process oriented virtual desktop. We used several clients and server side techniques and libraries such as Ajax [8], JQuery [6] and JSON [7].

The current implementation proposes the following anonymization services: k-Anonymization [3], Anatomy [5], Anonymizing sequential releases [4] and Entropy L-Diversity [1].

3.1 Connectivity

In PASS there are several ways to initialize the datasets, which can be from local CSV files, local relational tables stored in the local database (populated manually for testing or with imported data) to remote data sources (external database servers, web services, streams or cloud datasets). The standard protocols of PASS (reference) allow connectivity to a large variety of local and remote sources and formats.

Similarly, PASS services are easily added as they are integrated using a standard web service interface. Although they are local to the PASS server in the current architecture and implementation, the design and implementation cater for both distributed and remote services.

4 Conclusion

The system we propose to demonstrate is a proof of concept for a Cloud-based and service oriented approach to not only data privacy diagnosis and protection in particular but also to a workflow-oriented approach to the management of data in the cloud. It is made possible by the convergence of many technologies and standards: Web services, business process description and execution languages, Web 2.0 technologies. The system is easily demonstrated thanks to a simple yet complete Web-based interface.

References

1. Machanavajjhala, Gehrke, J., Kifer, D.: l-diversity: Privacy beyond k-anonymity. In: ICDE (2006)
2. Mirakabad, M.R.Z., Jantan, A., Bressan, S.: Towards a Privacy Diagnosis Centre: Measuring k-Anonymity. In: International Symposium on Computer Science and its Applications, CSA (2008)
3. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-based Systems, 557–570 (2002)
4. Wang, K., Fung, B.: Anonymizing sequential releases. In: SIGKDD (2006)
5. Xiao, X., Tao, Y.: Anatomy: Simple and effective privacy preservation. In: VLDB (2006)
6. JQuery new JavaScript library, http://jquery.com
7. JSON (JavaScript Object Notation), http://www.json.org
8. AJAX (Asynchronous JavaScript and XML), http://www.w3.org/TR/XMLHttpRequest/
9. WCF (Windows Communication Foundation), http://msdn.microsoft.com/en-us/netframework/aa663324.aspx

Visual Evaluation of Outlier Detection Models

Elke Achtert, Hans-Peter Kriegel, Lisa Reichert,
Erich Schubert, Remigius Wojdanowski, and Arthur Zimek

Institut für Informatik, Ludwig-Maximilians-Universität München
Oettingenstr. 67, 80538 München, Germany
http://www.dbs.ifi.lmu.de
{achtert,kriegel,reichert,schube,wojdanowski,zimek}@dbs.ifi.lmu.de

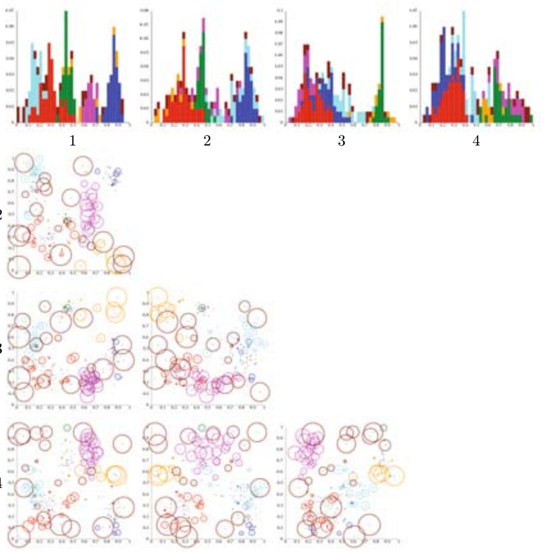
Abstract. Many outlier detection methods do not merely provide the decision for a single data object being or not being an outlier. Instead, many approaches give an "outlier score" or "outlier factor" indicating "how much" the respective data object is an outlier. Such outlier scores differ widely in their range, contrast, and expressiveness between different outlier models. Even for one and the same outlier model, the same score can indicate a different degree of "outlierness" in different data sets or regions of different characteristics in one data set. Here, we demonstrate a visualization tool based on a unification of outlier scores that allows to compare and evaluate outlier scores visually even for high dimensional data.

1 Introduction

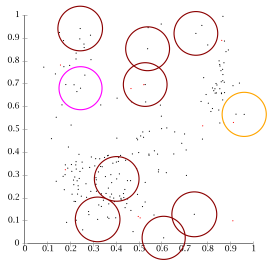
Finding outliers that do not fit well to the general data distribution is very important in many practical applications, including e.g. credit card abuse detection in financial transactions data, the identification of measurement errors in scientific data, or the analysis of sports statistics data. Outlier detection can be seen as not merely interested in removing noise but also in finding interesting database objects deviating in their behavior considerably from the majority and, as such, providing new insights. Indeed, both aspects of outlier detection are like two sides of a medal as one person's noise may be another person's signal. The application scenarios given above highlight both interests in outliers, as measurement errors in scientific data should possibly just be removed while a case of credit card abuse is the solely interesting fact among a wealth of usual data. As different as the questions that should be answered by outlier detection methods, as different are the approaches to the problem of identifying outliers. Aside from different properties of outlier detection methods, i.e., global vs. local, scoring vs. labeling, supervised vs. unsupervised, there is a huge variety of intuitions or techniques for how to decide on the "outlierness" of a database object. Examples of such intuitions are the original statistical approach [4], the distance based notion of outlierness [6,12,3,11], the density based approach [5,10] along with various adaptations to different scenarios [13,8,7], or the angle-based definition of outliers [9]. Since many approaches providing outlier scores are based on quite different assumptions, intuitions, and models, they naturally also differ substantially in the scaling, range, and meaning of values. In some cases, high values of an outlier score mean, the corresponding database object is *not at all* an outlier, in other cases, a higher value indicates more "outlierness". In some cases, the minimum occurring outlier score is around 1, indicating that the corresponding database object perfectly fits to the data distribution, in other cases, 1 is the maximum value, indicating the database object is an outlier as much as possible. For many methods, the scaling of occurring values of the outlier score even differs within the same method from data set to data set, i.e., outlier score x in one data set means, we have an outlier, in another data set the very same score x is not extraordinary at all. In many cases, even within one and the same data set, the identical outlier score x for two different database objects can denote substantially different degrees of outlierness, depending on different local data distributions around the two objects. Hence a major problem for any user not very acquainted with the outlier detection method in question is how to interpret the "factor" provided in order to decide whether or not the data object actually is an outlier and how to compare the outlier score provided by one outlier model to the outlier score provided by another model. Furthermore, since different models are based on different assumptions and intuitions, they are differently well suited for different data sets. Their results, however, are in many cases very difficult to compare. Aside from the scaling issue, opposite decisions on the outlierness of single objects may be equally meaningful since there is no generally valid definition of what constitutes an outlier in the first place. For different application scenarios, a different selection of outlier detection approaches may be meaningful.

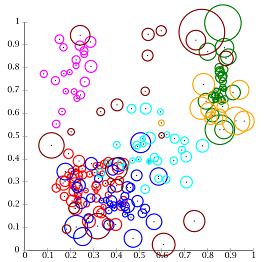
2 Demonstration

Here, we present a visualization tool that is able to visualize high dimensional data sets in 2D projections and mark the outlier score assigned by different methods to each object. The contrast of the outlier scores within one data set can also be visualized. In our framework, we implemented a range of different outlier detection models (examples have been given above) in a way that allows to freely combine the outlier detection method with different distance measures, where applicable, and therefore, e.g., to apply one implementation to different data types or to learn about the impact of the chosen distance function. Aside from standard distance functions like L_p norms or the cosine distance, there are also distance functions available specialized e.g. on time series data. Furthermore, the outlier detection algorithm can be backed by various index-structures provided by the framework. In Figure 1(a), a 4-dimensional data set is visualized in thumbnails for all axis-parallel two-dimensional projections. The color of each bubble shows the color of the corresponding cluster or noise. The data set consists of six clusters (colors red, blue, green, orange, cyan, and magenta), and some outliers (dark red). The top row shows stacked histograms of the distribution of values for each of the four dimensions, colored according to the respective



(a) Thumbnails of LoOP [7] scores for all 2-d projections (top row: one-dimensional distributions) of a 4-dimensional data set.





(b) 2-d projections with Reference Point Outlier [11] (top) and LDOF [13] (bottom) scores annotated.

Fig. 1. Some visualization features

cluster. Figure 1(b) shows the visualization of the outlier scores of two different methods for one data set in the same selected two-dimensional projection in detail. This visualization can be saved in various filetypes such as JPEG, PNG, EPS, PDF, or SVG. The latter is the original internal representation. Hence, the tool also allows for scale free zoom-in or zoom-out to inspect a specific region of the data set in more detail. The radius of a bubble reflects the magnitude of the corresponding outlier factor. This visualization requires a (non-trivial) scaling of the outlier scores for all available methods into a comparable range. We chose the range $[0, 1]$ which is provided already by some methods and can be thought of reflecting the probability of being an outlier. For all methods implemented within our framework and scaling differently, we provide adapters for the desired comparative scaling. This architecture is easily extendable for incorporation of new or different outlier detection methods. To assess how remarkable a specific outlier score is in a given data set, our tool also shows the contrast of outlier scores assigned by a specific model for a complete data set. If there is a low contrast of outlier scores, it is clear that the decision to select the top k outliers is rather arbitrary. Instead one could equally well select the top $k - 5$ or top $k + 7$ data objects as outliers. On the other hand, if there is a clear separation between top scores and lower scores, the user gets a valuable information on a possibly meaningful cut-off outlier score to define "real" outliers. Nevertheless, aside from the comparison of the absolute values of outlier scores, many approaches just aim at a concise ranking as only the top-k outliers are interesting. Hence, it is also possible to visualize just the top-k outliers (as shown in Figure 1(b), top, for the reference point approach [11], where also the chosen reference points are shown in red).

3 Conclusion

In summary, the contributions of the demonstrated software are (i) the possible application of a unifying scaling on arbitrary outlier scores; (ii) an intuitive visualization of (top-k) outlier scores on data of real vector spaces of arbitrary dimensionality; (iii) an illustration of the contrast of the outlier scores suitable as user guidance to select an appropriate threshold for final decision on outlierness; (iv) a flexible implementation of a broad selection of outlier detection methods that allows for usage of different distance measures and can be backed by various index structures.

Via `http://www.dbs.ifi.lmu.de/research/KDD/ELKI/`, the demonstrated software is available as release 0.3 of the ELKI framework [2,1].

References

1. Achtert, E., Bernecker, T., Kriegel, H.P., Schubert, E., Zimek, A.: ELKI in time: ELKI 0.2 for the performance evaluation of distance measures for time series. In: Mamoulis, N., Seidl, T., Pedersen, T.B., Torp, K., Assent, I. (eds.) SSTD 2009. LNCS, vol. 5644, pp. 436–440. Springer, Heidelberg (2009)
2. Achtert, E., Kriegel, H.P., Zimek, A.: ELKI: a software system for evaluation of subspace clustering algorithms. In: Ludäscher, B., Mamoulis, N. (eds.) SSDBM 2008. LNCS, vol. 5069, pp. 580–585. Springer, Heidelberg (2008)
3. Angiulli, F., Pizzuti, C.: Fast outlier detection in high dimensional spaces. In: Elomaa, T., Mannila, H., Toivonen, H. (eds.) PKDD 2002. LNCS (LNAI), vol. 2431, p. 15. Springer, Heidelberg (2002)
4. Barnett, V., Lewis, T.: Outliers in Statistical Data, 3rd edn. John Wiley & Sons, Chichester (1994)
5. Breunig, M.M., Kriegel, H.P., Ng, R., Sander, J.: LOF: Identifying density-based local outliers. In: Proc. SIGMOD (2000)
6. Knorr, E.M., Ng, R.T.: Algorithms for mining distance-based outliers in large datasets. In: Proc. VLDB (1998)
7. Kriegel, H.P., Kröger, P., Schubert, E., Zimek, A.: LoOP: local outlier probabilities. In: Proc. CIKM (2009)
8. Kriegel, H.P., Kröger, P., Schubert, E., Zimek, A.: Outlier detection in axis-parallel subspaces of high dimensional data. In: Proc. PAKDD (2009)
9. Kriegel, H.P., Schubert, M., Zimek, A.: Angle-based outlier detection in high-dimensional data. In: Proc. KDD (2008)
10. Papadimitriou, S., Kitagawa, H., Gibbons, P., Faloutsos, C.: LOCI: Fast outlier detection using the local correlation integral. In: Proc. ICDE (2003)
11. Pei, Y., Zaïane, O., Gao, Y.: An efficient reference-based approach to outlier detection in large datasets. In: Proc. ICDM (2006)
12. Ramaswamy, S., Rastogi, R., Shim, K.: Efficient algorithms for mining outliers from large data sets. In: Proc. SIGMOD (2000)
13. Zhang, K., Hutter, M., Jin, H.: A new local distance-based outlier detection approach for scattered real-world data. In: Proc. PAKDD (2009)

ADERIS: Adaptively Integrating RDF Data from SPARQL Endpoints

Steven Lynden, Isao Kojima, Akiyoshi Matono, and Yusuke Tanimura

Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Japan
{steven.lynden,a.matono,yusuke.tanimura}@aist.go.jp,
kojima@ni.aist.go.jp

Abstract. This paper describes the Adaptive Distributed Endpoint RDF Integration System (ADERIS), an adaptive, distributed query processor for integrating RDF data from multiple data resources supporting the SPARQL query language and protocol. The system allows a user to issue a federated query without any knowledge of the data contained in each endpoint and without specifying details of how the query should be executed. ADERIS relies on very limited information about each RDF data source to construct SPARQL source queries, the results of which are used to construct RDF predicate tables, which are integrated using pipelined index nested loop joins, the number and order of which may vary during query execution in order to reduce response time.

Keywords: Adaptive query processing, RDF query processing, distributed data integration.

1 Introduction

SPARQL [1], a W3C recommendation query language for RDF, is a widely used method of querying RDF data. Primarily, the SPARQL language allows sets of triple patterns to be matched against RDF graphs, supported by various features such as conjunctions, disjunctions, filter expressions, optional triple patterns and multiple representations of query results. The SPARQL query language has an associated protocol with HTTP and SOAP bindings, and an XML-based results format, which complement each other to promote interoperability, allowing clients to interact with RDF data sources without having to deal with syntactic heterogeneities between different infrastructures. This approach is used by many large repositories, such as DBPedia [2]. Federated queries across multiple SPARQL endpoints allow data from such endpoints to be integrated, and is also potentially beneficial in heterogeneous information systems where individual components use SPARQL wrappers to expose data. Data integration in this context is particularly appealing when bearing in mind the progress made by the Linked Open Data Project [3], which aims to promote the widespread usage of URI-based representations to allow RDF terms to be consistently defined. Where consistently defined RDF terms are used across multiple repositories, data can be joined together to answer federated SPARQL queries across multiple RDF repositories.

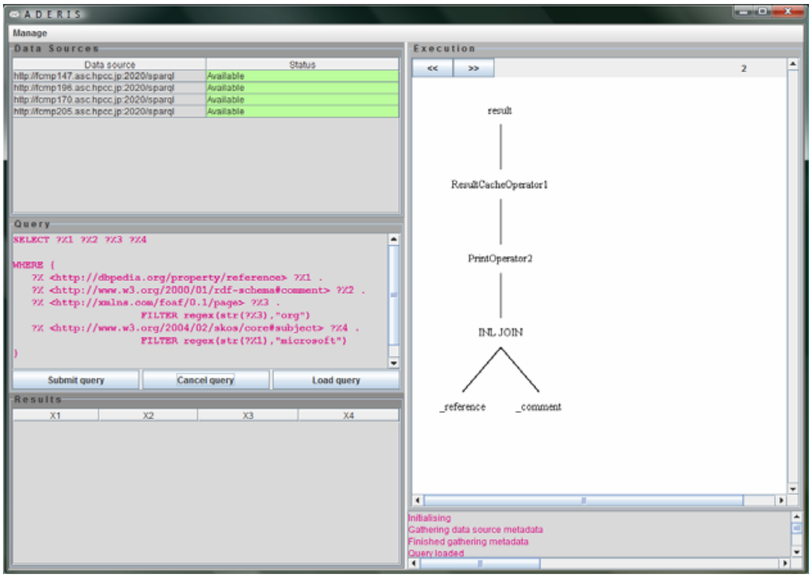
Various systems exist supporting federated queries across multiple RDF data sources, such as ARQ [4], the SPARQL query processor from the Jena Semantic Web Framework, which possesses capabilities for executing remote queries and is complemented by extensions for query optimisation in the form of the Distributed ARQ (DARQ) [5] system. SemWIQ [6] is another system based on ARQ for optimising and executing distributed RDF queries. These systems require accurate statistics about data sources in order to effectively optimise query plans and cannot alter their query plans during execution if performance is sub-optimal. In contrast, an assumption made in the design of ADERIS is that many RDF endpoints are made available without statistics about the data contained within them and gathering such information is difficult. Endpoints may be constantly updated and therefore behave unpredictably in terms of response time due to computational load and the effects of network communication times, therefore ADERIS aims to rely as little as possible on statistics about endpoints and avoids generating static query plans. Query processing is done in an almost entirely adaptive fashion, joining data as it becomes available and modifying join order based on selectivity information gathered at runtime.

2 System Details

Prior to issuing federated queries, the set of SPARQL endpoints over which queries are executed are registered with the system, which involves the user supplying the system with a URI for each of the endpoints. ADERIS performs the following steps during query execution:

1. **Issue Source Queries.** If available, metadata (e.g. presence/absence of terms, indexes etc) about RDF endpoints are used to generate a source query for each SPARQL endpoint registered with the system. A source query is a SPARQL query issued to an endpoint in order to fetch triples that may be needed to answer the query. The objective of source query generation is to generate queries that minimise the number of triples returned while still retrieving all triples that may be needed to execute the federated query. ADERIS employs various schemes to achieve this that can function with various kinds of metadata/statistics. The most basic scheme relies on extremely limited information, specifically the set of predicate terms contained in a data source, which can be obtained from an endpoint by issuing a simple SPARQL query when the endpoint is registered with the system.

2. **Construct Predicate Tables.** The results from each source query are used to construct a set of predicate tables, where a two-column table exists for each predicate returned as a result from a source query and each row of the predicate table represents a triple by storing the corresponding subject/object values. This vertically partitioned approach has been shown to be effective when processing RDF data in [7]; here it provides a set of tables, constructed by the source queries, which must be joined to answer the query. The mapping between the source queries and predicate tables is usually one-to-many (or in rare cases one-to-one) and the metadata used in (1) to generate source queries is used to determine this mapping, which is used in the next step to determine



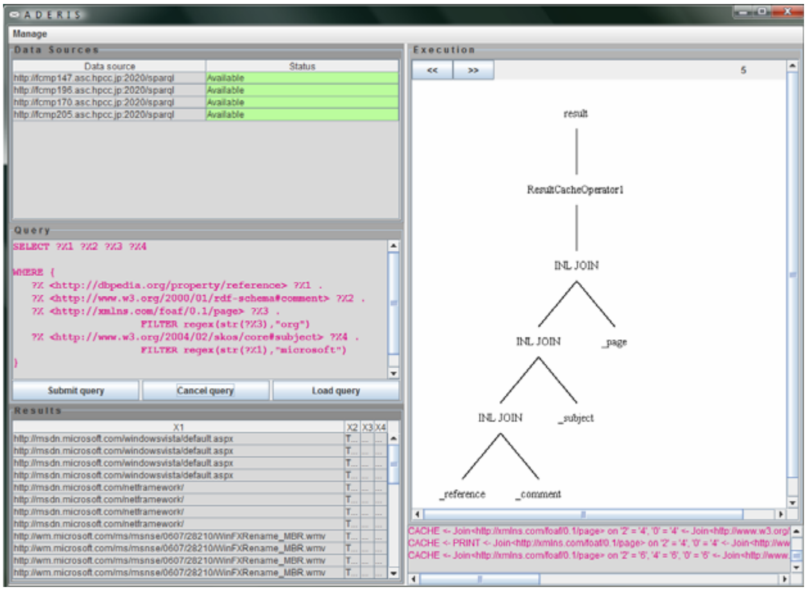
Data source	Status
http://fcmp147.asc.hpcc.jp:2020/sparql	Available
http://fcmp196.asc.hpcc.jp:2020/sparql	Available
http://fcmp170.asc.hpcc.jp:2020/sparql	Available
http://fcmp205.asc.hpcc.jp:2020/sparql	Available

```
SELECT ?X1 ?X2 ?X3 ?X4

WHERE {
   ?X <http://dbpedia.org/property/reference> ?X1 .
   ?X <http://www.w3.org/2000/01/rdf-schema#comment> ?X2 .
   ?X <http://xmlns.com/foaf/0.1/page> ?X3 .
                  FILTER regex(str(?X3),"org")
   ?X <http://www.w3.org/2004/02/skos/core#subject> ?X4 .
                  FILTER regex(str(?X1),"microsoft")
}
```

X1	X2	X3	X4

```
Initialising
Gathering data source metadata
Finished gathering metadata
Query loaded
```



Data source	Status
http://fcmp147.asc.hpcc.jp:2020/sparql	Available
http://fcmp196.asc.hpcc.jp:2020/sparql	Available
http://fcmp170.asc.hpcc.jp:2020/sparql	Available
http://fcmp205.asc.hpcc.jp:2020/sparql	Available

```
SELECT ?X1 ?X2 ?X3 ?X4

WHERE {
   ?X <http://dbpedia.org/property/reference> ?X1 .
   ?X <http://www.w3.org/2000/01/rdf-schema#comment> ?X2 .
   ?X <http://xmlns.com/foaf/0.1/page> ?X3 .
                  FILTER regex(str(?X3),"org")
   ?X <http://www.w3.org/2004/02/skos/core#subject> ?X4 .
                  FILTER regex(str(?X1),"microsoft")
}
```

X1	X2	X3	X4
http://msdn.microsoft.com/windowsvista/default.aspx	T...	...	...
http://msdn.microsoft.com/windowsvista/default.aspx	T...	...	...
http://msdn.microsoft.com/windowsvista/default.aspx	T...	...	...
http://msdn.microsoft.com/windowsvista/default.aspx	T...	...	...
http://msdn.microsoft.com/netframework/	T...	...	...
http://msdn.microsoft.com/netframework/	T...	...	...
http://msdn.microsoft.com/netframework/	T...	...	...
http://msdn.microsoft.com/netframework/	T...	...	...
http://wm.microsoft.com/ms/msnse/0607/28210/WinFXRename_MBR.wmv	T...	...	...
http://wm.microsoft.com/ms/msnse/0607/28210/WinFXRename_MBR.wmv	T...	...	...
http://wm.microsoft.com/ms/msnse/0607/28210/WinFXRename_MBR.wmv	T...	...	...
http://wm.microsoft.com/ms/msnse/0607/28210/WinFXRename_MBR.wmv	T...	...	...

```
CACHE <- Join<http://xmlns.com/foaf/0.1/page> on '2' = '4', '0' = '4' <- Join<http://www.w3.org/
CACHE <- PRINT <- Join<http://xmlns.com/foaf/0.1/page> on '2' = '4', '0' = '4' <- Join<http://ww
CACHE <- Join<http://xmlns.com/foaf/0.1/page> on '2' = '6', '4' = '6', '0' = '6' <- Join<http://www.
```

Fig. 1. The ADERIS client at two different points during the execution of a federated SPARQL query. Four endpoints, the URLs of which are listed in the top left frame under the heading "Data Sources", are queried. The panel on the right shows the query plan, initially (top) joining two predicate tables that have become available. The second (bottom) query plan has adapted to join two new tables and results have started to become available (shown in the bottom left frame). The join order can be modified further by the system if necessary.

when a predicate table is complete, i.e. the moment in time when all source queries that map to a given predicate table have returned all their results. Indexes are generated on potential join predicates to allow index nested loop joins to be used in the next step.

3. **Join Predicate Tables.** Predicate tables are inserted into the query plan when they become complete, using an extension of the technique described in [8], which is capable of reordering the joins without throwing away already generated results. Any other processing that needs to be done (evaluating FILTER expressions that couldn't be pushed down into source queries. etc.) is also done here in order to answer the query.

The main benefit of this approach is that it can adapt to different characteristics of the data obtained from remote data sources. The ADERIS interface, illustrated in Fig 1, allows the user to lookup and register SPARQL endpoints and issue federated queries. The query execution and its adaptations are highlighted by a real-time display of the join order as processing is performed. Query results are displayed in real-time as they are produced by the pipelined query plan. Although the system currently supports only a limited subset of the SPARQL query language, future work will increase the range of supported queries. Various applications of this work, where answering SPARQL queries over multiple RDF repositories can play an important role will also be investigated. The software prototype described in this paper forms the basis of an open source project, more information about which can be found at: http://dbgrid.org/FederatedSPARQL.

Acknowledgement
This work is supported by the Strategic Information and Communications R&D Promotion Programme (SCOPE) of the Japanese Ministry of Internal Affairs and Communications.

References

1. Prud'hommeaux, E., et al.: SPARQL Query Language for RDF, W3C Recommendation (January 2008), `http://www.w3.org/TR/rdf-sparql-query/`
2. DBPedia, `http://dbpedia.org`
3. Linked Data, `http://linkeddata.org`
4. ARQ - A SPARQL processor for Jena, `http://jena.sourceforge.net/ARQ/`
5. Quilitz, B., Leser, U.: Querying Distributed RDF Data Sources with SPARQL. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 524–538. Springer, Heidelberg (2008)
6. Langegger, A., Woss, A.: A Semantic Web Middleware for Virtual Data Integration on the Web. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 493–507. Springer, Heidelberg (2008)
7. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.: SW-Store: A Vertically Partitioned DBMS for Semantic Web data management. VLDB J. 18(2), 385–406 (2009)
8. Li, Q., Sha, M., Markl, V., Beyer, K., Colby, L., Lohman, G.: Adaptively Reordering Joins during Query Execution. In: Proc. ICDE, pp. 26–35. IEEE Computer Society, Los Alamitos (2007)

Outline of Community-Type Content Based on Wikipedia

Akiyo Nadamoto[1], Eiji Aramaki[2], Takeshi Abekawa[3], and Yohei Murakami[4]

[1] Konan University
Okamoto 8–9–1, Higashinada-ku, Kobe, 658–8501, Japan
nadamoto@konan-u.ac.jp
[2] The University of Tokyo
Hongou 7–3–1, Bunkyo-ku, Tokyo, 113–8655, Japan
eiji.aramaki@gmail.com
[3] National Institute Informatics
Hitotubashi 2–1–2, Chiyoda-ku, Tokyo, 101–8430, Japan
abekawa@nii.ac.jp
[4] National Institute of Information and Communications Technology
Hikaridai 3–5, Seika-cho, Soraku-gun, Kyoto, 619–0289, Japan
yohei@nict.go.jp

Abstract. It is difficult to understand the outline of community-type content such as Blog, Social Network Services(SNS), and Bulletin Board System(BBS) because multiple users post content freely. In this paper, we have developed a system that presents the outline of community-type content by using Wikipedia. We focus on the table of contents (TOC) collected from Wikipedia. Our system compares the comments in a thread with the information in the TOC obtained from Wikipedia and identifies contents that are similar. Thus, the user can understand the outline of community-type content when he/she views a table with similar contents.

1 Introduction

SNSs and blogs, which are maintained by a community of people, are popular Web 2.0 tools. We call the content of such Web 2.0 tools "community-type content." Community-type content, a representative example of Web 2.0 content, consists of multiple threads, and each thread consists of comments posted by multiple users. It is difficult to understand a thread in a community because users post comments freely, and their language is complicated. In community-type content, users sometimes enter into heated discussions, because of which they concentrate only on one issue and lose track of the actual theme. In this case, each user may want to know how relevant his/her point is to the discussion. Hence, we believe that it would be beneficial for users to understand the outline of their discussion. On the other hand, the contents of Wikipedia are posted by different users, whose policy is "Neutral point of view is the fundamental principle of Wikipedia [1]." Thus, we consider that Wikipedia contents are based

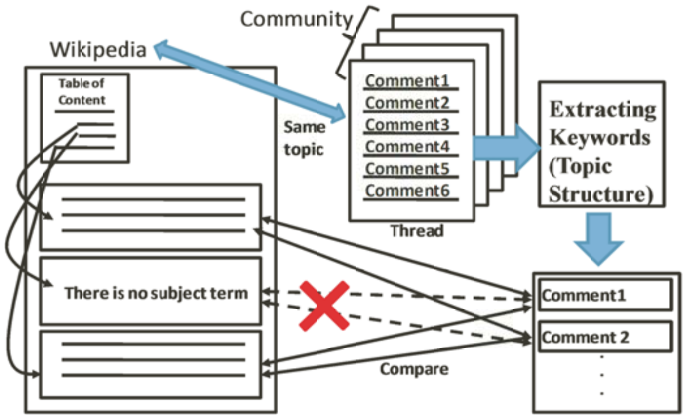


Fig. 1. Comparing in VOCC

on the general viewpoint of a given theme. In this paper, we propose a system that presents the outline of a thread in community-type content by comparing the comments in a thread with the Wikipedia content. In this case, we choose the title of an article from Wikipedia and compare each comment in a thread in the community with the smallest structure in the table of contents (TOC) from the article. (show Figure 1) We call this system "Viewing Outline of Community-type Content (VOCC)."

2 Outline of VOCC

Figure 2 shows user interface of VOCC. In Figure 2, the left-hand window in (a) shows the Wikipedia article that contains the keywords input by the user, and the left-hand window in (b) shows the results obtained with VOCC. We use the topic of the content structure list to identify the exact outline of a thread. Overview of our system:

1. A user inputs the theme of a thread that he/she wants to use as a keyword for comparison.
2. VOCC displays the target Wikipedia article on the left side of the window and the list of candidates in the community whose theme matches the user's input on the right side of the window(Figure 2(a)).
3. The user selects the community and thread he/she wants to compare.
4. A Wikipedia article and the thread in a community-type content are extracted on the basis of the keyword entered by a user. The Wikipedia article is called the "target article"; the thread in the community-type content , "target thread"; and the comment in the target thread , "target comment."
5. VOCC extracts the keywords that are based on topic structure[2] from a target thread.

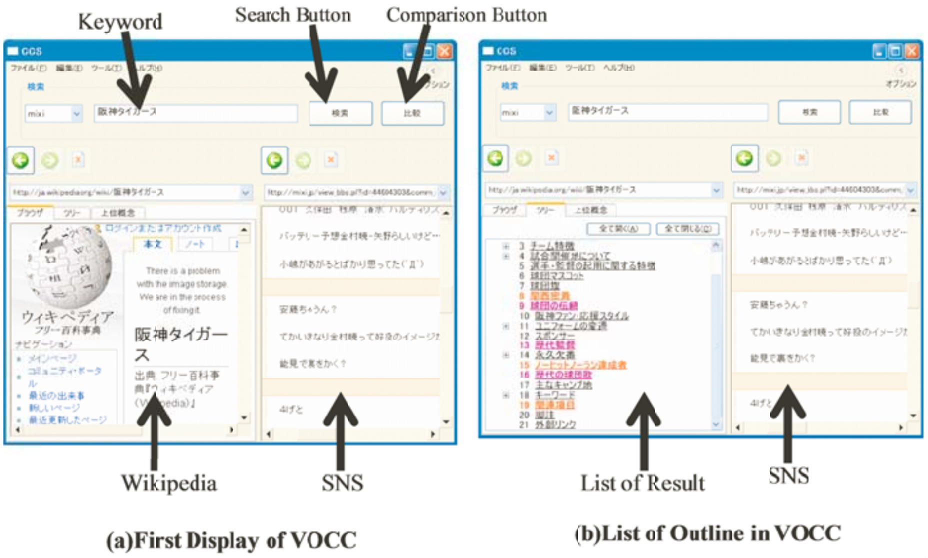


Fig. 2. Display of VOCC

6. VOCC then compares the target comment with each small passage in the target article on the basis of the TOC by using the topic structure and extracts the content discussed in the target thread.
7. In steps 5 and 6, VOCC calculates the number of comments in a target thread.
8. VOCC presents a similar title from Wikipedia as the outline of a thread (Figure 2(b)).

Since VOCC uses the tab browser, the user can browse both the target article and the results by simply clicking the tab.

3 Comparison between Threads and Articles

Target Passage of the article
In this study, we consider the topic of the target article to be the same as that in a thread (show Fig1). In other words, we identify the target article whose theme matches the keyword input by the user. After identifying the target article, we compare the comments in the thread with each small passage in the article. Each small passage in the article is classified on the basis of the TOC of the target article. In this case, not all the small passages in the article are regarded as target passages. If the topic of the community is "The Masters Tournament" and the target thread in the community discusses "Tiger Woods," since the target article refers to "Tiger Woods," the thread topic is "Tiger Woods in Masters." The article on "Tiger Woods" includes information other than that on "Tiger Woods

in Masters," and this information should be removed from the small passage in the target article. We consider the target passage with the highest target passage degree $DTaP_i$. $DTaP_i$ is determined as follows:

$$DTaP_i = \sum_{j=1}^{m} w_{s_j} + w_{c_t} > \beta$$

where w_{s_j} is the weight of the subject term s_j, and w_{c_t} is the weight of the topic of the community. These values are calculated by using tf/idf.

Comparison between comments in the target thread and the target passage of the article
First, we calculate the weight of each subject term and content term by using tf/idf. Then, we compare the comment in a thread CP_i with the target passage TaP_j of an article by using a cosine vector in the following manner:

$$Sim(CP_i, TaP_j) = \frac{\overrightarrow{F}(CP_i) \bullet \overrightarrow{F}(TaP_j)}{|\overrightarrow{F}(CP_i)| \bullet |\overrightarrow{F}(TaP_j)|} < \gamma$$

where $\overrightarrow{F}(CP_i)$ is a feature vector of CP_i and $\overrightarrow{F}(TaP_j)$ is a feature vector of TaP_j. If $Sim(CP_i, TaP_j)$ is greater than the threshold γ, the target passage in the article is the outline of a comment.

4 Conclusion

We have developed a method called "VOCC" for presenting the outline of community-type content by using Wikipedia. Our system compares the comments in a thread with the information in the TOC obtained from Wikipedia and identifies contents that are similar. Thus, the user can understand the outline of community-type content when he/she views a table with similar contents.

Acknowledgments. A part of this study was supported by a Grant-in-Aid for the Information Explosion Project (Number: 21013044).

References

1. NPOF of Wikipedia, http://en.wikipedia.org/wiki/NPOV
2. Nadamoto, A., Qiang, M., Tanaka, K.: B-CWB: Bilingual Comparative Web Browser Based on Content-Synchronization and Viewpoint Retrieval. World Wide Web Journal (Online) ISSN: 1573-1413

BISCAY: Extracting Riding Context from Bike Ride Data

Keiji Sugo[1], Manabu Miyazaki[1], Shin'ichi Konomi[1], Masayuki Iwai[2], and Yoshito Tobe[1]

[1] Department of Information Systems and Multimedia Design, Tokyo Denki University
2-2 Kanda-Nishiki-Cho, Chiyoda-ku, Tokyo, Japan
[2] Institute of Industrial Science, the University of Tokyo
4-6-1 Komaba, Meguro-ku, Tokyo, Japan

Abstract. Recently, global warming is a serious problem all over the world. Japan endeavors to promote using bicycles in order to protect environment. We developed a system for supporting cyclists by using the data from a bike sensor network. Cyclists lose comfort of riding when they need to go through crowded streets. We collect riding information obtained with a gyro sensor attached to handlebars of a bicycle. Based on such information, we then extract riding condition of cyclists. This paper presents the design and implementation of a Human Probe system that can collect and store sensor data to show comfortable cycling routes on a map.

Keywords: Bikenet, sensor network, Human Probe, gyro sensors.

1 Introduction

There is an increasing number of people who ride bicycles to save fuel resources, reduce the emission of carbon dioxide, and live healthily. However, cyclists sometimes need to use the lane for cars or sidewalks when there are no dedicated bike paths. If cyclists use sidewalks, they would feel uncomfortable, and make pedestrians feel uncomfortable as well. Thus, it is a challenge for a bike rider to find suitable roads in an urban area. To solve the above issue, we propose BISCAY which calculates comfort indicators of cycling roads. In BISCAY, we attach a three-axis gyro sensor and a GPS receiver to a bike, which monitors a hand operation and braking during a bike ride. The comfort is calculated using the information emitted by the gyro sensor. Bikenet[1] is the system which can gather environmental information on a route. However they do not focus on the comfort of riders. We detect the movement of handles to measure the comfort of riders. Furthermore, BISCAY enables the cyclists to share comfort indicators on the Web map.

In this paper we first define the context indicators for riding a bike. Secondly we describe an algorithm for calculating comfort indicators for cyclists. Finally, we show the details of our demonstration.

2 System Architecture

In a BISCAY bicycle, a three-axis gyro sensor and a GPS receiver are attached to the center of a handle and near the rear reflector, respectively, as shown in Fig.1. The three-axis Gyro sensor measures Yaw, Pitch and Roll in a raw data format. Here, the three physical quantities mean the following:

@@Yaw: degree of handle
@@Pitch: road grade
@@Roll: degree of carving of bike
@@GPS: latitude, longitude and speed of bike

Each data item is acquired in a 100-ms interval to analyze it with a sliding window. The length of the sliding window is 3.2-ms. In addition, we define a data set for analysis. A data set consists of three consecutive windows. The current data set shares the last two windows with the previous data set and the first two windows with the next data set. The data set is a unit of analyzing signals generated by the gyro sensor.

3 Extraction of Riding Context

In this section we define indicators for calculating comfort and use them to extract contextual information about bike rides.

3.1 Basic Indicators

Depending on the sensing rate of sensor nodes, sensing data from one bike can become a large amount. We define indicators and use them instead of sensed raw data. We next define the details about meandering driving, steepness of slope, smoothness of surface, and frequency of breaking as basic indicators which represent the characteristic to show the riders that a route is comfortable or not.

Meandering Indicator. We convert yaw signals of the gyro sensor into four contexts: normal running, fast and slow meandering, and left or right turns. Fast Fourier Transform (FFT) is applied to extract frequency information. We consider meandering as a high frequency operation of a handle in a few seconds. To capture such a high frequency signal, we utilize the yaw signal. Our preliminary analysis shows that meandering affects signals in the range between 1 to 5 Hz. Therefore, we capture the signal strength for 1-5-Hz frequency. We set two levels of thresholds to distinguish between a sudden change and a slow turning. Two thresholds are set as follows:

@@Th(high): above 160(Acceleration)
@@Th(low): between Th(high) and 80(Acceleration)

The actions of left or right turns exhibit similar characteristics; yawing captures the direction of running. However the turn differs from meandering that the magnitude of 1-5-Hz signals continues to rise instead of staying at the same level. Therefore we consider that a left or right turn happens if the average signal strength increases by more than 50 in the adjacent signal windows.

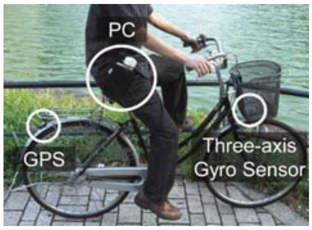


Fig. 1. Bike with GPS, three-axis sensors, and a PC for uploading data

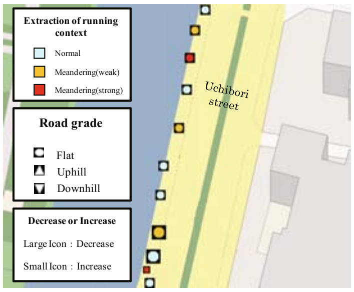


Fig. 2. BISCAY Web map showing the strength of meandering

Steepness Indicator. The steepness can be obtained using pitch signals. Therefore we can straightforwardly utilize the pitch signal to calculate the steepness. If the average of pitch in one data set exhibits 3.5%, it corresponds to a slope. As our experiential trials, most of riders feel stressed when the slope degree is more than 3.0%.

Braking Indicator. The speed of moving can be calculated using a change in positions indicated by the GPS. We also calculate a change in the speed and judge that breaking occurs when the speed decreases by more than 5 km/h. Another advantage of calculating the speed is that we can also identify the places where decrease or increase of the speed frequently happens.

3.2 Extraction

We conducted an experiment of validity of extracting contexts. The result is shown in Fig. 3. The above two graphs show snapshots of meandering. It indicates that indicators are successfully expressed. The left bottom graph shows a snapshot of running on a slope. It can detect the steepness of the road. Finally, the right bottom graph shows a situation of breaking. As shown in the graph, five times of breaking are clearly extracted.

In this experimentation, we have checked and monitored sensor data to evaluate accuracy. Braking graph shows the times of harsh braking. This evaluation shows the correct recognition of GPS data.

4 Demonstration

We will demonstrate the system that is shown in Fig. 1. Participants will be able to move the handlebar, tilt the bike, and so on, in order for the system to capture such actions using sensors. Therefore, we can easily demonstrate the system even without the space to ride the bike. The system stores and analyzes the sensed data to extract and display the comfort indicators in real time. The system then sends the extracted information to a server so as to generate a map-based visual

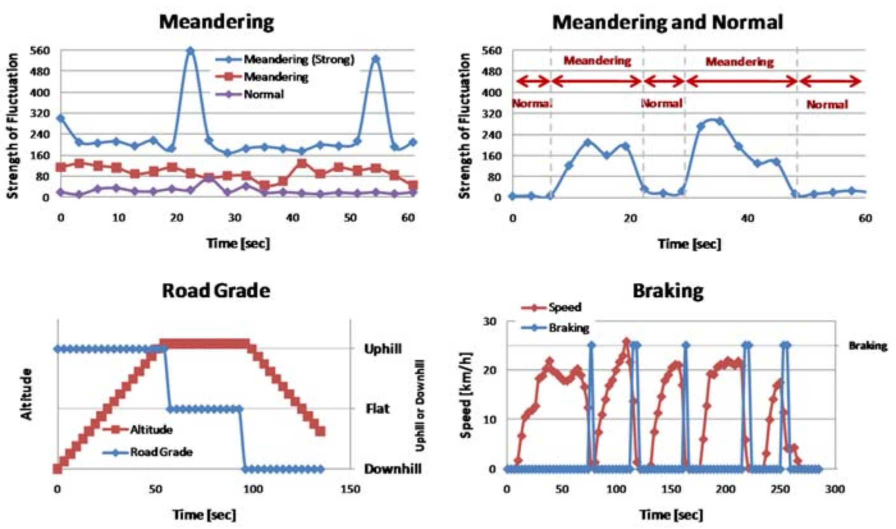


Fig. 3. Basic indicators:meandering, meandering and normal, road grade, and braking

representation (see Fig. 2) Overall, participants will be able to experience the entire process to capture, store, analyze and visualize the context of bike rides using a bike sensor network.

5 Conclusion

In this paper we have presented a system that can collect and store sensor data to show comfortable cycling routes on a map, and described the details of the demonstration of this system. Preliminary data show that the defined indicators well exhibits the status of riding and can be used for information sharing.

References

1. Eisenman, S.B., Miluzzo, E., Lane, N.D., Peterson, R.A., Ahn, G.-S., Campbell, A.T.: The BikeNet mobile sensing system for cyclist experience mapping. In: ACM Conference on Embedded Networked Sensor Systems (ACM SenSys 2007), pp. 87–101 (2007)
2. Sugo, K., Iwai, M., Tobe, Y.: BISCAY:Bike-to-Bike Sensor Network for Context Acquisition of Cycling Roads. In: 3rd International Workshop on SensorWebs, Databases and Mining in Networked Sensing Systems, SWDMNSS 2009 (2009)

SERPWatcher: A SERP Mining Tool as a Novel Social Survey Method in Sociology

Yoshifumi Masunaga[1], Naoko Oyama[2], Chiemi Watanabe[2], Kazunari Ito[1], Kaoru Tachi[2], and Yoichi Miyama[1]

[1] Aoyama Gakuin University, Fuchinobe 5-10-1, Sagamihara, Kanagawa 229-8558, Japan
{masunaga,kazu}@si.aoyama.ac.jp, d8108008@cc.aoyama.ac.jp
[2] Ochanomizu University, Otsuka 2-1-1, Bunkyo, Tokyo 112-8650, Japan
{oyama.naoko,watanabe.chiemi,tachi.kaoru}@ocha.ac.jp

Abstract. Web search engines accept keywords and return a search engine results page (SERP). Since the SERP itself and the ranking order change with time reflecting the changes in society, it might be possible to accurately follow the movement of society by mining SERPs. This demonstration shows a SERP mining tool named SERPWatcher, which could be a novel social survey method in the field of sociology in that it totally differs from the traditional methods such as questionnaires and interviews.

Keywords: Web, Search engine results page (SERP), SERP mining, SERPWatcher, Social survey method.

1 Introduction

The purpose of developing a search engine results page (SERP) mining tool named SERPWatcher is to provide a novel social survey method in the field of sociology [1]. The SERPWatcher has the following features: (1) It continuously collects SERPs with respect to a certain set of search keywords in order to formulate a “SERP archive” that the expected users, i.e. the scientists in the field of sociology, can mine. (2) It provides sophisticated interfaces and functions to the users for SERP mining. (3) It notifies the users when it detects certain changes in the ranking order of Web pages in SERPs. A beta version of SERPWatcher is currently available at our institute. The set of SERPs that SERPWatcher collected is called a SERP archive, and it constitutes a 4-dimensional “cube” whose axes are the search keyword, the search engine, the SERP collection date, and the Web page, and whose “cell” takes the value of the ranking order of the Web page. For the users to read the social movement from the SERP archive easily, the SERPWatcher collects the title, snippet (explanation of the Web page described under the page title), and the set of backlinks of the Web page in addition to its URL. Following the OLAP approach, SERPWatcher provides a user-friendly interface for a multidimensional analysis. It also provides an alert function in that it sends an e-mail to users when it detects certain ranking order changes in the latest SERP. Additionally, the top pages of a news site (namely, Google News), which are collected on the same day the SERP is collected, are also archived.

2 What Is SERPWatcher?

SERPWatcher is a tool that observes the change in the ranking order of Web pages on a SERP in a comprehensive manner and provides sophisticated interfaces and functions to expected users, i.e., scientists from the field of sociology. More precisely SERPWatcher has the following functions:

1. Periodic collection of SERPs
2. Construction of SERP archives
3. Multidimensional analysis and display of SERP archives
4. Alerts
5. Collection and display of news (Google News)

Figure 1 shows the system configuration of SERPWatcher. The SERP collection program, the backlink collection program, and the news collection program operate regularly as batch programs. Collected SERPs are recorded in a file whose records are ordered by search engine, collection date, and search keyword. Another batch program with this file as an input is run to store the file into a database. A relational database management system (MySQL, in our current implementation) is used for storing the collected SERPs. A beta version of SERPWatcher is realized as a remote system with a thin client. The operating system of the server is Red Hat Enterprise Linux 5.1 (x86/x86_64). The development programming language is Ruby 1.8.5 (2006-08-25) [i386-linux] and Ruby on Rails 2.1. MySQL 5.0.45 is used as a relational database management system.

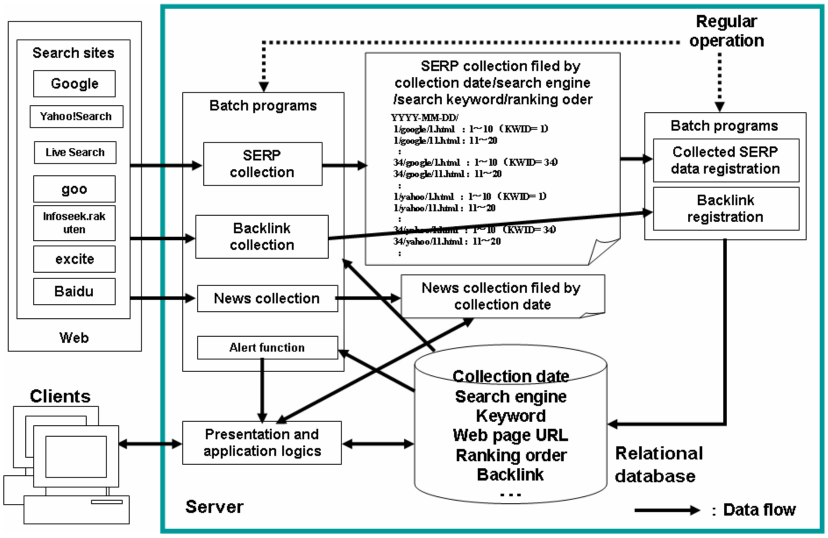


Fig. 1. System Configuration of SERPWatcher

3 Multidimensional Analysis of SERP Archive

The SERP archive that accumulates SERPs regularly collected on the basis of various search engines and the search keywords has two database features:

1. The SERP archive is not updated.
2. The SERP archive composes a multidimensional database.

Therefore, the SERP archive is suitable for OLAP (Online Analytical Processing). We examined methods of analyzing the SERP archive as a multidimensional database with the social scientists who were the intended users. As a result, it turned out that they wanted to specify a search keyword first (in other words, they want to fix the search keyword first), and then carry out the following three types of search carefully:

(a) {web page URL, collection date}→{ranking order of the web page}
(b) {web page URL, search engine}→{ranking order of the web page}
(c) {search engine, collection date}→{ranking order of the web page}

Note that demands (a), (b), and (c) correspond to the typical cubic operations named (a) search engine fixation view, (b) collection date fixation view, and (c) Web page fixation view, respectively.

Figure 2 shows a typical screenshot of SERPWatcher (in case (a)). By selecting a value from the "search engine" pull-down menu, users can select one of the seven search sites: Google, Yahoo! Search, Live Search, Baidu, goo, infoseek.rakuten, and excite. The search keyword can be selected from the pull-down menu in the "search keyword" window. The users can specify a range of interested ranking order and an interesting range of the collection dates by specifying their values in the "ranking displayed in this view. Because of space limitations, we have omitted the screenshots for case (b) and case (c). However, the screens are designed similarly.

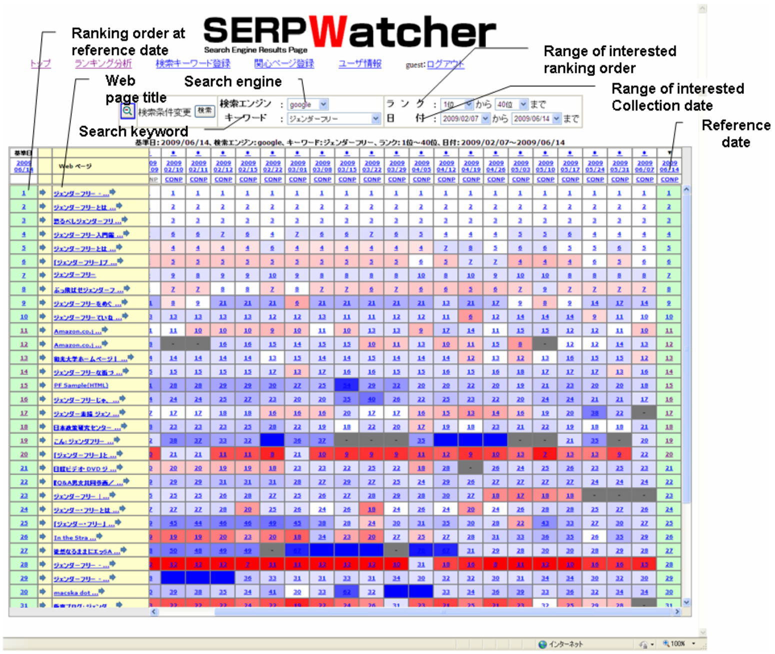

Ranking order at reference date
SERPWatcher
Search Engine Results Page
Range of interested ranking order
Web page title
Search engine
Range of interested Collection date
Search keyword
Reference date
検索エンジン : google
キーワード : ジェンダーフリー
ランク : 1位 から 40位 まで
日付 : 2009/02/07 から 2009/06/14 まで
基準日:2009/06/14, 検索エンジン:google, キーワード:ジェンダーフリー, ランク:1位~40位, 日付:2009/02/07~2009/06/14

Fig. 2. Snapshot of SERP Archive Display Corresponding to Search Engine Fixation View (Case (a)) (Vertical Axis: Web Pages, Horizontal Axis: Date, Subject: Ranking Order)

4 Demonstration Overview

The following aspects of SERPWatcher will be explained and demonstrated online via the Internet:

1. Background, purpose of the development of SERPWatcher, and related works
2. Design of SERPWatcher
3. Multidimensional analysis of SERP archive
4. Representation of multidimensional analysis of SERP archive
5. Alert function of SERPWatcher
6. Collection of auxiliary data for SERP analysis
7. Client server system configuration of SERPWatcher
8. Construction of SERP archive using a relational DBMS
9. Implementation of a beta version of SERPWatcher
10. SERP collection method and data cleaning

5 Conclusion

The development of SERPWatcher is based on our previous research on Web mining [2, 3], which gave us the insight that "SERP mining" could be a novel social study method that is completely different from the traditional social study methods such as questionnaires and interviews. The beta testers of SERPWatcher from the field of gender studies have expressed a very positive impression through the exploratory experimental use of SERPWatcher for more than 18 months.

Acknowledgments. The authors are thankful to Mr. Sumito Takeuchi of Aoyama Gakuin University for his contribution. This research is partly supported by a Grants in-Aid for Scientific Research of MEXT of Japan in the Category of Scientific Research (B) (Grant number 19300025) on "Construction of an Integrated Web Mining Environment as a New Methodology for Social Science Studies" (2007-2009).

References

1. Masunaga, Y., Oyama, N., Watanabe, C., Ito, K., Tachi, K., Miyama, Y.: Design and Implementation of SERPWatcher (2009) (to be submitted elsewhere)
2. Oyama, N., Masunaga, Y., Tachi, K.: A Diachronic Analysis of Gender-related Web Communities using a HITS-based Mining Tool. In: Zhou, X., Li, J., Shen, H.T., Kitsuregawa, M., Zhang, Y. (eds.) APWeb 2006. LNCS, vol. 3841, pp. 355–366. Springer, Heidelberg (2006)
3. Oyama, N., Masunaga, Y.: On the Trustworthiness and Transparency of a Web Search Site examined using "Gender-equal" as a Search Keyword. In: Zhang, Y., Yu, G., Bertino, E., Xu, G. (eds.) APWeb 2008. LNCS, vol. 4976, pp. 625–630. Springer, Heidelberg (2008)

Corona: Energy-Efficient Multi-query Processing in Wireless Sensor Networks

Raymes Khoury, Tim Dawborn, Bulat Gafurov, Glen Pink, Edmund Tse, Quincy Tse, K. Almi'Ani, Mohamed Gaber, Uwe Röhm, and Bernhard Scholz

The University of Sydney
School of Information Technologies
Sydney NSW 2006, Australia

Abstract. Wireless sensor networks (WSNs) are a core infrastructure for automatic environmental monitoring. We developed Corona as an in-network distributed query processor that allows to share a sensor network between several users with a declarative query language. It includes a novel approach for minimising sensor activations in shared wireless sensor networks: we introduce the notion of freshness into WSN so that users can ask for cached sensor reading with freshness guarantees. We further integrated a resource-awareness framework that allows the query processor to dynamically adapt to changing resource levels. The capabilities of this system are demonstrated with several aggregation queries for different users with different freshness and result precision needs.

1 Introduction

The latest generation of wireless sensor platforms allows for advanced in-network data processing. The central challenge remaining though is energy efficiency.

We have designed and implemented a distributed query engine for wireless sensor networks, called Corona, which runs on the Sun SPOT platform. This platform is unique in that it runs a Java VM as operating system 'on the bare metal'. In this demonstration, we show how Corona facilitates access to a shared WSN infrastructure for different users and how the integration of data caching and resource-awareness allows to trade result accuracy for energy efficiency:

- Using the declarative query interface of Corona, several users can share a WSN consisting of several Sun SPOT nodes. The queries run in the system with different start-times, epochs and lifetimes.
- Corona can adapt internal data processing functionalities to the available resources such as free memory or battery level.
- The multi-query capabilities of Corona allow to cluster sensor readings into local storage buffers on each node, which are less-frequently retrieved by de-coupled selection queries, significantly reducing communication efforts.
- Finally, Corona allows to specify a freshness constraint for queries in order to facilitate sharing of sensor readings between queries. This new freshness parameter works with arbitrary concurrent queries and helps to minimize sensor activations in the network .

The **innovation of our approach** is this combination of dynamic multi-query execution, freshness constraints for sharing of sensor activations, and adaptiveness to changing resource levels.

2 System Architecture

Corona is our research platform running on the Sun™ Small Programmable Object Technology (SPOT) sensor network platform, developed at Sun Microsystems Laboratories [1,2]. Our system is fully written in Java on top of the Sun SPOT's Squawk VM, a lightweight J2ME™ virtual machine. This gives our system the advantage of the ease of system maintenance and extension.

Corona follows a traditional WSN query processor architecture (cf. Figure 1) that consists of three components [3]:

1. the *query engine* that is executed on the Sun SPOTs,
2. the *host system* on the user's PC that is connected to the base station, and
3. a *GUI client* which connects via TCP/IP to the host system.

The control system runs on the user's PC that is connected to the base station. It provides some local persistent storage for query results and can be accessed by several users concurrently via a graphical user interface for query input and result visualisation as shown in Figure 1.

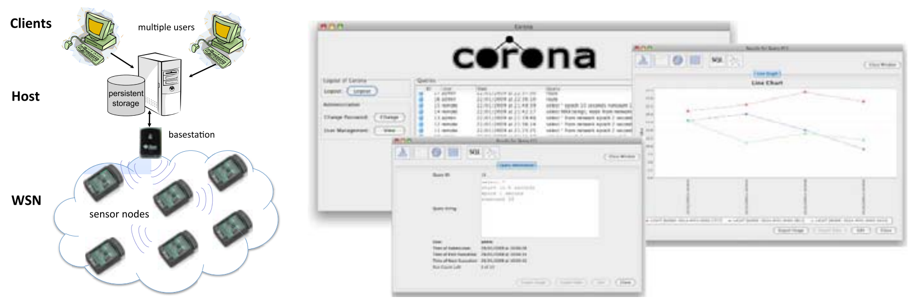


Fig. 1. System architecture and user interface of Corona

Corona uses a variant of an acquisitional SQL dialect as querying language. The language allows to access the internal state of each sensor node, filter it with selection predicates and perform in-network aggregation and clustering; attributes either specify physical sensors like light, temp (temperature), x, y, z, or meta information such as nodeid, parent, time. Queries can have specific start times, an epoch between activation and a lifetime.

An unique feature of our Corona system is its multi-tasking capability: Our query processor can execute several queries concurrently. This allows to share the same network for several applications, which helps reducing infrastructure costs. Queries can be submitted by different users or applications. Query tasks execute via a physical query plan; they are composed of relational query operators that operate on virtual horizontal partitioned relational tables. The query engine supports all the fundamental query operators including selection, projection, join and aggregation, as well as an in-network clustering operator.

The major challenge is energy efficiency because sensor nodes are battery powered. To this end, Corona provides two unique features to dynamically improve the lifetime of the system: Firstly, its in-network clustering operator allows clustering of results on the nodes in order to provide more information about the sensor readings when aggregating while still keeping number transmitted messages small. Instead of sending each individual sensor reading to the basestation, this operator allows to cluster the sensor readings around frequent values into a local storage on the node. A second de-coupled query than allows to retrieve those clustering results only. This de-coupled selection query can run with a much longer activation epoch, yielding a reduction of the communication effort. Furthermore, this clustering operator is resource-aware, meaning it can dynamically adapt its processing granularity (by means of the clustering distance threshold) and its output granularity (by means of merging neighbouring clusters) based on pre-defined resource limits [4].

A second innovation is the freshness mechanism that Corona incorporates which allows to specify per query a threshold up to which sensor readings of previously activated queries are to be used for a second query too:

```
SELECT temperature
  FROM sensors
 WHERE  light > 100
 EPOCH 60s RUNCOUNT 60
 FRESHNESS 10s
```

The freshness constraint specifies how much time is allowed to pass since the last sensor acquisition so that the query engine would not activate the sensor again, but rather use the previous sensor reading from the cache on the sensor node. This decision is done dynamically on every query activation. Additionally, the query optimizer at the host can take this freshness constraint into consideration to determine the optimal time shift (up to a maximum value) of the start time of a new query, so that the cache hits during its lifetime are maximized. We have developed a heuristic algorithm which calculates the near-optimal time-shift at the host. Our performance studies show that heuristic is very scalable in terms of run count and query count, but has linear one to one dependency on number of queries.

3 Background and Related Work

Distributed query processing in wireless sensor networks has been an active research area over the last few years. TinyDB [5] and Cougar [6] represent the first generation of query processing systems in wireless sensor networks. The state of the art is to abstract from the underlying sensor hardware by basing the WSN query processors on virtual machines (VM) that run on the sensor nodes [7]. Corona falls into this category by using the Squawk Java VM of Sun SPOT nodes and focuses specifically on multi-query execution and resource-awareness. The adaptive clustering operator in Corona, called ERA-Cluster, was initially presented in [8].

Running several queries concurrently can also be seen as a way to reduce power consumption. [9] introducing the notion of a network query which is a combination of all user queries running in the system. Hence, the query processor on the noes is actually executing only one query in the system. Corona in contrast only executes the currently needed query and especially activates only the necessary sensors.

4 Summary and Conclusions

The central challenge for WSNs is energy and communication efficiency. We have developed a distributed query processor for the novel Sun SPOT platform that provides a declarative query interface to users and allows to dynamically share the WSN infrastructure between several user queries.

In order to maximize the network lifetime, Corona provides a resource monitoring component through which the query engine can dynamically adapt its processing to changing resource levels. We have further applied the idea of caching to shared wireless sensor networks in order to minimise sensor activations. Corona's query language allows users to choose how outdated sensor readings are allowed to be before requiring a separate sensor activation for a query. We also developed a technique of shifting query starting times to increase cache usage. All these capabilities are implemented in a working prototype that is available under GPL.

Acknowledgements. This work is supported by the Australian Research Council (ARC) under grant DP0664782, and by Sun Microsystems Labs.
Corona is available under GPLv3 from [10].

References

1. Simon, D., Cifuentes, C., Cleal, D., Daniels, J., White, D.: Java on the bare metal of wireless sensor devices: the Squawk java virtual machine. In: Proc. of the 2nd Int. Conf. on Virtual Execution Environments, VEE (2006)
2. Sun Microsystems: SunSpotWorld website, `http://www.sunspotworld.com/`
3. Scholz, B., Gaber, M.M., Dawborn, T., Khoury, R., Tse, E.: Efficient time triggered query processing in wireless sensor networks. In: Lee, Y.-H., Kim, H.-N., Kim, J., Park, Y.W., Yang, L.T., Kim, S.W. (eds.) ICESS 2007. LNCS, vol. 4523, pp. 391–402. Springer, Heidelberg (2007)
4. Roehm, U., Gaber, M.M., Tse, Q.: Enabling resource-awareness for in-network data processing in wireless sensor networks. In: Proceedings of ADC 2008 (2008)
5. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: TinyDB: an acquisitional query processing system for sensor networks. ACM TODS 30(1) (2005)
6. Yao, Y., Gehrke, J.: The cougar approach to in-network query processing in sensor networks. SIGMOD Record 31(3), 9–18 (2002)
7. Müller, R., Alonso, G., Kossmann, D.: SwissQM: Next generation data processing in sensor networks. In: Proceedings of CIDR 2007, Asilomar, USA (2007)
8. Phung, N.D., Gaber, M.M., Röhm, U.: Resource-aware online data mining in wireless sensor networks. In: Proceedings of the IEEE CIDM Symposium (2007)
9. Trigoni, N., Yao, Y., Demers, A., Gehrke, J., Rajaraman, R.: Multi-query optimization for sensor networks. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 307–321. Springer, Heidelberg (2005)
10. The Corona Website (2009), `http://www.it.usyd.edu.au/~wsn/corona/`

Aquiba: An Energy-Efficient Mobile Sensing System for Collaborative Human Probes

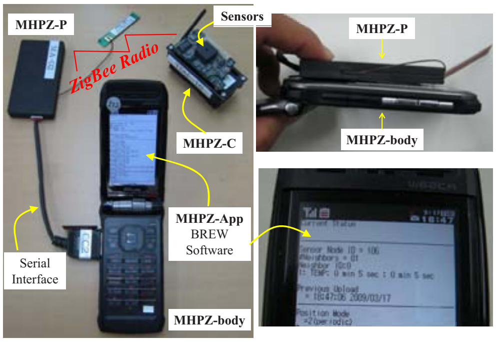
Niwat Thepvilojanapong[1,2], Shin'ichi Konomi[1,2], Jun'ichi Yura[3], Takeshi Iwamoto[4], Susanna Pirttikangas[5], Yasuyuki Ishida[1], Masayuki Iwai[6], Yoshito Tobe[1,2], Hiroyuki Yokoyama[7], Jin Nakazawa[3], and Hideyuki Tokuda[3]

[1] Tokyo Denki University, Japan
wat@osoite.jp
[2] CREST, JST, Japan
[3] Keio University, Japan
[4] Toyama Prefectural University, Japan
[5] University of Oulu, Finland
[6] University of Tokyo, Japan
[7] KDDI R&D Laboratories, Japan

Abstract. Portable sensory devices carried by humans—which are referred to as *Human Probes*—facilitate easy-to-use sensing and monitoring of urban areas. In this demonstration, we developed a prototype of *Aquiba* sensing system from off-the-shelf mobile phone. Aquiba involves *collaborative sensing* that helps in achieving high-fidelity sensing while minimizing overall energy consumption. We validated the benefit of collaborative sensing through field experiments.

1 Introduction

Human-Probe sensing, which allows ordinary people carrying sensors to participate in data collection, provides an exciting opportunity to design a novel humans-in-the-loop sensing environment because it eliminates the hindrance of deploying myriad static sensors across a wide area and leads to a fully distributed urban sensing. Human Probes consider not only participatory and opportunistic modes of data capture but also collaboration without community bonds. Such collaboration could take place automatically among strangers, even without them being aware of it. Therefore, this paper focuses on the design and analysis of collaborative Human Probes by exploiting dynamic, emergent, and ephemeral pedestrian groups. We propose a mobile sensing system called *Aquiba* for efficiently collect environmental data in urban areas. In order to minimize total energy consumption while still achieving high-fidelity sensing, Aquiba autonomously adjusts sensing rate of each Human Probe based on the availability of nearby Human Probes which always changes along the time. We develop prototype devices and conducted field experiments in which the results confirm the efficacy of Aquiba system in terms of energy efficiency and sensing fidelity.



```
Current Status
Sensor Node ID = 106
#Neighbors = 01
Neighbor ID:0
1: TEMP: 0 min 5 sec : 0 min 5 sec
Previous Upload
 = 18:47:06 2009/03/17
Position Mode
 =2(periodic)
```

Fig. 1. A prototype of Human-Probe device

2 Aquiba: Collaborative Sensing System

Aquiba considers the following issues which are decided on the basis of the various factors relating to Human-Probe sensing: (i) an environmental sensing system consists of a server, mobile phones, and sensors; (ii) the mobile phones are equipped with cellular and short-range wireless interfaces; (iii) the sensors are either integrated in the mobile phones or positioned at various locations in the environment to capture environmental data; (iv) the server issues a query including the desired *sensing rate* R_i for each data type for each *sensing area* A_i, where i indicates the index of the sensing areas; (v) the mobile phones are able to acquire their current location information. In particular, the Human Probes that we consider perform implicit sensing tasks, i.e., the sensors automatically capture the requested data and the mobile phones upload the data to the server via a cellular network. The mobile phone uses short-range wireless interface to communicate with nearby sensors and other mobile phones.

Upon receiving a query from the server, Human Probe which exists in A_i performs the collaborative sensing by adjusting its sensing rate to $\frac{R_i}{k_i}$, where k_i is the number of Human Probes in A_i. To maintain information of other Human Probes in the same sensing area, a mobile phone uses short-range radio to broadcast beacon packets periodically. Based on received beacon packets, a Human Probe can determine the current number of neighboring Human Probes and accordingly adjust its sensing rate. Each Human Probe needs to set an expiry time for each neighboring Human Probe and delete the expired neighbors from its neighbor table periodically. In our implementation, the determination of the beacon interval and neighbor management is based on those of AODV protocol [1].

3 Prototype Implementation

We developed a platform of *Mobile-phone-based Human Probe with ZigBee (MHPZ)*. The hardware of MHPZ consists of three components: MHPZ body, MHPZ parent (MHPZ-P), and MHPZ child (MHPZ-C) as shown in Fig. 1. The MHPZ body is an off-the-shelf mobile phone and functions as a Human Probe if it is physically connected with MHPZ-P. The MHPZ-P, which is equipped with a ZigBee interface, is connected to the MHPZ body via a serial interface and can communicate with any ZigBee devices. Thus the MHPZ-P functions as a gateway between the mobile phone and various sensors with ZigBee communication capability. Finally, the MHPZ-C can sense and transmit data to the MHPZ-P via ZigBee radio.

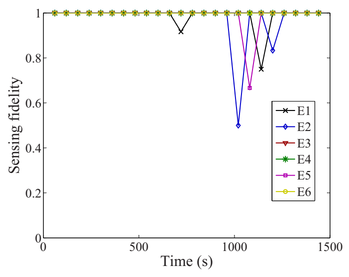
Inside the MHPZ body, we implemented a BREW application software (MHPZ-App) to handle communication with an MHPZ-P and control cellular communication of mobile phone. The MHPZ-App controls and monitors the MHPZ-P through a synchronous command, i.e., the MHPZ-App issues a command (e.g., asking to broadcast a beacon packet) to the MHPZ-P, which in turn sends corresponding replies (e.g., informing neighbor information or sensed data) to the MHPZ-App. The MHPZ-App calculates an appropriate sensing rate based on the Aquiba protocol. In our prototype implementation, MICAz Motes are used for both MHPZ-P and MHPZ-C, and Casio G'zOne W62CA is the MHPZ body.

4 Experimental Study

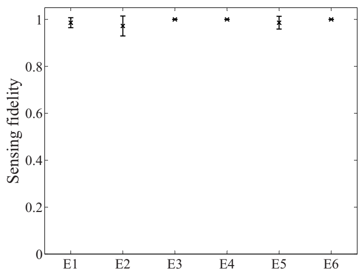
We asked 12 participants to carry the MHPZ platforms and walk in an 156m-by-132m area. The MHPZs collaboratively captured temperature data and uploaded them to a server. There were six experimental scenarios (E1–E6), each of which specified a different walking pattern and stopping probability. Participants walked freely in scenarios E1–E4, while they followed a different sequence of predetermined way points in scenarios E5 and E6. When people run into their acquaintances on the street, they may stop for a short period to say hello and have a conversation. We examined the impact of such a behavior by using different stopping probability in each scenario, i.e., 1.00, 0.50, 0.33, 0.25, 1.00, and 0.33 for scenarios E1, E2, E3, E4, E5, and E6, respectively. Each participant received a trump card, and they stopped when encountering a participant who carry the same trump card. In all scenarios, R_i was set to 12 times per minute, and the beacon packets were broadcast every five seconds.

Human Probes aim to maximize *sensing fidelity* which is defined as a ratio of the sensing rate perceived by the server to the desired sensing rate in a given sensing area. If the total number of packets arrives at the server during the period ΔT is N, the sensing fidelity is $\frac{\min\left(\frac{N}{\Delta T}, R_i\right)}{R_i}$. We also calculate *upload ratio* which is a ratio of the number of uploads carried out by Aquiba protocol to the number of uploads carried out by non-collaborative Human Probes.

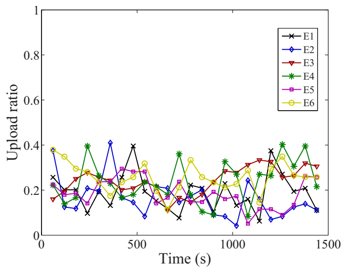
The results of all scenarios are shown in Fig. 2. Fig. 2a shows the variation of sensing fidelity along time axis. For all scenarios, sensing fidelity generally achieves the highest level (1.0) except for the several instances being temporarily below 1.0. The lower sensing fidelity is partially due to stale neighbor information. Fig. 2b shows overall sensing fidelity for the entire experimental period along with 95% confidence intervals. As far as sensing fidelity is concerned, there is no significant



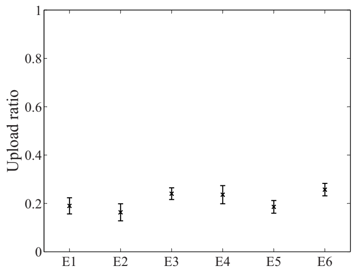
(a) Temporal variation of sensing fidelity.



(b) Overall sensing fidelity.



(c) Temporal variation of upload ratio.



(d) Overall upload ratio.

Fig. 2. Experimental results

difference among all scenarios. The experiments showed that sensing fidelity is not significantly affected by the stopping probabilities and walking patterns.

The temporal variation of upload ratio (Fig. 2c) was normally lower than 40%. Fig. 2d shows overall upload ratio for the entire experimental period along with 95% confidence intervals. The upload ratios of scenarios E5 and E6 (predetermined routes) are significantly different, i.e., the number of uploads is reduced when the stopping probability is high. The upload ratios of scenarios E1, E2, E3, and E4 (free walk) suggests a similar impact of the stopping probability on upload ratio, i.e., higher stopping probability leads to lower upload ratio.

5 Demonstration

In the demonstration, we will prepare MHPZ devices and show how the mobile phones acquire data from sensor, and how the phones find neighbor Human Probes and adjust their sensing rate automatically. Users can check current status and set parameters of Aquiba through GUI of mobile phones. We will also prepare the server to show collected sensor data in real-time.

Reference

1. Perkins, C.E., Belding-Royer, E.M., Das, S.: Ad hoc on-demand distance vector (AODV) routing. RFC 3561, IETF (July 2003)

A Shoes-Integrated Sensing System for Context-Aware Human Probes

Kazumasa Oshima[1], Yasuyuki Ishida[1], Shin'ichi Konomi[1,2], Niwat Thepvilojanapong[1,2], and Yoshito Tobe[1,2]

[1] Tokyo Denki University, Japan
shima@u-netlab.jp
[2] CREST, JST, Japan

Abstract. *Human Probes*, which are human integrated or embedded with sensors, allow the acquisition of a variety of contextual information, facilitate collaborative information sharing and community action as well as the provision of personalized services such as personal health management and context-aware advertisements. Recently, we have examined the usefulness of pressure sensors embedded in shoes [2]. In this demonstration, we extend our previous research on embedded pressure sensors by considering complimentary uses of accelerometers so as to capture precise and meaningful context in our daily lives. Pressure sensors and accelerometers are similarly useful for capturing the motion of pedestrians; however, the close examination of the signals from both sensors reveals the strengths and the weaknesses of each, and suggests the possibility of their complimentary use to support Human Probes.

1 A Human-Probe System

We designed and implemented a Human-Probe system that captures data from accelerometers and pressure sensors embedded in shoes. Figure 1 shows the overview of the system that integrates pressure sensors and Sun SPOTs [1]. The Sun SPOTs, which are equipped with a three-axis accelerometer and a wireless communication interface, are attached at the heel portions of the shoes, with x, y, and z axes pointing to the directions shown in Fig. 1. Two pressure sensors are embedded in a shoe, one at the toe and the other at the heel portion.

Using the system, we carried out field experiments and collected data from a pedestrian who put on the sensor-enabled shoes and walked in four different walking conditions (see Table 1).

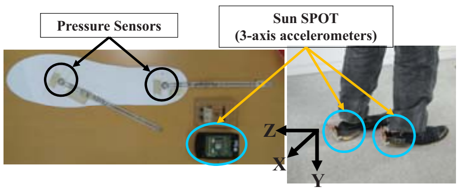


Fig. 1. Sensor-enabled shoes

Table 1. Walking conditions

Walking conditions	Details
Flat surface	Asphalt pavement
Stair	18cm-high, 28cm-wide steps
Slope	25-degree slope
Lawn	15cm-high grass

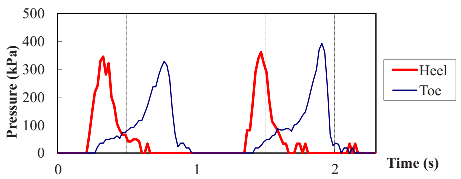


Fig. 2. Pressure values when walking on a flat surface

2 Data Analysis

We collected data in four different places in Tokyo, each of which corresponds to different walking conditions in Table 1. This section presents the analysis of the data.

2.1 Analysis of Pressure Data

Figure 2 shows the pressure values measured by two pressure sensors in a shoe. It is likely that the peak patterns could provide useful information for estimating the context of walking. Figure 3 shows the distribution of the peak values at the toe and the heel for each walking environment. The distribution suggests that we can estimate the ground-surface conditions using learned threshold values. We conclude that the pressure sensors embedded in shoes can provide information that is useful for understanding the context of walking such as ground-surface conditions.

2.2 Analysis of Acceleration Data

We can also estimate the context of walking by examining acceleration patterns. Figure 4 shows the acceleration data along the y and z axes, which were captured from a pedestrian walking on a flat surface as he pulls his foot up and moves it forward and down. We found that fluctuation patterns of y-axis acceleration data in A and B shaded periods in the figure are different for each walking condition, thereby allowing us to capture information related to up-down motion of a foot. The acceleration data along the z-axis show a repeating waveform (see Figure 4). We also verified that the waveform of z axis changes when we impose restrictions on step length. This suggests that the acceleration data along the z-axis is useful for acquiring information about step length. We conclude that the accelerometers embedded in shoes can provide information that is useful for understanding the motion of walking.

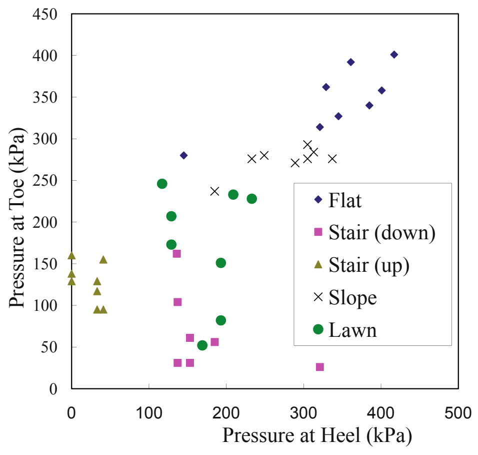


Fig. 3. A distribution of peak values of pressure sensors

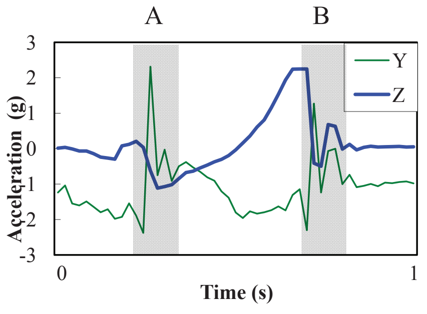


Fig. 4. Acceleration values when walking on a flat surface

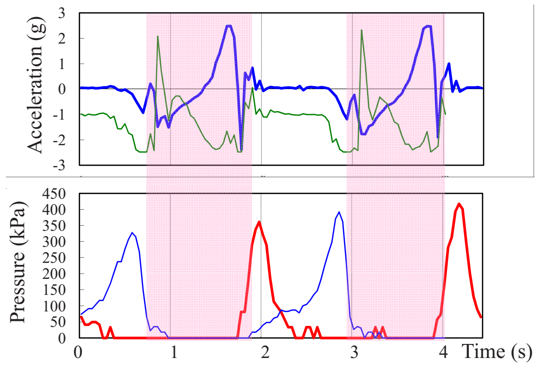


Fig. 5. Complimentary use of accelerometers and pressure sensor.

2.3 Complimentary Usage of Sensors

According to the above discussion, accelerometers are able to capture dynamic context such as walking patterns, on the other hand, pressure sensors are able to capture static context such as ground-surface conditions. This suggests complimentary sensing functions of both sensors. Figure 5 supports this conclusion by showing a situation that only one kind of sensor is able to capture changes of the sensing value (the shaded periods in the figure), while the value captured by the other sensor does not change during the periods. Figure 6 shows a complicated case of walking data. Acceleration data do not show any significant patterns; thus a high-level, high-cost technique is required to analyze and understand the data. However, we can extract each walking step easily by applying a simple recognition algorithm on pressure data.

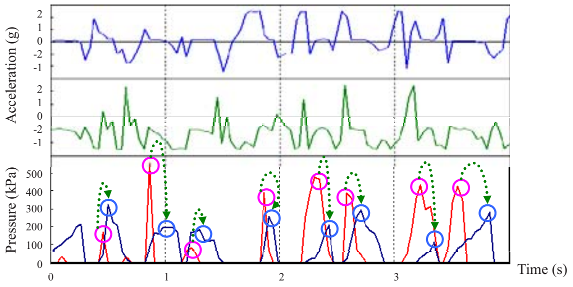


Fig. 6. Detection of complicated walking data

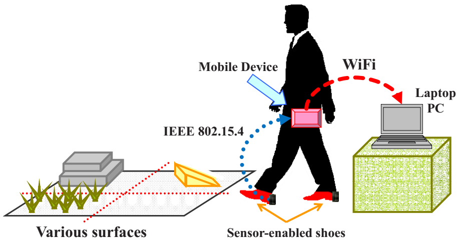


Fig. 7. Illustration of demonstration

3 Demonstration

Although we are using mobile phones to validate participatory sensing, we use WiFi in this demonstration to avoid possible troubles. Subjects can try to put on shoes, which are equipped with accelerometers and pressure sensors, and walk on four kinds of surfaces (as shown in Table 1). Figure 7 illustrates the structure and components of demonstration. As described in Sect. 1, we integrate Sun SPOTs and pressure sensors in shoes. The Sun SPOTs use IEEE 802.15.4 to transmit sensor data to a mobile PC, which in turn forwards the data to a laptop PC through WiFi interface. Upon receiving sensor data, the laptop PC analyzes and shows current ground-surface condition as well as other walking information such as walking patterns of subjects.

References

1. Sun SPOT World, http://www.sunspotworld.com
2. Uehara, Y., Uchiyama, T., Mori, M., Saito, H., Tobe, Y.: Always-on karte: A system for elderly people's healthcare using wireless sensors. In: INSS 2006, May 2006, pp. 45–48 (2006)

DigestJoin: Expediting Joins on Solid-State Drives[*]

Shen Gao, Yu Li, Jianliang Xu, Byron Choi, and Haibo Hu

Depart of Computer Science
Hong Kong Baptist University
Kowloon Tong, Hong Kong SAR, China
{sgao,yli,xujl,bchoi,haibo}@comp.hkbu.edu.hk

Abstract. This demonstration presents a recently proposed join algorithm called *DigestJoin*. Optimized for solid-state drives (SSDs), *DigestJoin* aims at reducing intermediate join results and hence expensive write operations while exploiting fast random reads. The demonstration system consists of an implementation of *DigestJoin* in the open-source PostgreSQL database management system on an Intel SSD. In the demonstration, we will showcase the performance benefits of *DigestJoin* in comparison to a traditional join algorithm and highlight the workloads in which *DigestJoin* is particularly favorable.

1 Introduction

Solid-State Drives (SSDs) have recently been a competitive alternative secondary storage for database applications, thanks to their superiority such as low access latency, low power consumption, and excellent shock resistance [1]. However, compared to magnetic disks, SSDs possess a number of distinct I/O characteristics, which affect database applications, among others. First, SSDs do not involve any mechanical components so that there is a negligible *seek time* in reading pages. A random read is almost as fast as a sequential read on SSDs. Second, SSDs have an *erase-before-write* constraint: a page has to be erased before it can be overwritten. Although this can be addressed by the *out-place* update strategy, new issues such as wear leveling and garbage collection arise, rendering a write slower than a read on SSDs. Third, with the short I/O latency (*e.g.*, the random read of an SSD is 150X faster than that of a magnetic disk [1]), I/O cost may no longer dominate CPU computation cost in evaluating a query on an SSD-based database system. These distinct I/O characteristics make the state-of-the-art join algorithms, which assume I/O characteristics of magnetic disks, suboptimal when implemented on SSDs.

In this demonstration, we present *DigestJoin* — a recently proposed algorithm that optimizes join performance for SSDs by reducing intermediate join results and

[*] This work was partially supported by the Research Grants Council of Hong Kong (Grants HKBU210808 and HKBU211307) and Natural Science Foundation of China (Grant No. 60833005).

exploiting fast random reads [2]. *DigestJoin* consists of two phases: *digest-table joining* and *page fetching* (see Figure 1). In the first phase of digest-table joining, *DigestJoin* projects the tuple id (*tid*) as well as the attribute(s) that participate in the join. The projected tables are called the *digest tables*. A traditional join algorithm is then applied on the digest tables to generate the *digest join results*. The digest join results are simply pairs of *tid*s together with the join attribute(s), thereby minimizing the size of intermediate join results. In the second phase of page fetching, based on the digest join results, *DigestJoin* loads the full tuples that satisfy the join from the original tables to produce the final join results. Whenever a tuple is fetched from disk, the entire page containing the tuple is fetched. Ideally, each page should be fetched at most once during the process of final-result construction. However, this is difficult to achieve in practice due to memory constraints. As the digest join results are not clustered with respect to page address, a page may be fetched multiple times during the construction process. Thus, a page fetching strategy is needed to minimize the number of page accesses. In the following sections, we provide more details of each of the two phases in *DigestJoin* with an example.

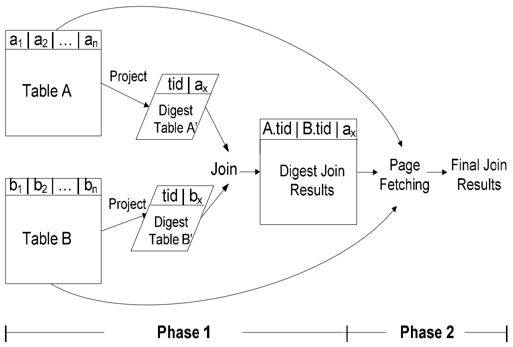


Fig. 1. Overview of *DigestJoin*

Fig. 2. Connecting SSD to motherboard

2 Digest-Table Joining

Consider two tables $A = \{a_1, a_2, ..., a_n\}$ and $B = \{b_1, b_2, ..., b_n\}$. Denote the tuple ids of these two tables by *A.tid* and *B.tid*, respectively. In the first step, we scan tables *A* and *B* and compute the digest tables that contain only the join attributes and the tuple ids. For example, given a simple join $A \bowtie_{A.ax=B.bx} B$, the digest tables will be $A' = \{A.tid, A.a_x\}$ and $B' = \{B.tid, B.b_x\}$. After that, we apply a traditional join algorithm (e.g., nested-loop, hash join, or sort-merge) to the digest tables to generate the digest join results, *e.g.*, in the form of $\{A.tid, B.tid, a_x\}$ for the above example. As the digest tables are often much smaller than the original tables, the I/O of the join, especially the write operations on SSDs, would be greatly reduced.

3 Page Fetching

The digest join results consist of only the *tid*s of the tuples that satisfy the join. To produce the final join results, we fetch the tuples from the original tables according to *tid*s. The fetching is performed at page-level granularity. This has been known to be the classical *page fetching problem* in index-based joins. However, as random reads are no longer an issue on SSDs, we can simply minimize the *amount of I/O*s in fetching the full tuples. On the other hand, due to the short I/O latency of SSDs, the CPU computation cost of page fetching should also be taken into consideration.

As an illustration, one straightforward solution is to fetch the pages of the tuples as soon as they are generated in the digest-table joining phase, and cache them in a buffer for future use. Since random reads are fast on SSDs, we assign as few input buffers as possible in the digest-table joining phase in order to maximize the number of buffers for page caching. For example, suppose that sort-merge is used to join the digest tables. Page fetching is incorporated in the merge phase of the digest-table join (*i.e.*, after the digest tables have been sorted). We assign only two input buffers to merge the two sorted digest tables. The remaining buffer space forms a page cache. A cache replacement policy, LRU, is used for the management of the page cache. This page fetching strategy maximizes the amount of cached pages. Meanwhile, it does not incur much CPU computation cost. More advanced page fetching strategies for SSDs have also been proposed; interested readers may refer to [2] for details.

4 Demonstration Description

We have implemented *DigestJoin* in PostgreSQL 8.3.6 [3], an open-source database management system. We store the TPC-H tables on an Intel 80GB X25-M SSD, which are connected to the motherboard via a SATA II connection (see Figure 2). In this demonstration, we will showcase the performance benefits of *DigestJoin* in comparison to a traditional sort-merge join (*TraditionJoin*) algorithm and highlight the workloads in which *DigestJoin* is particularly favorable.

Figure 3(a) gives a screenshot of the GTK+ interface of our demonstration system. After launching the system, the user can input a join query in standard SQL form. Below is an example:

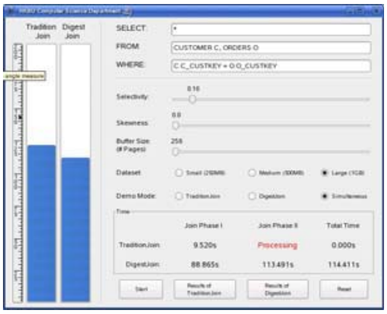
```
SELECT *
FROM CUSTOMER C, ORDERS O
WHERE C.C_CUSTKEY = O.O_CUSTKEY
```

Next, the user can set a number of parameters for execution:

- **Join result selectivity**: This adds to the WHERE clause of the user query an additional filtering function that selects part of the original join results. The selectivity can be ranged from 0.01 to 1.0.
- **Skewness of join results**: When used with the join selectivity, this parameter controls the page distribution of selected join results. When it is set at 0, the join results are evenly distributed; when it is set at 1, the selected join results are highly clustered on a few hot pages.

- **Buffer size**: This sets the size of the buffer (in terms of the number of 8KB pages) used for the join algorithm.
- **Dataset size**: Each dataset under testing has three sizes for selection: small (250MB), medium (500MB), and large (1GB).

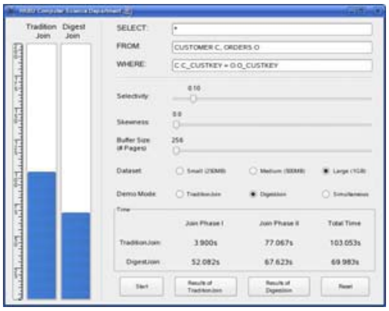
The user can also choose an execution mode. In *simultaneous* mode (Figure 3(a)), to contrast the performance difference, *DigestJoin* and *TraditionJoin* will be executed side-by-side, and their elapsed times will be visualized in the performance bars (see the left part of Figure 3(a)) in real time. For *TranditionJoin,* the whole join process is divided into three stages: scanning and sorting the outer table (Phase I), scanning and sorting of the inner table (Phase II), and merging join results (Total Time). For *DigestJoin,* the whole process is divided into generating digest join results (Phase I), page fetching (Phase II), and generating final results (Total Time). During the execution, the "Processing" status will be displayed in the corresponding stage. After completing all stages, the total elapsed time will be reported. In an alternative *separate* mode (Figure 3(b)), the two join algorithms will be executed one after the other in order to eliminate their possible performance interference of simultaneous execution. Finally, the user can check the join results by clicking the Results buttons. An online demo video for *DigestJoin* is available at http://www.comp.hkbu.edu.hk/~db/demo.



SELECT: *
FROM: CUSTOMER C, ORDERS O
WHERE: C.C_CUSTKEY = O.O_CUSTKEY
Selectivity: 0.10
Skewness: 0.0
Buffer Size (# Pages): 256
Dataset: ( ) Small (250MB) ( ) Medium (500MB) [✓ Large (1GB)]
Demo Mode: ( ) TraditionJoin ( ) DigestJoin [✓ Simultaneous]

	Join Phase I	Join Phase II	Total Time
TraditionJoin	9.520s	Processing	0.000s
DigestJoin	88.865s	113.491s	114.411s

(a) Simultaneous mode



SELECT: *
FROM: CUSTOMER C, ORDERS O
WHERE: C.C_CUSTKEY = O.O_CUSTKEY
Selectivity: 0.10
Skewness: 0.0
Buffer Size (# Pages): 256
Dataset: ( ) Small (250MB) ( ) Medium (500MB) [✓ Large (1GB)]
Demo Mode: ( ) TraditionJoin [✓ DigestJoin] ( ) Simultaneous

	Join Phase I	Join Phase II	Total Time
TraditionJoin	3.900s	77.067s	103.053s
DigestJoin	52.082s	67.623s	69.983s

(b) Separate mode

Fig. 3. User interface of the demonstration system

References

1. Lee, S.-W., Moon, B., Park, C., Kim, J.-M., Kim, S.-W.: A Case for Flash Memory SSD in Enterprise Database Applications. In: Proceedings of SIGMOD, pp. 1075–1086 (2008)
2. Li, Y., On, S.T., Xu, J., Choi, B., Hu, H.: DigestJoin: Exploiting Fast Random Reads for Flash-based Joins. In: Proceedings of the 10th International Conference on Mobile Data Management (MDM 2009), pp. 152–161 (2009)
3. PostgreSQL, `http://www.postgresql.org/`

A Large Scale Key-Value Store Based on Range-Key Skip Graph and Its Applications

Susumu Takeuchi[1], Jun Shinomiya[2], Toru Shiraki[3], Yoshimasa Ishi[3], Yuuichi Teranishi[1,3], Mikio Yoshida[4], and Shinji Shimojo[1]

[1] National Institute of Information and Communications Technology, Japan, {stakeuti,sshinji}@nict.go.jp

[2] School of Engineering, Osaka University, Japan shinomiya.jun@ise.eng.osaka-u.ac.jp

[3] Graduate School of Information Science and Technology, Osaka University, Japan {shiraki.toru,ishi.yoshimasa,teranisi}@ist.osaka-u.ac.jp

[4] BBR Inc., Japan, yos@bbr.jp

Abstract. An overlay network called Range-key Skip Graph (RKSG) has been proposed that can perform range-to-range search on peer-to-peer network. In this paper, we propose a large scale key-value store realized on RKSG, which is possible to scale-out a database with range query. Moreover, applications that utilize the store are implemented and will be demonstrated through private PlanetLab on JGN2plus.

Keywords: P2P Overlay Network, Range-key Skip Graph, Key-value Store.

1 Introduction

Recently, key-value store that allows a massive number of computers to have a large scale fast database is adopted in many systems. However, existing key-value stores only support to retrieve data by a single key, i.e., only exact matching is provided. In the ubiquitous environment, range retrieval that can extract data within a specific range of a key is required as well as exact matching. For example, a weather sensor application would need to retrieve the current observed data in each sensor located in a specific region, and a navigation application would need to retrieve movement histories of people in a specific region to recommend a route to go to a certain place.

In order to realize key-value store that supports exact matching, DHT (Distributed Hash Table), which is a structured P2P (Peer-to-Peer) overlay network such as Chord [1] and Kademlia [2], is effective for archiving high scalable one. Windows Azure[1] is one of the available implementations of DHT-based key-value store. However, DHT-based architecture cannot support range retrieval so that realizing key-value store which supports range query is difficult. Accordingly, a structured overlay network Skip Graph (SG) [3] is proposed to support range

[1] Windows Azure Platform: http://www.microsoft.com/azure/windowsazure.mspx

retrieval, but SG is designed for searching a node so that it is difficult to handle a massive amount of data.

Therefore, we have proposed Range-key Skip Graph (RKSG) [4] that is based on SG and is developed to handle range-to-range query. In RKSG, a node can have multiple ranges and search other nodes that manage the required range of a query. A large scale key-value store is implemented by utilizing RKSG in this paper, which allows retrieving all the data that is assigned to the keys within the range of the query. As a demonstration, applications are developed to show the availability and scalability of the store that can handle a massive amount of data and nodes. The applications include a visualization tool of weather sensor stations and movement histories of users.

2 Range-Key Skip Graph

Skip Graph is a structured overlay network that is based on Skip List [5], but is extended for applying to the distributed environment. In SG, each node belongs to higher level independently at a certain probability, and the nodes that are belong to the same level connect each other based on membership vector. To retrieve a key, a search node transfers a query to the neighboring connected nodes from the upper level to the lower level. Since SG includes range query feature in its routing algorithm, implementing it in the distributed environment is easier than DHT-based methods that support range query. However, each node can have only one key, so managing enormous data, e.g., location-dependent contents, by the limited number of nodes efficiently is difficult.

To address this issue, Range-key Skip Graph that handles a range as a key has been proposed. In SG, keys must be arranged totally ordered relation, so a range key should be managed by dividing the range to the independent range(s) or by defining order relation by extracting a certain part of the range as a representative key. The former method is very difficult to implement in P2P distributed environment where a node joins and leaves frequently, because dividing and combining the ranges of the keys is inadequate whenever it occurs. The latter method is easier to implement, but range-keys that meet a request sometimes cannot be retrieved because the part or all the ranges of the keys are included in another range key.

Accordingly, in RKSG, the minimum value of a range of a key is treated as a representative key of the range key as illustrated in Figure 1. In addition, each link and range of other range keys that include the minimum value of the range key is hold in the inclusion keys list of the key. When a node wants to retrieve a range, a query is forwarded by utilizing the inclusion keys list of the retrieved keys. Consequently, RKSG supports range-to-range retrieval based on SG.

3 Demo Applications

For demonstrating the availability and scalability of RKSG, the following demo applications which include the RKSG-based key-value store are implemented

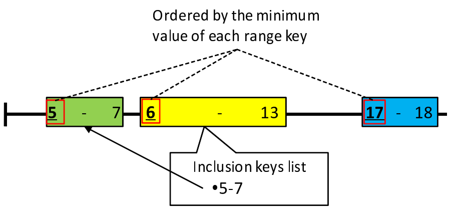


Fig. 1. Key Arrangement in Range-key Skip Graph

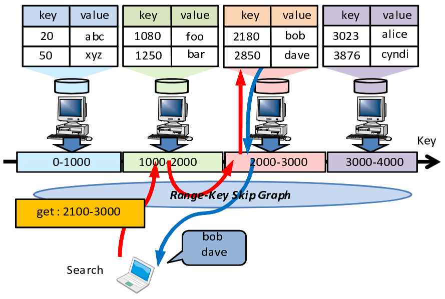


Fig. 2. Key-value Store Supports Range Query

and will be demonstrated through private PlanetLab in JGN2plus where many nodes are available over wide area.

3.1 Key-Value Store with Range Query

Key-value store is a sort of database that holds data as a pair of a key and a value. Traditional database system, i.e., relational database management system (RDBMS), can have multiple values for a piece of data as columns, but is difficult to apply in the environment where an enormous amount of data is available because of lack of scalability. On the contrary, key-value pairs in key-value store can be divided by the ranges of the keys so that the data can be easily divided and distributed in multiple servers. Thus, key-value store can scale-out a database, which is difficult to achieve by RDBMS, and that is necessary for the distributed environment.

Several implementations of key-value store have been provided, but existing key-value stores cannot support range key and range query that is inadequate in the ubiquitous environment where many nodes (e.g., sensors, cellular phones) join, leave, or move frequently. Thus, the RKSG-based key-value store is implemented in this paper. By utilizing RKSG, each node can have its management area that is indicated by range-key, and the key-value store can support range query as illustrated in Figure 2. Consequently, an application can search the related nodes that manage a specific region based on those range-keys.

3.2 Visualization of Weather Sensor Information

Figure 3 shows an application that demonstrates visualization of weather sensor information on Live E! project[2]. The observed data is stored in the key-value store continually by each deployed sensor node, and this application draws contour lines of weather information, e.g., temperature, humidity, pressure, by collecting related sensing information from the region of the screen.

[2] Live E!: http://www.live-e.org/en/

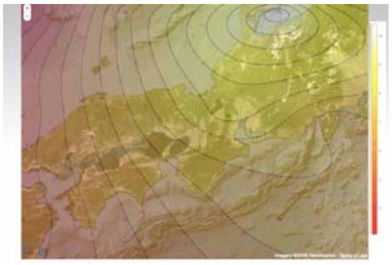


Fig. 3. Weather Sensor Information

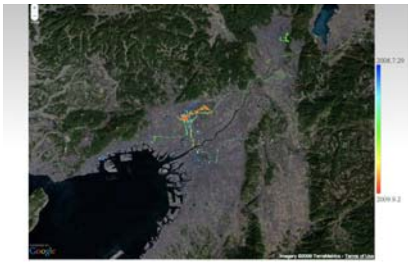


Fig. 4. Movement Histories

3.3 Visualization of Movement Histories

Figure 4 shows an application that demonstrates visualization of users' movement histories that can be obtained by GPS etc. The histories are stored in the key-value store by each user's node, and this application collects them that are located in the region of the screen.

4 Conclusion

In this paper, Range-key Skip Graph that enables nodes on P2P overlay network to retrieve data by range-to-range query was introduced. RKSG manages the inclusion keys list so that a node can retrieve any range that contains duplicate ranges among different nodes. To share several kinds of information in the distributed environment, key-value store was realized on RKSG that is easy to scale-out and enables the multiple nodes to handle huge amount of information. As a demonstration, two applications that utilize the store and require range query were implemented.

References

1. Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: ACM SIGCOMM 2001 (September 2001)
2. Maymounkov, P., Mazieres, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, p. 53. Springer, Heidelberg (2002)
3. Aspnes, J., Shah, G.: Skip Graphs. ACM Transactions on Algorithms 3(4), 37 (2007)
4. Ishi, Y., Teranishi, Y., Yoshida, M., Shimojo, S., Nishio, S.: A proposal of range-key extension of Skip Graph. IPSJ Technical Report 2009-DPS-139 (1), 1–7 (2009) (in Japanese)
5. Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. In: Dehne, F., Santoro, N., Sack, J.-R. (eds.) WADS 1989. LNCS, vol. 382, pp. 437–449. Springer, Heidelberg (1989)

Answering Range-Aggregate Queries over Objects Generating Data Streams

Marcin Gorawski and Rafal Malczok

Silesian University of Technology,
Institute of Computer Science,
Akademicka 16,
44-100 Gliwice, Poland
{Marcin.Gorawski,Rafal.Malczok}@polsl.pl

Abstract. Nowadays computer systems process various types of data such as images, videos, maps, data streams to name a few. In this paper we focus on a problem of answering range-aggregate queries over objects generating data streams. Our motivating example is a network of meters monitoring utilities consumption and continuously reporting the readings to central gathering points. An answer to a range-aggregate query is a merged stream of aggregates allowing analyses of utilities consumption in a given region. In order to calculate the answer we integrate MAL (Materialized Aggregates List) with spatial aggregating index, e.g. aR-Tree. The result we obtain is a spatial aggregating index with functionality of answering range queries over objects generating data streams. The index is embedded in an experimental stream data warehouse system implemented in Java. The implementation provided us with the possibility of presenting the index operation and also carrying out a number of tests.

1 Introduction

In recent years data warehouse systems become more and more popular. Users find the ability of processing large amounts of data in a short time very useful and convenient. This trend is supported by products of large software companies who enrich their business offer with ready-to-use data warehouse solutions integrated with broadly known database systems. A simple data warehouse can be created by a user who, by clicking, models data warehouse structure, builds dimensions, attributes and hierarchies and finally defines the available reports. Therefore, systems built from scratch and dedicated for a single company are no longer the only way to create a data warehouse.

The extension of the domain where data warehouse systems are applied, results in the need for supporting various, often far from standard, types of data. Very interesting are aspects of adapting data warehouse to be used for processing stream data. To the stream data category we classify, for example, car traffic and cell phones tracking data as well as utilities consumption data. For the purpose of storing and processing stream data stream data warehouses are being designed and implemented.

The area of stream data processing and storing is an active research field. There are many projects focused on designing systems which make possible to register and evaluate continuous queries [1]. Stream data warehouse systems pose many new challenges which do not occur in standard data warehouses. One of the most important is the problem concerning the data loading process (ETL – *Extract, Transform and Load*). In standard data warehouses ETL is a batch process launched from time to time (every night, every weekend etc) while in stream data warehouses the ETL process is a continuous one. On the other hand, changing nature of the ETL process forces stream data warehouses designers to provide efficient mechanisms for processing and managing stream data.

We started our research from a problem of answering range spatial queries over objects generating data streams. In this paper we first describe our motivating example. Then we address the issues of integrating MAL (Materialized Aggregates List), an extended version of a solution previously developed in our laboratory for processing long aggregates lists, with spatial aggregating index, e.g. aR-Tree. The result of integration is a spatial aggregating index with the functionality of answering range queries over objects generating data streams. Finally we present the system in which the index is implemented.

1.1 Motivating Example and Problem Definition

Stream data processing is, in many cases, motivated by the need of handing endless streams of sensor data [3]. The motivation for the research presented in this paper is a system of integrated utilities meters reading. The system monitors consumption of utilities such as water, natural gas and electrical energy.

Considering the telemetric system operation we can assume that every single meter is an independent source generating an endless stream of readings. A user who analyzes the system operation may be interested in the following aspects:

– What is the total number of meters located in a specified region (or regions) of the telemetric installation?
– How much utilities is consumed by inhabitants of the region (or regions)?

Let us consider the above as a query which answer consists of two parts. The first part provides information about the number of various kinds (water, gas etc.) of meters located in the query region. The second part is a merged stream (or streams) of aggregated readings flowing from the meters encompassed by the region query.

To answer the first part we can use a spatial index which, in very fast and efficient way, can answer any range query [2].

In order to calculate the second part, we need to apply an indexing structure which in index nodes stores aggregates concerning objects located in the nodes regions. The first solution proposed for this problem was aR-Tree [5] (aggregation R-Tree). aR-Tree index nodes located on the higher levels of the hierarchy store the number of objects on the nodes located in the lower levels.

The designers of the mentioned spatial aggregating indices assumed that the size of the aggregated data is well defined and small enough to fit into the computer main memory. Such an assumption cannot be made for a stream data. In the following section we present the extensions introduced to a spatial aggregating index which allow indexing objects generating data streams.

2 Answering Range Aggregating Queries

The main idea of our solution is to extend the existing functionality of aR-Tree (or any spatial aggregating index with similar features) providing a component that could be an integral part of every index node and would be able to process data streams of any length. Such a component is MAL [4] (Materialized Aggregates List).

MAL is a combination of a memory structure and a set of dedicated algorithms. MAL bases its operation on the concept of list and iterator. Mechanisms implemented in the list allow generating and then optionally materializing the calculated aggregates while iterators are used for browsing the generated data and communicating with the list.

When used as a component of indexing structure nodes, MAL must be able to merge aggregates streams. Aggregates streams merging operation merges two or more aggregates streams creating one aggregates stream. All merged streams must contain aggregates used during adding operation, merging cannot be performed for streams where some required aggregates are missing.

2.1 Spatial Aggregating Index Extensions

Processing data streams requires efficient solutions. The efficiency is regarded as both the time complexity of the applied algorithms and memory complexity of used data structures. Designing MAL we tried to cover all efficiency aspects: we used multithread implementations of page-filling algorithms and static table to store the aggregates. Hence we can be sure that, properly configured, a single instance of MAL will not consume too much system resources.

More problematic is the situation when there are multiple MALs in a spatial index. In order to control the number of iterator tables used by those instances we apply the concept of a resource pool (referenced below as table pool). We also developed a dedicated iterator tables assignment algorithm that manages the available tables to make the query evaluation process efficient.

The operation of the algorithm depends on the number of elements (nodes and objects) in FAN and M sets. The sets are created while the query answering algorithm browses the index and contain: index nodes entirely covered by the query region (*Full Access Nodes: FAN*) and single telemetric elements (meters) encompassed by the query region (*meters: M*).

If the table pool contains more than one iterator table, the nodes (FAN set) and objects (M set) involved in the answer stream generation process are sorted according to a set of below-listed criteria. Then, the available tables are assigned to the nodes and objects according to the sorting order: (1) the number of materialized data available for a given node, (2) generated stream materialization possibility and (3) the amount of objects encompassed by the node's region.

3 Presentation

The described solution is a part of DWE (Data Warehouse Experimental), a framework implemented in Java. DWE is equipped with graphical user interface. The user can see a map of telemetric system installation with utilities meters marked as points of different colors. Using a rubber rectangle user defines a set of regions which create spatial query. Another query parameter is the moment of time since when the aggregation should start. After sending the query to the system the user can observe the process of query evaluation. The observation includes changes of stream data sources (database, raw stream, materialized data), table pool statistics (the number of available tables, the number of usages of a single table) and continuously updated query results presented in the form of tables and charts.

Demonstration of the DWE system and the discussed solution is going to be preceded by a short presentation that would address the most important aspects of the research motivation and system assumptions. During the demonstration the working system would be presented with the special focus on the process of answering range aggregate queries over objects generating data streams.

References

1. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Popa [6], pp. 1–16
2. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R*-Tree: An efficient and robust access method for points and rectangles. In: Garcia-Molina, H., Jagadish, H.V. (eds.) SIGMOD Conference, pp. 322–331. ACM Press, New York (1990)
3. Bonnet, P., Gehrke, J., Seshadri, P.: Towards sensor database systems. In: Tan, K.-L., Franklin, M.J., Lui, J.C.-S. (eds.) MDM 2001. LNCS, vol. 1987, pp. 3–14. Springer, Heidelberg (2001)
4. Gorawski, M., Malczok, R.: On efficient storing and processing of long aggregate lists. In: Tjoa, A.M., Trujillo, J. (eds.) DaWaK 2005. LNCS, vol. 3589, pp. 190–199. Springer, Heidelberg (2005)
5. 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)
6. Popa, L. (ed.): Proceedings of the Twenty-first ACM SIGACT-SIGMOD- SIGART Symposium on Principles of Database Systems, Madison, Wisconsin, USA, June 3-5. ACM, New York (2002)

Real-Time Log Analysis Using Hitachi uCosminexus Stream Data Platform

Yoshiyuki Hayashida[1], Nobuhiro Ioki[1], Naomi Arai[1], and Itaru Nishizawa[2]

[1] Hitachi Software Division, [2] Hitachi Central Research Laboratory, Japan
{yoshiyuki.hayashida.cf,nobuhiro.ioki.yz,naomi.arai.ek, itaru.nishizawa.cw}@hitachi.com

Abstract. In this demo, we present real-time log analysis using Hitachi uCosminexus Stream Data Platform, uCSDP for short. Real-time log analysis is one of the key applications that offers preventive measures to detect irregular manipulations and human mistakes in system management, and reduces the risk and loss caused by such operations to the minimum in advance. uCSDP is the stream data processing system featuring its declarative query processing language, flexible time management, RAS support for high-available processing, and eager scheduling for ultra low latency processing. This demo highlights the uCSDP features for realizing real-time log analysis very easily and effectively.

Keywords: Stream data processing, Real-time Log analysis, CQL.

1 Introduction

Recently the quantity of data produced by IT systems has been increasing rapidly corresponding to the growth in large scaled complex computer systems, and diffusion of IC cards and sensor technology. As fusion of real world and IT accelerates business speed, revolution in the IT infrastructure is required. The stream data processing (SDP for short) system [1, 2] is a new IT platform responding to this requirement. Main aspect of SDP is an ability to calculate a large amount of data in real-time manner by processing them sequentially in memory.

Hitachi uCosminexus Stream Data Platform, uCSDP for short, provides a SDP engine, and supports a declarative continuous query language based on CQL [3]. uCSDP has several extended features: flexible time management, RAS support for high-available processing, and eager scheduling for ultra low latency processing.

Real-time log analysis is one of the promising applications of SDP system. IT systems compute a large quantity of data including their own log files generated by themselves. Although the log files contain the business process information and system status records that give the precious information for detecting system status and user misoperations, they are usually only stored in the storage system and are not effectively used in the absence of those IT platforms which analyze them in real-time manner so far. Considering this demand, SDP system is far more suited to handle this task compared to traditional database management systems.

This paper explains the features of uCSDP in Section 2, describes how to analyze log files in real-time using uCSDP in Section 3, and concludes this paper in Section 4.

2 Basic and Extended Features of Hitachi uCSDP

2.1 Declarative Query Language and Flexible Time Management

Hitachi uCSDP models stream as an unbounded bag of pairs of tuple and timestamp, and models relation as time-varying bag of tuples, and offers a general-purpose declarative continuous query language based on CQL [3].

uCSDP adopts two timestamp handling modes (server timestamp and application timestamp) to support wide variety of applications. In server timestamp mode, the timestamp of stream tuple is determined at the time when the tuple arrives at uCSDP. In application timestamp mode, uCSDP uses the timestamp embedded in the stream tuple itself. The user can select an appropriate mode suitable for the application.

In addition to the timestamp handling modes, the user can set time “range” for event time adjustment. uCSDP adjusts out-of-order events if input data are in the “range”. This feature is necessary to collect data from different data sources.

2.2 High Available Data Processing

High availability mechanism is essential to the real business systems. uCSDP is equipped with RAS supporting features such as application tracing, tuple logging, processing time monitoring, queue monitoring, and partial blockage to ensure high availability.

In the SDP systems, burst data input to a time-based sliding window causes the huge memory consumption that may cause system instability. uCSDP provides “time resolution management” mechanism in order to avoid huge memory consumption. The mechanism pre-calculates summary value for each sub-window and generates query results using the pre-calculated summary values. With this mechanism, uCSDP reduces the memory consumption without any approximation technique such as sampling [4] for burst data input to the time-based sliding window aggregation queries, and it greatly contributes to improvement of system stability.

2.3 Eager Scheduler for Ultra Low Latency Processing

For actual applications, a certain number of CQL queries are needed to describe complicated business logics. Data processing latency to process data for such business logics increases in the manner of traditional round robin scheduler.

To cope with this problem, uCSDP adopts eager scheduler and queue management mechanism. It concatenates queries, removes intermediate queues between queries, and schedules each stream tuple processing execution starting from the input and ending with the output of uCSDP to minimize data transfer overhead.

3 Real-Time Log Analysis Demo Description

3.1 Demo Overview and Configuration

Fig. 1 shows the real-time log analysis demo system consisting of uCSDP, input and output adapters, and the user interface component written in Java. It detects improper use of an IT system by monitoring and analyzing Web access logs in large quantities to find out symptoms of system anomalies and unusual user access patterns immediately from log data streams. As a demo platform, we use Red Hat Enterprise Linux 5.1 on Pentium 4 (1.2GHz) PC with 1GB memory. The inputs are Web proxy and certification log files. The input adapter converts the log files to stream tuples and sends them to uCSDP. uCSDP outputs the analysis results as streams. Then the output adapter converts the results and sends them to the system manager's dashboard GUI.

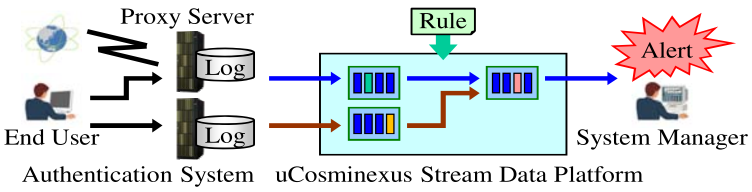
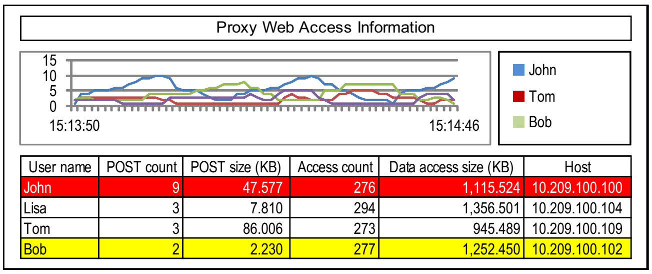


Fig. 1. Real-time Log Analysis System Configuration

3.2 Demo Scenario

In this demo, we apply a simple rule considering that POST submissions with a certain pattern within a short period might include illegal Web accesses, for example, three trials of more than five POST operations in ten minutes. Fig. 2 displays a GUI snapshot of a monitoring result. The graph area displays the frequency of POST operations and the table area shows detailed access histories with warning colors that let the system manager know which users satisfy the scenario conditions.



User name	POST count	POST size (KB)	Access count	Data access size (KB)	Host
John	9	47.577	276	1,115.524	10.209.100.100
Lisa	3	7.810	294	1,356.501	10.209.100.104
Tom	3	86.006	273	945.489	10.209.100.109
Bob	2	2.230	277	1,252.450	10.209.100.102

Fig. 2. Real-time Log Analysis Monitoring GUI

3.3 Merit of Applying uCSDP to Real-Time Log Analysis

In the past, after a problem occurred, the log files were used as materials to investigate causes. Real-time log analysis using uCSDP, however, enables detection of the problem cause at the same time with the problem occurrence from quantities of log information including Web proxy logs and certification logs.

By defining the analysis scenario in CQL, the system manager does not have to care about lower layer implementations. Analysis conditions including parameters and thresholds can be modified easily and flexibly, so that the system manager can concentrate on system operations.

In real situations, the event time management feature of uCSDP described in Section 2.1 is necessary because the logs from different servers arrive at the analysis system in out-of-order. In addition to the processing efficiency, high availability mechanism is the key to applying a new data processing system to the real business situation. We believe that RAS features of uCSDP mentioned in Section 2.2 are powerful enough to realize it.

4 Conclusion

We have presented real-time log analysis system using our stream data processing platform uCSDP. Real-time log analysis system is one of the key applications that offers preventive measures of detecting irregular manipulations and human mistakes at system management, and reduces the risk and loss caused by such operations to the minimum in advance also. We believe that uCSDP offers stream data processing features not only realizing real-time log analysis described in this paper but also realizing other real-time data processing applications.

References

1. Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G., Olston, C., Rosenstein, J., Varma, R.: Query Processing, Resource Management, and Approximation in a Data Stream Management System. In: Proc. of CIDR 2003 (2003)
2. Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, S.: Monitoring streams: a new class of data management applications. In: Proc. of the 28th VLDB (2002)
3. Arasu, A., Babu, S., Widom, J.: The CQL Continuous Query Language: Semantic Foundations and Query Execution. The VLDB Journal 15 (2006)
4. Babcock, B., Datar, M., Motwani, R.: Sampling From a Moving Window Over Streaming Data. In: Proc. of SODA 2002 (2002)

Anddy: A System for Author Name Disambiguation in Digital Library

Jia Zhu, Gabriel Pui Cheong Fung, and Xiaofang Zhou

School of ITEE, The University of Queensland, Australia
{jiazhu,zxf,uqgfung}@itee.uq.edu.au

Abstract. In this demonstration, we implement a system called Anddy that tries to disambiguate author names in digital library by clustering. Our system will try to cluster the citation records based on some user input parameters such that, ideally, each cluster should contain citation records that belong to the same author, meanwhile different clusters denote different authors. For each cluster, we will further display a taxonomy representing it and the social network that the cluster has.

Keywords: Clustering, Name Disambiguation, Taxonomy.

1 Introduction

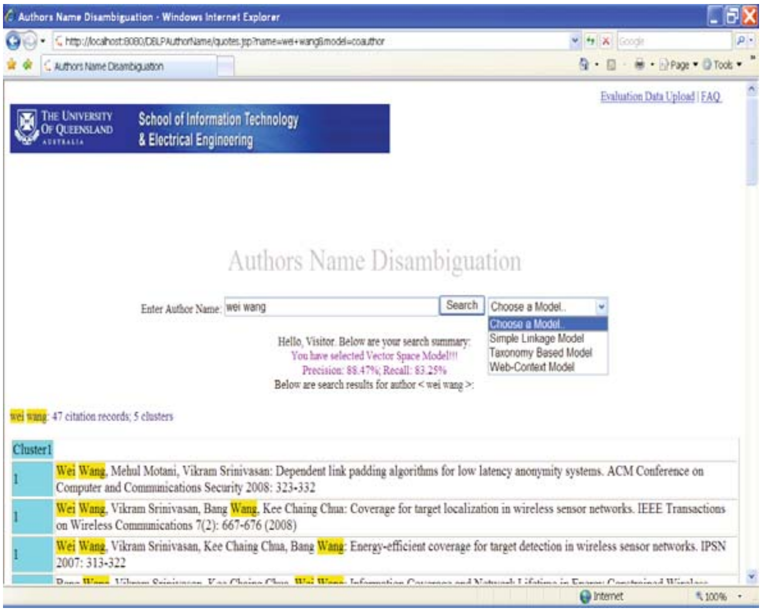
In digital library, it is very common that several authors share the same name because name is not an unique identifier. For instance, if we search "Wei Wang" in DBLP [1], more than 60 different "Wei Wang" are mixed in the same page, and DBLP cannot disambiguate these 60 different "Wei Wang" successfully. This will make the users feel very inconvenient because it is difficult for them to identify which records belong to which author.

In this demo, we present a system called Anddy (**A**uthor **N**ame **D**isambiguation in **D**igital Librar**y**) to group the citation records that refer to the same author. Fig. 1 shows a sample interface of our system. For example, users can input an author name, e.g. "Wei Wang", in the search field and select the model that they want to use to cluster the records. Our system will then try to cluster the citation records based on the input parameters (the author name and a user selected clustering model) such that, ideally, each cluster should contain citation records that belong to the same author, meanwhile different clusters denote different authors. In addition, our system will further display some advanced and useful information for each cluster, such as the linkage among different authors, a graphical taxonomy to show the linkage among different records within the same cluster, etc.

2 System Description

Anddy consists of two parts: a clustering framework and a user interface. As shown in Fig. 2, the clustering framework takes the citation records need to be identified as input

[1] http://www.informatik.uni-trier.de/ ley/db/


Authors Name Disambiguation - Windows Internet Explorer
http://localhost:8080/DBLPAuthorName/quotes.jsp?name=wei+wang&model=coauthor
Authors Name Disambiguation
Page
Tools
Evaluation Data Upload | FAQ
The University Of Queensland Australia
School of Information Technology & Electrical Engineering
Search
Choose a Model..
Simple Linkage Model
Taxonomy Based Model
Web-Context Model
Internet
100%

Authors Name Disambiguation

Enter Author Name: wei wang

Hello, Visitor. Below are your search summary:
You have selected Vector Space Model!!!
Precision: 88.47%; Recall: 83.25%
Below are search results for author < wei wang >:

wei wang: 47 citation records; 5 clusters

Cluster1	
1	Wei Wang, Mehul Motani, Vikram Srinivasan: Dependent link padding algorithms for low latency anonymity systems. ACM Conference on Computer and Communications Security 2008: 323-332
1	Wei Wang, Vikram Srinivasan, Bang Wang, Kee Chaing Chua: Coverage for target localization in wireless sensor networks. IEEE Transactions on Wireless Communications 7(2): 667-676 (2008)
1	Wei Wang, Vikram Srinivasan, Kee Chaing Chua, Bang Wang: Energy-efficient coverage for target detection in wireless sensor networks. IPSN 2007: 313-322

Fig. 1. The Interface of Anddy

and perform clustering based on a user selected clustering model. The user interface accepts the user input and presents the clustering results to the user. We will describe these two components in this section.

2.1 Clustering Framework

We have implemented three different clustering models in Anddy and allows users to choose using which of the clustering models to cluster the citation records. The three models that we have implemented are: simple linkage model, taxonomy based model and web-context model. We implemented these three models because they are some of the most popular models nowadays.

Simple Linkage Model

In this model [3], it tries to link all the citation records based on graph theory. Specifically, given a graph G, each vertex v is a citation record, and each edge e between two vertices represents the similarity of two vertices (i.e. the similarity between two citation records). The similarity is calculated based on the token strings in three attributes: (1) authorships, (2) paper title and (3) publication venue. If the similarity is high enough, then these two records refer to the same entity.

Taxonomy based Model

In our early work [4], a model called taxonomy based model is proposed to enhance the name disambiguation issue by enriching the information from citation records. Taxonomy

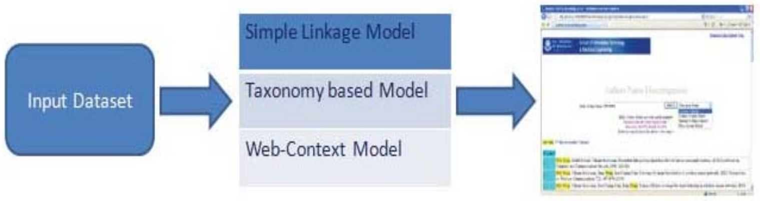


Fig. 2. Clustering Framework

is a data structure that can express the relationship among different terminologies [1]. We construct some term-based taxonomies based on terms' lexical similarity and their co-occurrences in the paper titles. In these taxonomies, each node is a term extract from paper titles and the linking between two nodes is the relationship of them (their similarities). Once we have the taxonomies, we can calculate the similarity among clusters by exploring the links in these taxonomies. For example, assume a record A contains a list of terms T_A, and a record B contains a list of terms T_B. If T_A and T_B have heavy linking according to the taxonomies, then record A and B might refer to the same entity.

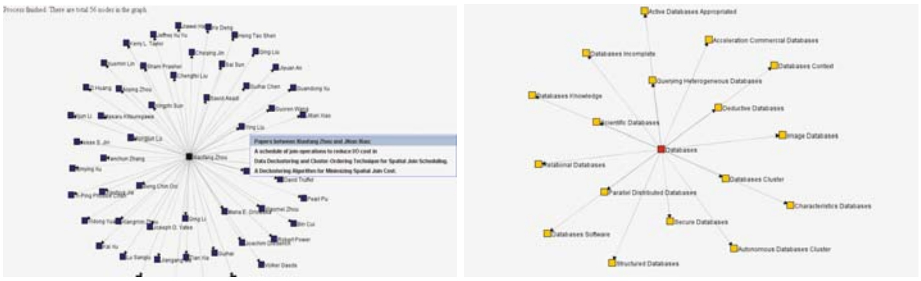
Web-Context Model

This model [2] tries to retrieve more information from the web automatically so as to identify whether two citation records belong to the same author. This model is different from the above two models because it does not only consider the information within a single domain, but further extract information from some other external domains. Yet, information like email and affiliation, although can usually disambiguate the citation records correctly, are difficult to retrieve because of the unstructured nature of web pages. In this demo, we implement a model to analyze the URLs return from search engine and determine if two records refer to the same entity or not. For example, assume there are two records need to be identified, A and B, and each record has a list of URLs returned from search engine after input the paper title as query. If there are high overlap in their URLs, then these two records may refer to the same author. Obviously, there are some digital libraries web pages should be ignored, e.g. Citeseer [2], because these sites contain records in one page but it does not mean these records refer to the same author.

2.2 User Interface

Our system is a web based application and implemented by Java/JSP. The interface includes: (1) A search bar which allow users to search authors' citation records; (2) An drop down box which allows users to choose different clustering models to cluster the citation records; (3) A results page which displays records that has been assigned to different cluster, so that each cluster, ideally, should contain citation records that belong to the same author, meanwhile different clusters should refer to different author; (4) An advanced information section in the page which displays the authors' network (Fig. 3) and the term-based taxonomy related to the cluster (Fig. 3).

[2] http://citeseer.ist.psu.edu/



Authors Network Term-based Taxonomy

Fig. 3. Graphical Authors Network and Term-based Taxonomy

3 Demonstration Overview

Our demonstration uses a real-world dataset extract from DBLP. In the demonstration, we will show how Anddy works and how a user can employ it to efficiently search citation records by author' name. Specifically, we will: (1) demonstrate the process from submit a query, select a clustering model to obtain a clustering result; (2) show how important and the practical usefulness of the graphical authors' network and the term-based taxonomy.

References

1. Bast, H., Durpret, G., Piwowarski, B.: Discovering a term taxonomy from term similarities using principal component analysis. In: Semantics, Web and Mining, pp. 320–331 (2006)
2. Jiang, J.Y., Lee, H.M., Yang, K.H., Peng, H.T., Ho, J.M.: Author name disambiguation for citations using topic and web correlation. In: Research and Advanced Technology for Digital Libraries, pp. 185–196 (2008)
3. Yin, X.X., Han, J.W.: Object distinction: Distinguishing objects with identical names. In: IEEE 23rd Int. Conf. on Data Engineering, pp. 1242–1246 (2007)
4. Zhu, J., Fung, G.P.C., Zhou, X.F.: A term-based driven clustering approach for name disambiguation. In: Proc. Joint. APWeb/WAIM, pp. 320–331 (2009)

A System for Keyword Search on Hybrid XML-Relational Databases Using XRjoin

Liru Zhang, Tadashi Ohmori, and Mamoru Hoshi

Graduate School of Information Systems, The University of Electro-Communications
1-5-1, Chofugaoka, Chofu City, Tokyo, Japan
{zhangliru,omori}@hol.is.uec.ac.jp

Abstract. With storing XML data as a type of a column, relational databases have become more powerful. A relational database including both XML data and relational data is termed a *hybrid XML-Relational database (XML-RDB)* in this paper. Because existing keyword-search techniques on either relational databases (RDB) or XML databases (XML DB) cannot get appropriate results, we propose a new method of keyword search on XML-RDB to get more reasonable results than existing methods. To realize it, a new join operator, named *XRjoin*, is designed and utilized to join XML data with relational data. We construct a demo system by using DB2 v9.5, and our experiments show that the system can find the answers that we want to get.

1 Introduction

Keyword search is a popular function of databases to enable users to extract information under databases without any knowledge of the schema or query languages. Since major relational databases management systems have allowed the residence of XML data in relations, there is increasing need for users to retrieve both XML and relational data by keyword search. On a hybrid XML-RDB such as IBM DB2 9.5 [5], Microsoft SQL Server 2008, applying the keyword search is a challenging task, because keyword-search techniques on RDB widely differ from those on XML DB.

There are several studies on keyword search over structured databases such as DBXplorer [1], DISCOVER [4] on RDB, XRANK [2] on pure XML DB. However, XRANK cannot get appropriate results if the result is composed of XML subtrees related by various relational linkages. DBXplorer does not consider XMLs and cannot retrieve the information obeying the hierarchy of XML. DISCOVER does join by any relationship, but the maximum Candidate Network's size T limits its ability to retrieve the subtrees of XMLs having any depth or heterogeneity. To overcome the problem of retrieving subtrees of schema-less XMLs related by any linkage, we propose a new keyword-search method on XML-RDB. A new join operator, *XRjoin*, uses the SQL/XML query language to do join between XML and relational data. This paper demonstrates how XRjoin works in practice and gives reasonable samples to approve our hybrid XML-RDB system useful.

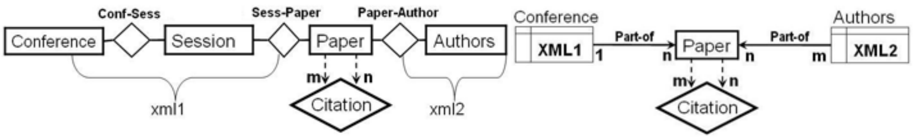
2 Our Proposal on a Hybrid XML-RDB

We design a new schema for a hybrid XML-RDB, based on the ER model of a RDB which stores the corresponding information of DBLP.xml.

Fig. 1(a) shows that the RDB has four entities “Conference”, “Session”, “Paper”, “Authors”, and four relationships “Conf-Sess”, “Sess-Paper”, “Paper-Author”, “Citation”. In this case, the XML data have been decomposed into several tables.

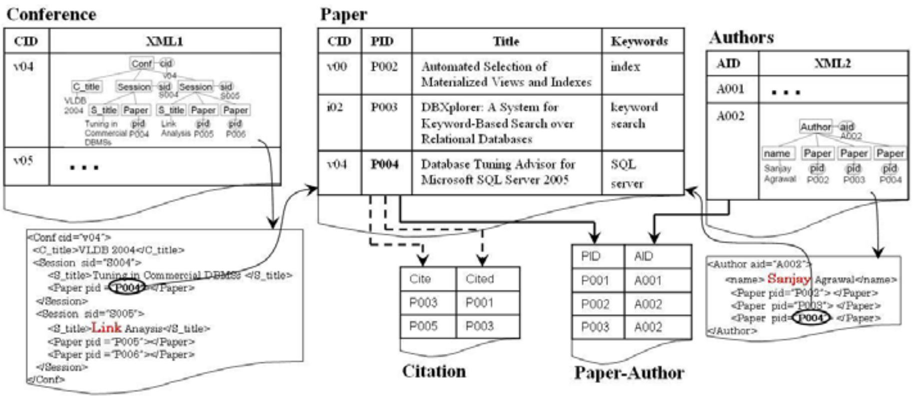
For obtaining useful subordinate information of XML, a hybrid XML-RDB is utilized to contain XML without any change of its format. Fig. 1(b) shows the schema of a hybrid XML-RDB. This schema includes three entities “Conference”, “Paper”, “Authors”, and two “part-of” relationships. “Conference” and “Authors” are *hybrid entities* storing XML data. An instance of “XML1” in “Conference” is presented in the left of Fig. 2, which has the “Conf-Session-Paper” hierarchy. In the right of Fig. 2, “XML2” in “Authors” has the “Author-Paper” hierarchy. In both of “XML1” and “XML2”, the information of “Paper” is described by “PID” only.

The processes of keyword search on XML-RDB are composed of the following four steps: (1) identify entities hit by keywords, (2) enumerate join-trees, (3) generate statements, (4) execute these statements and obtain results. The entities including keywords are identified by searching the auxiliary tables (details in [3]). Based on these entities, the system enumerates minimum-cost Steiner-trees that contain all keywords in the schema-graph. These trees are called *join-trees*. If there is a hybrid entity in a join-tree, our proposed *XRjoin* (section 3), which is an operator to join XML with relational data, is needed. A join-tree decides how to join between these



(a) The ER (Entity-Relationship) model of a RDB (b) The schema of a hybrid XML-RDB

Fig. 1. Data model



CID	XML1
v04	
v05	. . .

```
<Conf cid="v04">
 <C_title>VLDB 2004</C_title>
 <Session  sid="S004">
    <S_title>Tuning in Commercial DBMSS </S_title>
    <Paper pid ="P004"></Paper>
 </Session>
 <Session  sid="S005">
    <S_title>Link Anaysis</S_title>
    <Paper pid ="P005"></Paper>
    <Paper pid ="P006"></Paper>
 </Session>
</Conf>
```

CID	PID	Title	Keywords
v00	P002	Automated Selection of Materialized Views and Indexes	index
i02	P003	DBXplorer: A System for Keyword-Based Search over Relational Databases	keyword search
v04	**P004**	Database Tuning Advisor for Microsoft SQL Server 2005	SQL server

AID	XML2
A001	. . .
A002	

Cite	Cited
P003	P001
P005	P003

PID	AID
P001	A001
P002	A002
P003	A002

```
<Author aid="A002">
   <name> Sanjay Agrawal</name>
    <Paper pid="P002"> </Paper>
    <Paper  pid="P003"> </Paper>
    <Paper  pid="P004"> </Paper>
</Author>
```

Fig. 2. A fragment of our experimental XML-RDB

entities. For one join-tree, the statements for natural join or XRjoin are generated and executed automatically. If the results exist, they will be sent to the user interface.

3 XRjoin

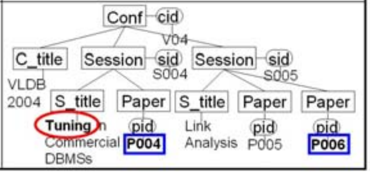
XRjoin is designed to extract relevant information between XML data and relational data. We present an illustration of concrete contents to explain how XRjoin works.

Fig. 3 shows several tuples of "Paper", and an XML in "Conference". The keyword "tuning" exists in "S_title" of the session **SID:S004** in "XML1". The keyword "SQL" exists in "Title" of two tuples **PID:P004** and **PID:P006** in "Paper".

XRjoin ((Conference, "tuning"), (Paper, "SQL")) is an operator to extract relevant hybrid information from "tuning"-related XMLs in "Conference" and "SQL"-related tuples in "Paper". It extracts the LCA (Least Common Ancestor) between the element **SID:S004** (including "tuning") and one of the paper elements **PID:P004** and **PID:P006** (including "SQL" as a paper). The subtree from the LCA appears as a new XML in *CXML* column of *T* (the resulting table of XRjoin in Fig. 4). Besides *CXML*, related relational data (*PID, Title, Keywords*) in *Paper* also appears as additional columns of *T*. As a result, **the first hybrid tuple** in Fig. 4 means that a paper including "SQL" belongs to a session including "tuning". **The second hybrid tuple** means that a paper including "SQL" belongs to a conference that has a session including "tuning".

Formally, when a keyword *K1* exists in a hybrid entity *X* and when another keyword *K2* exists in a relation entity *E*, *XRjoin ((X, K1), (E, K2))* is defined as a set of new tuples [*e*, *LCA-T(e_id,K1) on x*] for all *e* in *E* and all *x* in *X*, such that

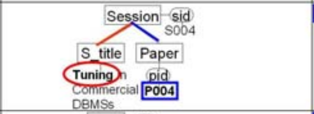
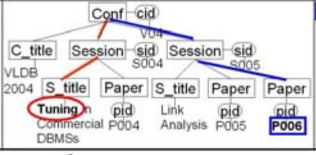
Conference "tuning"

CID	XML1
V98	...
V04	 Conf cid V04 C_title VLDB 2004 Session sid S004 Session sid S005 S_title Tuning in Commercial DBMSs Paper pid P004 S_title Link Analysis Paper pid P005 Paper pid P006

Paper "SQL"

CID	PID	Title	Key words
V98	...	...	...
V04	P004	Database Tuning Advisor for Microsoft SQL Server 2005	...
V04	P005	ObjectRank: Authority-Based Keyword Search in Databases	...
V04	P006	Efficient XML-to-SQL Query Translation	...

Fig. 3. The instances of the hybrid entity "Conference" and the relation "Paper"

CID	CXML	PID	Title	Keywords
V04	 Session sid S004 S_title Tuning in Commercial DBMSs Paper pid P004	P004	Database Tuning Advisor for Microsoft SQL Server 2005	...
V04	 Conf cid V04 C_title VLDB 2004 Session sid S004 Session sid S005 S_title Tuning in Commercial DBMSs Paper pid P004 S_title Link Analysis Paper pid P005 Paper pid P006	P006	Efficient XML-to-SQL Query Translation	...

"tuning" "SQL"

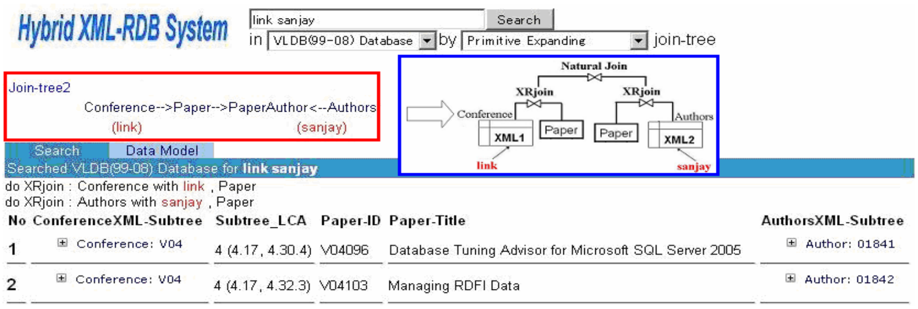
Fig. 4. The result ***T*** of *XRjoin ((Conference, "tuning"), (Paper, "SQL"))*

- *e* is a tuple satisfying *K2* in *E*; let *e_id* be the tuple-id representing *e* in *X*;
- *LCA-T (e_id, K1) on x* is a subtree in the XML of *x* whose root element is the LCA for the element *e_id* and an element satisfying *K1*.

Namely, the *XRjoin ((X, K1), (E, K2))* retrieves the XML information satisfying *K1* (in *X*) which further contains a relational data-item satisfying *K2* (in *E*).

Demonstration: We will demonstrate a keyword search engine on a subset of DBLP which includes 10 years information of VLDB (1999-2008). On this hybrid XML-RDB, join operations for keyword search can be finished in 5 seconds at most. Our proposed system (details in [3]) will show its efficiency at the demo scene. The system can interactively find appropriate and ordered XML-relational hybrid information which cannot be obtained by XRANK [2] or DBXplorer [1].

At the end of this paper, a snapshot of demo system is given in Fig. 5. This is an example when keywords "link" and "sanjay" are inputted. The original data about "link" and "sanjay" in this hybrid XML-RDB are shown in Fig. 2. The results are derived from a join-tree (in the left box) which does XRjoin twice and natural join once (in the right box). These results are listed by ascending order of the sum of two subtrees' scores that are computed when XRjoin works. The answer of No.1 in Fig. 5 means that an author named "sanjay" has written a paper in the conference VLDB2004, which has a session hit by "link" (Click the plus sign to view the detail of XML in this interface). The answers cannot be obtained by existing techniques.



Searched VLDB(99-08) Database for **link sanjay**
do XRjoin : Conference with link , Paper
do XRjoin : Authors with sanjay , Paper

No	ConferenceXML-Subtree	Subtree_LCA	Paper-ID	Paper-Title	AuthorsXML-Subtree
1	⊞ Conference: V04	4 (4.17, 4.30.4)	V04096	Database Tuning Advisor for Microsoft SQL Server 2005	⊞ Author: 01841
2	⊞ Conference: V04	4 (4.17, 4.32.3)	V04103	Managing RDFI Data	⊞ Author: 01842

Fig. 5. A snapshot of the demo system

References

1. Agrawal, S., Chaudhuri, S., Das, G.: DBXplorer: A System for Keyword-Based Search over Relational Databases. In: ICDE, pp. 5–16. IEEE, San Jose (2002)
2. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: XRANK: Ranked Keyword Search over XML Documents. In: SIGMOD, pp. 16–27. ACM, California (2003)
3. Zhang, L., Ohmori, T., Hoshi, M.: Keyword Search on Hybrid XML-Relational Databases By using XRjoin. In: Kitagawa, H., Ishikawa, Y., Li, Q. (eds.) DASFAA 2010, Part II. LNCS, vol. 5982, pp. 292–298. Springer, Heidelberg (2010)
4. Hristidis, V., Papakonstantinou, Y.: DISCOVER: Keyword search in relational databases. In: VLDB, pp. 670–681. Morgan Kaufmann, Hong Kong (2002)
5. IBM DB2 Database Information Center, http://publib.boulder.ibm.com/infocenter/db2luw/

MediaMatrix: A Video Stream Retrieval System with Mechanisms for Mining Contexts of Query Examples

Shuichi Kurabayashi and Yasushi Kiyoki

Graduate School of Media and Governance, Keio University
5322 Endoh, Fujisawa, Kanagawa, 252-0816, Japan
{kurabaya,kiyoki}@sfc.keio.ac.jp

Abstract. This paper presents a context-based whole video retrieval system that retrieves an entire video stream, not just individual scenes in videos. The main feature of this system is a novel query processing paradigm that involves data mining techniques to extract a query context, which is expressed as a combination of multiple videos. A query for this system consists of an example video and a context video. The example video is for representing the color-emotions hidden within video content, which are extracted using methods developed for color psychology research. The context video is for performing a context relevance computation between the example video and videos in a database. This paper demonstrates an implemented system that includes a SQL-like video stream query language and mechanisms for visualizing emotive contexts of a video stream.

Keywords: video search engine, context-computing, query language.

1 Introduction

As video data becomes prevalent on the Web, the result of a video search engine query can easily exceed the practical limits on the time a user has to watch to them. Despite the fact that many video-search engines have been provided for this purpose, extensive manual tweaking and heuristic trial-and-error is still required for users to find their desired videos. This is primarily because user-generated videos have insufficient and/or unreliable metadata in the form of titles and tags. It is increasingly important to utilize the concept of context-dependent database computations for retrieving video data according to emotive contexts[1].

This paper presents a *context-based whole video retrieval system*[2], called *MediaMatrix*. This system analyzes the color-emotions hidden within video content using methods developed for color psychology research. As shown in Fig.1, a query for this system consists of an example video and a context video. The example video is for representing the color-emotions of video content. The context video is for performing a context relevance computation between the example video and videos in a database. This computation evaluates similarities between the example video and context video, for extracting which context of the video is most relevant to the user.

Such contexts as dominance, prominence, and hierarchical granularity of color-emotions might be selected. The system realizes new ways of querying artistic, visually rich and dynamic video content such as television dramas, animation, and movies – something not possible when limited to just a few words. This is done by inspecting the dynamic, temporally evolving contexts of video data. Movies are a prime example of media in dire need of new retrieval mechanisms for returning an entire video stream according to the query contexts. This is because the performance of a movie in its entirety is what users search for, as opposed to independent fragments which lose their meaning when separated from the whole.

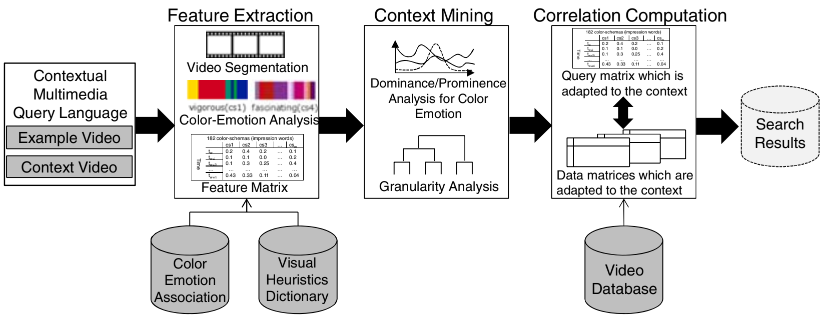
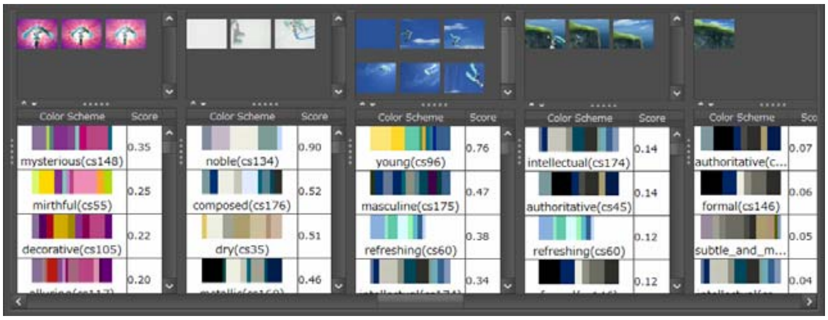

Contextual Multimedia Query Language
Example Video
Context Video
Feature Extraction
Video Segmentation
vigorous(cs1) fascinating(cs4)
Color-Emotion Analysis
182 color-schemas (impression words)
Feature Matrix
Context Mining
Dominance/Prominence Analysis for Color Emotion
Granularity Analysis
Correlation Computation
Query matrix which is adapted to the context
Data matrices which are adapted to the context
Search Results
Color Emotion Association
Visual Heuristics Dictionary
Video Database

Fig. 1. Overview of the context-based whole video retrieval system



Color Scheme	Score
mysterious(cs148)	0.35
mirthful(cs55)	0.25
decorative(cs105)	0.22
	0.20

Color Scheme	Score
noble(cs134)	0.90
composed(cs176)	0.52
dry(cs35)	0.51
	0.46

Color Scheme	Score
young(cs96)	0.76
masculine(cs175)	0.47
refreshing(cs60)	0.38
	0.34

Color Scheme	Score
intellectual(cs174)	0.14
authoritative(cs45)	0.14
refreshing(cs60)	0.12
	0.12

Color Scheme	Sco
authoritative(c...	0.07
formal(cs146)	0.06
subtle_and_m...	0.05
	0.04

Fig. 2. The system analyzes emotive contexts by computing the color-emotion correlation of every video frame

2 MediaMatrix System Implementation

The core component of the MediaMatrix system is a context-based similarity video search engine with dynamic video content analysis mechanisms. This system analyzes the color-emotions hidden within video content by referencing color-emotion definitions developed in the course of color psychology research [3, 4]. Each color-emotion, which corresponds to a specific emotional perception of humans, consists of 120 chromatic colors and 10 monochrome HSV colors defined in "Color Image Scale [3]". The system extracts the color-emotion for each video frame by computing the correlation between 182 color schema and 130 HSV colors. Fig.2 shows a color-emotions analysis result visualization computed by the implemented system. In the query execution process, the system extracts the following three query contexts (CX1, CX2, and CX3), which are expressed as a combination of videos. (CX1) Dominance / (CX2) Prominence of Color-Emotions: the system provides LEF-IGEF (Local Emotion Frequency – Inversed Global Emotion Frequency) operator[2] for selecting color-emotions by computing the specificity of color-emotions to each frame throughout the entire video stream. The LEF-IGEF operator increases its prominence proportionally to the appearance ratio of a color-emotion in a video frame, but is offset by the frequency of the color-emotion in the entire video. Fig. 3 shows a visualization of the dominance of color-emotions and Fig.4 shows a visualization of prominence of color-emotions and the extracted prominent scenes. (CX3) Granularity of Color-Emotions: since one single video may contain different meanings at multiple granularity levels, the system applies the hierarchical agglomerative clustering method with UPGMA for recognizing the granularity of a video as shown in Fig.5.

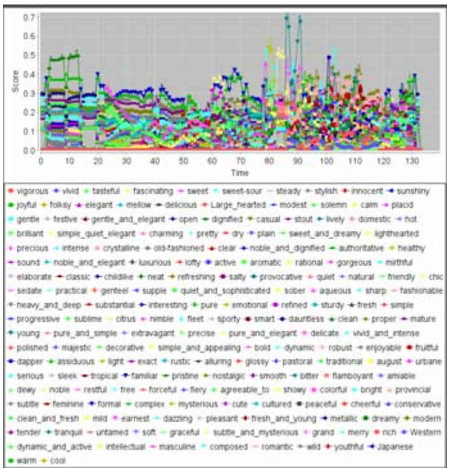


Fig. 3. A visualization of the dominance (appearance ratio) of a color-emotion in the frames (vertical axis: the correlation score of each color-emotion, horizontal axis: the timeline of the video data stream)

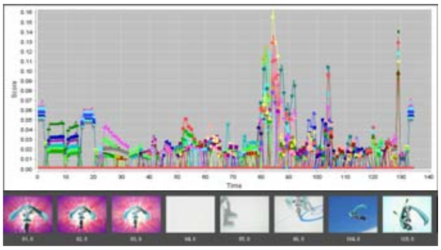


Fig. 4. A visualization of the prominence of color-emotions and the extracted prominent scenes. A high score means that the color-emotion has strong significance

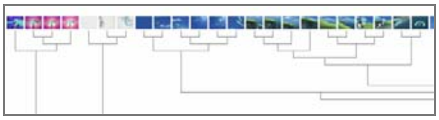
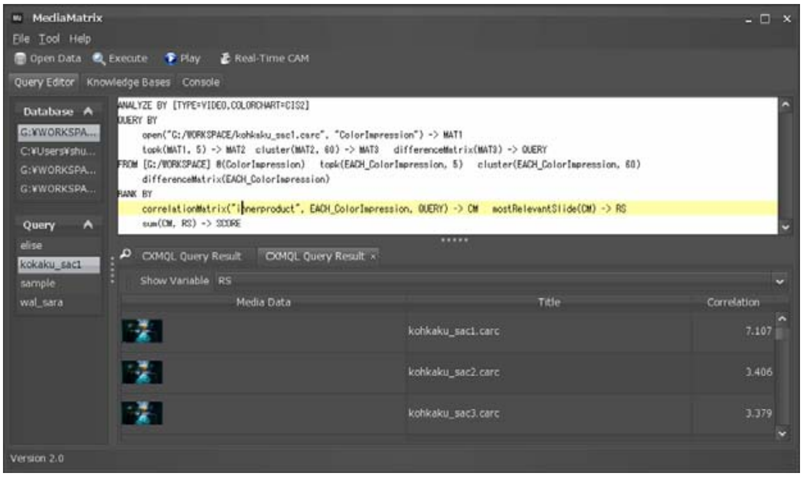


Fig. 5. The system applies the hierarchical agglomerative clustering method for analyzing the granularity of color-emotions

3 System Demonstration

Fig.6 shows an integrated video search environment with a structured query language for video. To demonstrate the proposed system, we use 128 Japanese animation videos. Each video file is 24 minutes in duration. The total duration of the video files is

51 hours, and the total size is 6.8 gigabytes. Each video is accompanied by a text annotation written by its publisher. To evaluate the performance, we have compared our method with a traditional keyword-based video retrieval provided by video hosting services. Overall, our approach can achieve an average precision of 56.1% which is about 50% better than that of keyword-based retrieval systems (5.2%).



```
ANALYZE BY [TYPE=VIDEO,COLORCHART=CIS2]
QUERY BY
    open("G:/WORKSPACE/kohkaku_sac1.carc", "ColorImpression") -> MAT1
    topk(MAT1, 5) -> MAT2  cluster(MAT2, 60) -> MAT3   differenceMatrix(MAT3) -> QUERY
FROM [G:/WORKSPACE] #(ColorImpression)   topk(EACH_ColorImpression, 5)   cluster(EACH_ColorImpression, 60)
    differenceMatrix(EACH_ColorImpression)
RANK BY
    correlationMatrix("innerproduct", EACH_ColorImpression, QUERY) -> CM   mostRelevantSlide(CM) -> RS
    sum(CM, RS) -> SCORE
```

Media Data	Title	Correlation
	kohkaku_sac1.carc	7.107
	kohkaku_sac2.carc	3.406
	kohkaku_sac3.carc	3.379

Fig. 6. The integrated video search environment with a structured query language for video

4 Conclusion and Future Work

This paper has proposed MediaMatrix, a system that provides a novel query language for allowing users to express and characterize their queries by using existing video clips as contexts. Its capabilities enable new ways of querying artistic, visually rich and dynamic video content such as television dramas, animation, and movies – something which is not possible by using just a few words. As future work, we are expanding the scalability of the system for performing real-time video analysis.

References

1. Kiyoki, Y., Kitagawa, T., Hayama, T.: A metadatabase system for semantic image search by a mathematical model of meaning. SIGMOD Rec. 23, 34–41 (1994)
2. Kurabayashi, S., Ueno, T., Kiyoki, Y.: A Context-Based Whole Video Retrieval System with Dynamic Video Stream Analysis Mechanisms. In: 11th IEEE International Symposium on Multimedia, San Diego, USA, pp. 505–510 (2009)
3. Kobayashi, S.: Color Image Scale. Oxford University Press, USA (1992)
4. Valdez, P., Mehrabian, A.: Effects of color on emotions. Journal of Experimental Psychology: General 123, 394–409 (1994)

Application Developments in Mashup Framework for Selective Browsing

Takakazu Nagamine and Tomio Kamada

Dept. of Computer Sci. and System Eng., Kobe University
{nagamine,kamada}@cs26.scitec.kobe-u.ac.jp

Abstract. We are developing a new mashup framework for creating flexible applications in which users can selectively browse through mashup items. The framework provides GUI components called widgets through which users can browse mashed-up data selectively, and the system processes demand-driven creation of mashed-up data upon receiving access requests through widgets. In this demonstration, we introduce some applications with the above-mentioned features, and show how users can build these applications on our framework.

1 Introduction

In recent years, many kinds of information has been provided via Web services, and developers have chances to combine different types of information in order to provide a new integrated service using mashup technologies.

We are proposing a new mashup framework [1,2] for creating flexible applications. The developer can prepare various mashup items in an application, where the users can browse items selectively according to their interests through GUI components called *widgets*. Receiving interactive actions for the browsing, our system incrementally creates mashed-up data with focusing on the current view of the application. The application developer only has to specify how to combine web services and how to display mashed-up data through widgets. This demonstration introduces some applications with the above features, and shows how to build these applications in our framework.

2 Application Features

Fig. 1 is a screenshot of a sample application for a hotel search. The user first inputs an address and a date of check-in. The "hotel table" lists nearby hotels and their information, and the "map" plots their locations. The user can click a row of the hotel table and get detailed information of the corresponding hotel on lower placed tables. The application handles user interactions, and invokes Web services in the background if necessary. When the system gets the results of Web services, it displays them instantly. These table widgets have facilities to add/change columns interactively (Fig. 1 A), and the use can use various criteria derived from mashups, such as photos, room rates, or information of

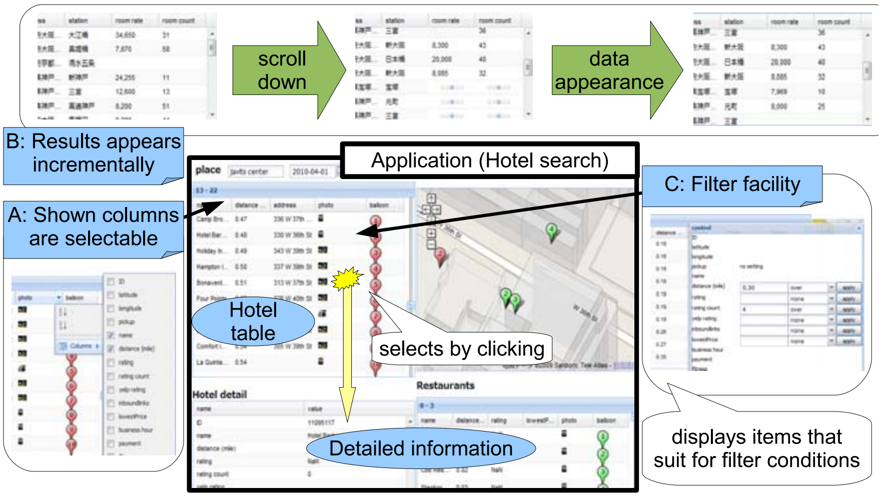

scroll down
data appearance
B: Results appears incrementally
Application (Hotel search)
C: Filter facility
A: Shown columns are selectable
Hotel table
selects by clicking
Detailed information
displays items that suit for filter conditions
Hotel detail
Restaurants

Fig. 1. Screenshot of a sample mashup application d its features

neighboring restaurants. Receiving scroll down actions, the table shows further results that will appear incrementally (B). When the user finds hotels having unsuitable properties, the user can apply filters to remove such hotels for the subsequent browsing (C). If the user has optional days to be checked, the system will update only affected results (e.g. room rates) on the current view.

As our system adopts demand-driven creation of mashed-up data, it first calculates only mashed-up data needed for the initial view. When the system gets user actions such as scroll down of the tables, it starts calculation of newly needed data for display. Web services that may often produce many results adopt paginated queries in order to return results in multiple pages, and our system fetches these pages in a demand-driven manner. If the user sets filter conditions on hotels, the system avoids needless mashups for hotels that are filtered out except condition values. In cases where the user changes some inputs, the system re-calculates only data depending on the changing values and needed for the current view.

3 How to Build Applications

Our framework provides a data management engine for demand-driven data creation and its built-in widget library, and the developer just specifies configurations of the data model and the display for an application (Fig. 2). The data model adopts declarative configurations to cope with our demand-driven data creation, and the developer can design large combinations of Web services. The widgets provide facilities of property monitoring and event handling to make applications interactive. This section gives a brief introduction of how to build applications in our framework. The detailed descriptions are described in [1]. The developer can use the GUI editor for the configurations.

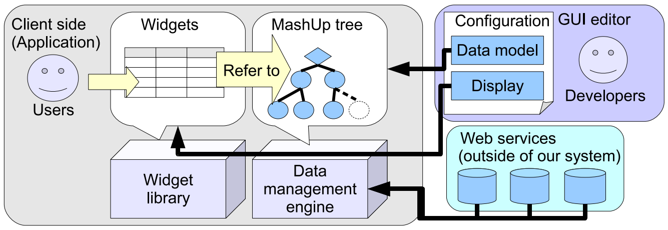


Fig. 2. Overview of our framework

3.1 Data Model Configuration

Our framework uses a tree data structure called a *mashup tree* to represent mashed-up data (Fig. 3 A). Each node of the mashup tree represents data operation and its result, while arrows represent data flows to deliver arguments to the operators. Using the tree structure, we can simply express nested list structures such as a list of `hotel` nodes and the list of neighboring `restaurant` nodes for each hotel. For child node creation at the elements of a list, the same operation is used for all the elements. Our framework provides the following four types of operations: user inputs, Web service calls, extractions of XML elements, and user-defined operations.

In the model canvas of our GUI editor (Fig. 3 B), top level nodes are placed in the "top box", and each declaration of a list structure is represented by a table. Each table column represents child node declarations of list elements. There is a catalog of major Web services and operation templates (e.g. aggregators like `sum` operation). The developer can add child node declarations by drag and drop actions from the catalog. To create the sample application, the developer first drops a hotel search Web service into the "top box". As the hotel search service returns the results as a list, our editor adds a hotel table linked with the "top box". Secondly, the developer drops a photo search service into the hotel table, and the photo column is created. When the developer drops a Web service or an operation that returns a list (e. g. restaurant search) into the hotel table, the corresponding table appears.

3.2 Display Configuration

There are two main way of specifying the display target. One is to specify static part of the mashup tree using the path from the top node. In the case of the hotel table, the developer specifies the path to `hotel` nodes. The other is to specify the target that is bounded to the property of other widgets. For example, each table widget has a "selected" property that denotes the element node corresponding to the currently selected row by the previous click action. In this application, lower placed tables monitor the "selected" property of the hotel table to display detailed information of the selected hotel.

In the display configuration mode (Fig. 3 C), our GUI editor allows the developer to choose widgets (e.g., table or map) from the widget catalog, layout

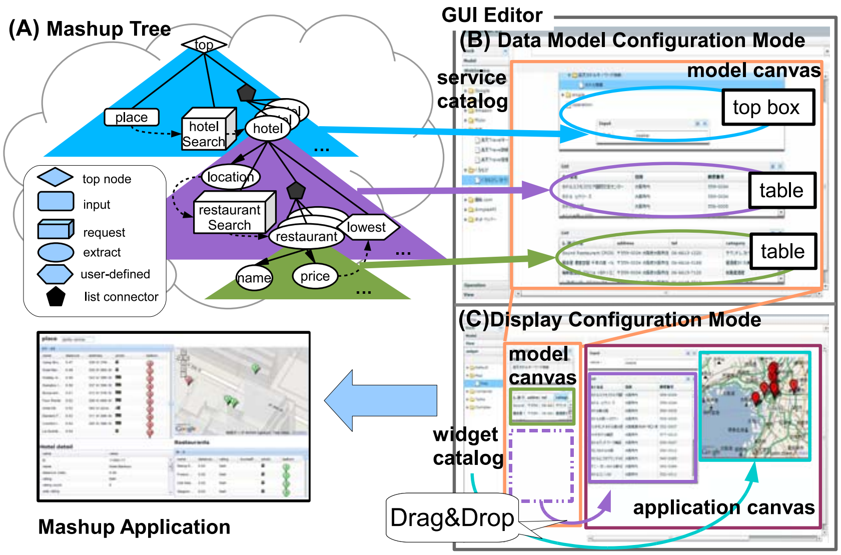


Fig. 3. Building applications in our framework

them on the application canvas, and set display targets afterwards. In the case of table widgets, there is more simple way. The developer can drag tables from the model canvas and drop them directly into the application canvas. In the sample application, the developer just drags hotel and restaurant tables from the model canvas into the application canvas, then two table widgets are created with default display target. Afterwards, the developer chooses the map from the catalog, and bounds the display target to a property of the hotel table.

Acknowledgements. This research was partially supported by Exploratory IT Human Resources Project (MITOU Program) of Information-technology Promotion Agency, Japan (IPA) in the fiscal year 2008.

References

1. Ikeda, S., Nagamine, T., Kamada, T.: Application Framework with Demand-Driven Mashup for Selective Browsing. Journal of Universal Computer Science 15(10), 2109–2137 (2009)
2. Connecollect Project, http://www.cs26.scitec.kobe-u.ac.jp/farm/PL/?Connecollect

Retrieving System of Presentation Contents Based on User's Operations and Semantic Contexts

Daisuke Kitayama and Kazutoshi Sumiya

University of Hyogo
1–1–12 Shinzaike-honcho, Himeji, Hyogo 670-0092, Japan
`ne07p001@stshse.u-hyogo.ac.jp, sumiya@shse.u-hyogo.ac.jp`

Abstract. More and more presentation contents, which consist of heterogeneous media such as videos and slides, have recently been recorded and viewed. In e-Learning, we often use presentation contents archives. However, if there is a slide that includes a keyword that the users do not know, it is difficult for them to understand the rest of the presentation contents. So, they must stop viewing it and look up the keyword. In this paper, we propose an interval-retrieving method that is based on the user's viewing operations. By using this method, we extract the user's interval-retrieving intention using his/her viewing operations and selected keywords. Queries are also generated for the intervals by using the user's retrieving intention and a keyword role in the intervals. This method also enables the users to obtain intervals that help them to more efficiently understand the contents of the presentation without discontinuing viewing them.

1 Introduction

Presentations can be stored for repeated viewing [1], which means you never have to worry about missing a lecture. In the case of e-learning, a speaker or listener who has viewed the content once might want to review it. The content then can be retrieved from the archives where the presentation data is stored. Such content is often located at specific intervals of a presentation[2,3]. As many presentations are now accessible via multi-media[4], a practical method is necessary for extracting the relevant scenes from integrated heterogeneous media content.

It has proven difficult, however, to extract a meaningful interval from a single medium of a multi-media presentation. For example, trying to extract an interval from text media using a keyword input is likely to yield countless intervals containing the key word and is thus basically useless. In the case of video media, excessively long or inadequately short intervals will be extracted.

We propose a presentation contents retrieval system that is based on the users' viewing operations and the semantic contexts from presentation archives. Our system uses the following three processes. At first, we extract the users' operations for the interested scenes and the keywords from the viewing interactions.

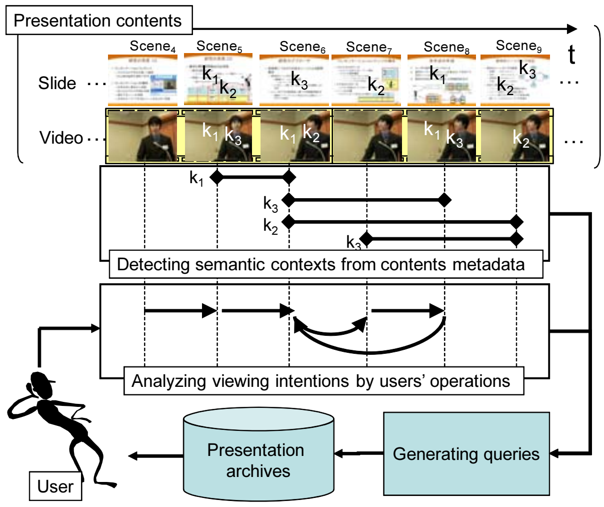

Presentation contents
Scene4 Scene5 Scene6 Scene7 Scene8 Scene9
t
Slide
Video
k1 k2 k3
Detecting semantic contexts from contents metadata
Analyzing viewing intentions by users' operations
User
Presentation archives
Generating queries

Fig. 1. Concept of interval retrieving using user's operation

Next, we analyze the semantic context of the scenes they are interested in and the keywords from the presentation contents based on the users' operations. Finally, we generate a search query for the presentation archives using the analyzed semantic contexts. Users can then retrieve intervals from other presentation contents that they are interested in without inputting query keywords by using our system.

2 Our Approach

We analyze the semantic context of presentation contents from the users' operations and generate a query for use in searching through presentation archives. Figure 1 shows a conceptual image of our system. Our system has the following three feature.

Analyzing semantic contexts. We hypothesized that there is a specific role for each scene in a presentation and these specific roles might be determined by analyzing the relation among the scenes across all media. Suppose that a speaker is introducing a new game machine. When he or she is explaining the "basic performance of the game machine", the scene is more likely to be regarded as a general-content scene than an explanation of the "mechanism of the game machine". However, this same scene is likely to be regarded as having more detailed content than an explanation of the "list of game machines". Consequently, the role of a scene is assumed to be relatively determined. If the speech content is detailed, the slide content is presumed to be detailed, and there will be even more speech in the video [5]. We assume that an interval of interest for extraction can be selected by analyzing the semantic roles of the scenes and their relation across heterogeneous media.

We call these roles semantic contexts. We define semantic contexts as detailed, generalized, instance, and additional intervals. We use the semantic contexts to extract the users' intentions and generating queries.

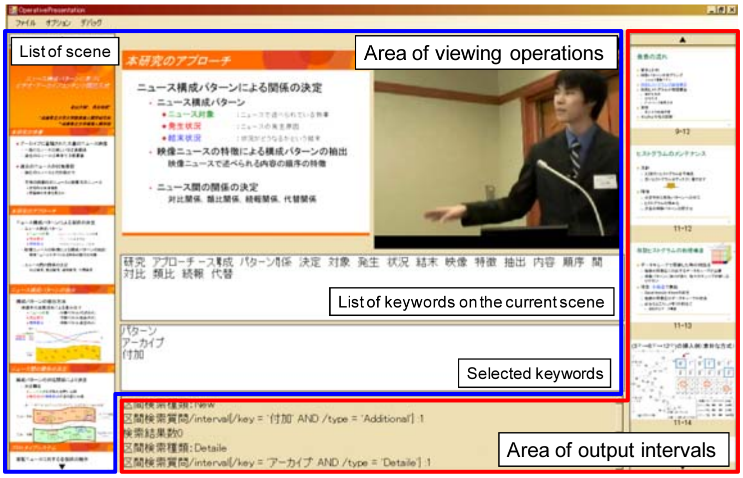
Extracting users' searching intentions. We define viewing operations as viewing video contents, selecting scenes, and browsing contents using slides. We took into account that the users' searching intentions are extracted from these operations. For example, when a user views scenes and then selects keywords that do not include the viewed scenes, he or she wants the meanings of the keywords because he or she does not know how to use the keywords for any contexts. The users' searching intentions are limitless. Therefore, we define five examples of users' searching intentions, a new word search, a detailed interval search, an abstract search, an instance search, and a comparing search. We use the searching intentions for extracting keywords and generating queries.

Generating queries using semantic contexts and users' intentions. We generate a query from viewing operation pattern, selected keywords and order relation of scenes. We use an order relation to represent the semantic contexts. For example, when a user selects the keyword "controller" after viewing scenes about "game machines", a searching intention is used as a detailed interval search. In this case, We generate a query for extracting intervals that have a semantic relation to the detailed interval, and a scene concerning a "controller" should be followed by another scene concerning "game machines'f. If the order is a reverse sequence, the extracted interval has a different meaning from that of the user viewing the presentation.

3 Concluding Remarks

Our prototype system was implemented by analyzing the users' viewing operations and the retrieving interval from presentation archives. The users' viewing operations are analyzed for extracting the users' intentions from the viewing operations, detecting the semantic contexts, and generating queries. The output intervals are retrieved by the generated queries. In particular, we retrieve the candidate intervals by using keywords, which are included in the generated queries. Then, the system filters by the order condition and detected semantic contexts. The prototype system was developed using Visual C$\sharp$ with Microsoft Visual Studio 2005. The terms in the slides and videos were extracted using the *ChaSen* Japanese morphological analysis system [6] called SlothLib [7]. Figure 2 shows some screen shots of the interface for the prototype system.

In this paper, we introduced our prototype system. We discussed how to retrieve archived presentation contents by using semantic contexts and users' operations. In our future work, we will evaluate our system using a large amount of real presentation contents and participants. We will confirm the accuracy of detecting semantic contexts and users' intentions, and the effectiveness of the generated queries by comparing them with a conventional keyword search.


List of scene
Area of viewing operations
本研究のアプローチ
ニュース構成パターンによる関係の決定
List of keywords on the current scene
Selected keywords
Area of output intervals

研究 アプローチ ース構成 パターン関係 決定 対象 発生 状況 結末 映像 特徴 抽出 内容 順序 間 対比 類比 続報 代替

パターン
アーカイブ
付加

```
区間検索種類:New
区間検索質問/interval[/key = '付加' AND /type = 'Additional'] :1
検索結果数0
区間検索種類:Detaile
区間検索質問/interval[/key = 'アーカイブ' AND /type = 'Detaile'] :1
```

Fig. 2. Screen image of prototype system

Acknowledgment

This research was supported in part by a Grant-in-Aid for Scientific Research (B)(2) 20300039 and Grant-in-Aid for JSPS Fellows 21.197 from the Ministry of Education, Culture, Sports, Science, and Technology of Japan.

References

1. DBSJ Archives, http://www.dbsj.org/Japanese/Archivess/archivesIndex.html
2. Kan, M.Y.: SlideSeer: A digital library of aligned document and presentation pairs. In: Proc. of the 7th ACM/IEEE-CS joint conference on Digital libraries, pp. 81–90 (2007)
3. Repp, S., Linckels, S., Meinel, C.: Towards to an Automatic Semantic Annotation for Multimedia Learning Objects. In: Proc. of the international workshop on Educational multimedia and multimedia education, pp. 19–26 (2007)
4. Ricoh Corporation: MPmeister, http://www.ricoh.co.jp/mpmeister/
5. Kitayama, D., Otani, A., Sumiya, K.: A Scene Extracting Method based on Structural and Semantic Analysis of Presentation Content Archives. In: Proc. of The Seventh International Conference on Creating, Connecting and Collaborating through Computing, C5 2009 (2009)
6. Asahara, M., Matsumoto, Y.: Extended Models and Tools for High-performance Part-of-Speech Tagger. In: Proc. of The 18th International Conference on Computational Linguistics (COLING 2000), pp. 21–27 (2000)
7. SlothLib, http://www.dl.kuis.kyoto-u.ac.jp/SlothLibWiki/

Fuzzy Keyword Search on Spatial Data

Sattam Alsubaiee and Chen Li

Department of Computer Science, University of California, Irvine, CA 92697, USA
{salsubai,chenli}@ics.uci.edu

Abstract. In recent years, many websites have started providing keyword-search services on maps. In these systems, users may experience difficulties finding the entities they are looking for if they do not know their exact spelling, such as the name of a restaurant. In this paper, we present a solution to support fuzzy keyword search on spatial data. We combine a spatial index structure with inverted indexes on grams to efficiently answer fuzzy queries on maps. We show two system prototypes to demonstrate the practicality of our solution.

1 Motivation

Many websites based on geographical information nowadays support keyword search on their data such as business listings and photos. Such services accept queries consisting of two parts: a set of keywords and a spatial location. The goal is to find objects with these keywords close to the location. Such a query is called a *spatial-keyword* (SK) query [1]. There are several local-search websites, such as Google Maps, Yahoo! Local, Bing Maps, Yellow Pages, and MapQuest Maps. At such a website, a user might look for a restaurant called "Aomatsu" close to Irvine in California. The website returns business listings close to the city that match the keywords. Another example website is the service by Flickr that supports location-based photo search (http://www.flickr.com/map). A user may ask for photos about the "Coliseum Stadium" close to Los Angeles.

Users often do not know the exact spelling of keywords. For example, the user may mistype a query as (`aumatso restaurant`) *near* (`Irvine`, `CA`) when looking for the restaurant `Aomatsu`. Similarly, a user could mistype the word "`coliseum`" and submit a query: (`colisum stadium`) *near* (`Los Angeles`, `CA`). It is important to find relevant answers to such mistyped queries. Unfortunately, most existing location-based systems do not provide correct answers to a query even with only a single typo. Table 1 shows how several systems behaved for five mistyped variations of the query "`aomatsu restaurant`" as of June 20, 2009. In most cases, the search engines either returned an empty answer or gave irrelevant results. Both Google and Yahoo could suggest alternative queries, but they very often could not give the right suggestion. We also experimented with the Flickr Maps photo search engine and saw similar limitations. An interesting observation is that during the development of our work, the results of these systems kept changing. For instance, the evaluation results as of September 2009 had more "no-results" cases.

Table 1. Results of Local-Search Engines for Mistyped Queries (as of June 20, 2009)

Search Engine	Results of Mistyped Queries				
	`aumatso restaurant`	`aomatso restaurant`	`aumatsp restaurant`	`amatsu restaurant`	`aumatso`
Yahoo! Local	●	✓	●	●	●
Bing Maps	●	●	●	●	●
Yellow Pages	●	●	●	✗	●
MapQuest Maps	✗	✗	✗	✗	●

● : No results ✓ : Correct suggestion ✗ : Wrong suggestion/answer.

In this paper, we study how to solve this problem by supporting fuzzy keyword search on spatial data. Given a query with keywords and a location, we want to find objects close to the location with those keywords, even if those keywords do not match exactly. Thus we can find relevant objects for the user even in the presence of typos in the query or data. Notice our approach is more powerful than the approach of suggesting an alternative query (the "Did you mean" feature used by many systems). The latter can only suggest a new query, while our approach can find relevant answers, even if the answers' keywords are only similar to those of the query.

2 Problem Formulation and Our Solution

Formulation: Consider a collection of spatial objects $o_1, \ldots, o_n$, and each object has a textual description (a set of keywords) T_i and a location L_i. A query consists of the following: $Q = \langle Q_s, Q_t \rangle$, where Q_s is a spatial region such as a rectangle or a circle. Q_t is a fuzzy-keyword condition, which consists of a set of keywords and an edit-distance threshold δ. Our goal is to find the objects in the collection such that each of them r is within the region Q_s. In addition, for each keyword k in Q_t, the object r has a keyword d in its description, such that the edit distance between k and r is within the threshold δ. For simplicity, we assume the threshold is a constant, and our results can be easily generalized to the case where the threshold varies based on the length of the keywords. A related problem is fuzzy string search: given a collection of strings, how to efficiently find those that are similar to a given query string? Many algorithms have been proposed to answer fuzzy keyword queries using inverted lists of grams [2]. Several algorithms have been proposed in the literature to answer spatial-keyword queries by assuming exact matching of keywords [1,3]. A recent paper [4] also studies how to support fuzzy keyword search on spatial data. Their approach is probabilistic, and does not guarantee to find *all* the answers to a query.

Our solution: We use an R*-tree to index the objects based on their spatial attribute. Our solution extends naturally to other tree-based structures, such as kd-trees and quadtrees. Each node in the tree stores the keywords of the spatial objects in its leaf nodes. To support fuzzy keyword search, we choose nodes in the tree to build gram-based inverted indexes for their stored keywords. In this paper, we choose one level of the tree and construct gram indexes for all the nodes at that level, denoted by L.

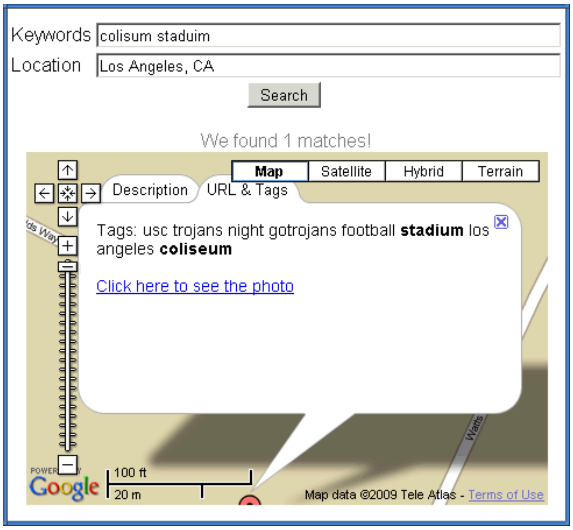
We answer a fuzzy spatial-keyword query as follows. Let Q be a query with a spatial condition Q_s and a fuzzy-keyword condition Q_t. Intuitively, the algorithm traverses the tree top-down. Before reaching level L, where the gram-based inverted indexes reside, the algorithm only relies on the spatial information of each node to decide which nodes to traverse. The rationale is that higher levels can have many keywords, and it is computationally expensive to do pruning based on the condition Q_t by finding similar keywords. At level L, for each candidate node, the algorithm uses the node's gram inverted index to find keywords that satisfy the fuzzy-keyword condition Q_t, i.e., finding keywords that are similar to at least one keyword in Q_t according to the edit-distance threshold. This set of similar keywords, denoted by C, is propagated in the later process of the traversal in order to prune branches in the tree.

We studied how to choose the level L of tree nodes to construct gram indexes. Notice that at each tree node, its stored keywords is the *union* of the keywords of its leaf-node objects. If multiple objects have the same keyword, this keyword is stored only once in the common ancestors of their leaf nodes. In particular, the root of the tree ($L = 1$) has all the keywords in the dataset. We can see a trade-off between the query performance and the size of the gram inverted indexes. As L increases, the total number of keywords on which we need to build gram inverted indexes increases. Thus the total size of the gram inverted indexes will increase. Meanwhile, the performance of finding similar keywords from a gram inverted index is very related to the size of the index.

3 Demonstration Description

We used two real datasets to develop two prototypes for demonstration. The first dataset was a multimedia metadata collection extracted from Flickr pages, called "CoPhIR Test Collection" (http://cophir.isti.cnr.it). We processed the dataset to extract the photos taken in the U.S. based on their latitude and longitude values. Moreover, we used the keywords in the title, description, and tags of a photo as its textual attribute. The final dataset had about two million objects, with a size of 300MB. Each record had a URL corresponding to the photo or a page including the photo. The second dataset had geographical objects (such as lakes and hills) obtained from http://www.geonames.org/. We used the objects residing in the U.S., and the final dataset had about 1.8 million objects. The total data size was 90MB. In the experiments, we built inverted indexes using 2-grams.

Both systems provide an interface similar to existing local-search and photo-search services on maps. Each interface has a map and two input boxes, one for textual keywords and one for a location. The map is using the Google Maps API, and can display the search results for a query. We also use the Google Maps Geocoder API to obtain the latitude and longitude of the entered location. Once the user clicks the search button, the objects satisfying the query conditions will be shown as red markers on the map. If the user clicks a marker, the information about the corresponding object (e.g., a photo) will be displayed. Some markers overlap with each other, and we grouped these near-by markers under a green marker. If the user clicks a green marker, the map will zoom in to show the included objects. We will use the prototypes to demonstrate the capabilities and advantages of supporting fuzzy keyword queries. Fig. 1 shows a screenshot of the results page for a query with a mistyped keyword on the first dataset.


Keywords colisum staduim
Location Los Angeles, CA

We found 1 matches!

Tags: usc trojans night gotrojans football **stadium** los angeles **coliseum**

Click here to see the photo

Map data ©2009 Tele Atlas - Terms of Use

Fig. 1. A screenshot of our system on the CoPhIR dataset for answering the mistyped query "`colisum stadium` *near* `Los Angeles, CA`"

Acknowledgements. We thank Kensuke Ohta for his discussions in this work.

References

1. Hariharan, R., Hore, B., Li, C., Mehrotra, S.: Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In: SSDBM, p. 16 (2007)
2. Li, C., Lu, J., Lu, Y.: Efficient merging and filtering algorithms for approximate string searches. In: ICDE, pp. 257–266 (2008)
3. Felipe, I.D., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: ICDE, pp. 656–665 (2008)
4. Yao, B., Li, F., Hadjieleftheriou, M., Hou, K.: Approximate string search in spatial databases. In: ICDE (2010)

Adaptive Recommendation for Preferred Information and Browsing Action Based on Web-Browsing Behavior

Kosuke Takano[1] and Kin Fun Li[2]

[1] Kanagawa Institute of Technology, Department of Information & Computer Sciences
1030 Shimo-ogino Atsugi, Kanagawa 243-0292, Japan
takano@ic.kanagawa-it.ac.jp

[2] University of Victoria, Department of Electrical & Computer Engineering
Victoria, BC, Canada V8W 3P6
kinli@uvic.ca

Abstract. A Web recommender system based on the inference from a user's Web-browsing behavior has been proposed and implemented. This system is capable of recommending items of interest to a user and specific Web-browsing action on the current item using a novel similarity measure approach. The recommender is adaptive to individual user's preference as well as a user's changing interest via a dynamic user feedback mechanism and empirical statistics on Web-browsing actions taken. Furthermore, users' quantitative comments and the qualitative measures of users' behavior provide an ideal setting to ascertain the premise, implicitly used in several other existing recommender systems, that there is a correlation between preference information and browsing behavior.

Keywords: recommender system, adaptive feedback, web-browsing behavior, preference thesaurus, browsing action.

1 Motivations and Objectives

The World Wide Web has become an indispensable resource for people to gather information as the Web provides a quick search capability to its rich data and abundant services. To retrieve desired and relevant information effectively, many techniques have been proposed. In particular, recommender systems have been designed to assist a user in navigating the myriad of information on the Web and suggesting items that the user is most likely interested in. Explicit user preference information based on user feedbacks and implicit measures such as browsing history are being used in interest prediction and information filtering.

Web-browsing behaviors such as dwell time, mouse click, scroll action, and search query, together with site visit history and personal document collection, are often used in usage and content mining to assist in making recommendation. In the "Stuff I've Seen" system [2], personal contextual items, such as authors and thumbnails from the documents that the user has already seen, are used to search for relevant information. The SEARCHY system [3] filters and re-ranks the Web search results by exploiting the user's profile as obtained from his/her Web-browsing behavior.

Morita et al. propose an information reminder system [4] where a user's action such as printing, copying and pasting, are recorded during a Web-browsing session. This user profile is then utilized to provide personalized information to the user. A personalized information provision system [1], [2], [3], [4], [5] recommends or navigates to preferable information based on the implicit assumption that a user's preference is strongly correlated to his/her browsing behavior. However, to the best of our knowledge, this assumption has neither been studied nor validated.

In our previous work [5], we proposed an adaptive personalized recommender system using a preference-thesaurus constructed based on Web-browsing behavior and user feedback. This system is personalized for an individual user by capturing his/her browsing behavior into a preference-thesaurus. Moreover, the system can adapt to different users as well as their changing behavior and/or interest through direct feedback and continuous update to each individual's preference-thesaurus.

We have extended and made several enhancements to our initial prototype recommender. Our main goal here is to demonstrate the capabilities of this system:

- Improve recommendation through continuous use
- Personalize to individual user
- Recommend Web-browsing action for the current document/page
- Provide mechanism to monitor dynamically changing user interest

In addition to the above contributions, this work is also a major endeavour aimed to validate the conjecture that there is a correlation between Web-browsing behaviour and information preference. This correlation study will be carried out based on user interviews and empirical data.

2 An Adaptive Personalized Recommender System

The current version of the recommender system provides two types of recommendation to the user: Web items or pages that are most likely of interest to the user, and a recommended Web-browsing action such as bookmarking or printing the current page. Feedback from the user in the form of ranked results and the monitored Web-browsing actions empower the dynamic adaptive nature of the system.

As shown in Fig. 1, the recommendation consists of three iterative phases. During the first phase, a user's Web-browsing behaviors are monitored and important term sets are extracted for the associated behaviors from the viewed/collected documents. An initial personal preference-thesaurus is constructed based on each behavior's term set and its term score. In the second phase, Web items to be recommended are ranked by the similarity between the preference-thesaurus term set and each document. During the final learning phase, the preference-thesaurus is updated based on the user's evaluation feedback on the most recent recommended items. In this phase, our recommender detects influential Web-browsing behavior in order to make better recommendation by filtering out non-influential behaviors.

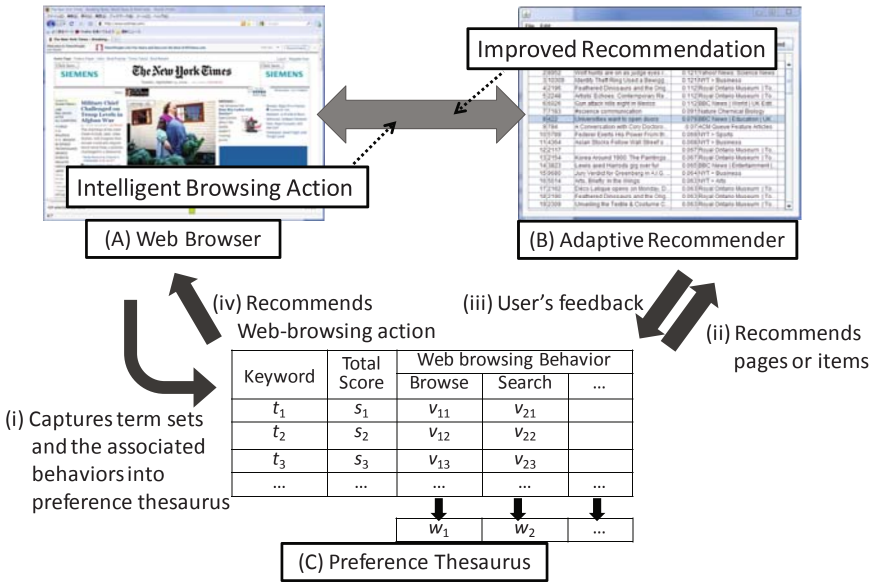
Based on a mapping of the preference-thesaurus and the current page or item, the Web-browsing action recommended is shown to the user as an enlarged icon (e.g., printer on tool bar) or in a pop-up window to suggest bookmarking as shown in Fig. 2. If the user took the Web-browsing action recommended, the preference-thesaurus is updated by increasing the behavior's association with the term set included in the current page or item.

User survey in written or electronic form and/or interview will provide some qualitative evidence on the premise that Web-browsing behavior is correlated to information preference. Also, whether the user will follow the suggested Web-browsing action will give quantitative measure to ascertain this correlation which is the central tenet of the proposed recommender system.

3 Demo Proposal

The demo system will be a prototype recommender system as described in previous sections. All the capabilities of the system will be available to the users. User survey and/or interview, as well as statistical measurement on Web-browsing actions, will be carried out. A preliminary on-line demo can be found at:

http://www.chen.ic.kanagawa-it.ac.jp/dasfaa2010/demo.html


Improved Recommendation
Intelligent Browsing Action
(A) Web Browser
(B) Adaptive Recommender
(iv) Recommends Web-browsing action
(iii) User's feedback
(ii) Recommends pages or items
(i) Captures term sets and the associated behaviors into preference thesaurus
(C) Preference Thesaurus

Keyword	Total Score	Web browsing Behavior		
		Browse	Search	…
t_1	s_1	v_{11}	v_{21}	
t_2	s_2	v_{12}	v_{22}	
t_3	s_3	v_{13}	v_{23}	
…	…	…	…	…
		w_1	w_2	…

Fig. 1. An Adaptive Web-browsing Recommender System

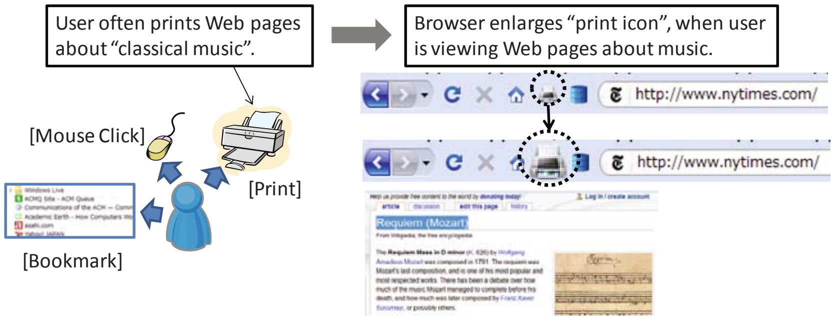

User often prints Web pages about "classical music".
Browser enlarges "print icon", when user is viewing Web pages about music.
[Mouse Click]
[Print]
[Bookmark]
http://www.nytimes.com/
Requiem (Mozart)

Fig. 2. Web-browsing Action: Enlarged Icon for Printing

Table 1. Typical Web-browsing behaviors

ID	Web-browsing behavior	Term set to be extracted
I_1	Web pages browsed	Terms appeared on the Web pages
I_2	Terms on Web pages selected by mouse-click	Terms selected
I_3	Terms on Web pages copied onto the clipboard	Terms copied onto the clipboard
I_4	Keywords searched within Web pages	Search keywords
I_5	Web pages saved	Terms appeared on the saved Web pages
I_6	Web pages printed	Terms appeared on the Web pages printed
I_7	Web pages bookmarked	Terms appeared on the Web pages bookmarked
I_8	Search keywords input to the Web search engines	Search keywords input to the Web search engines
I_9	Web pages browsed from search results	Terms appeared on the returned Web pages browsed

References

1. Chirita, P.-A., Firan, C.S., Nejdl, W.: Personalized Query Expansion for the Web. In: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 7–14 (2007)
2. Dumais, S., Cutrell, E., Cadiz, J.J., Jancke, G., Sarin, R., Robbins, D.C.: Stuff I've Seen: A System for Personal Information Retrieval and Re-Use. In: Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 72–79 (2003)
3. Marcialis, I., Vita, E.D.: SEARCHY: An Agent to Personalize Search Results. In: Proceedings of the 3rd International Conference on Internet and Web Applications and Services (ICIW 2008), pp. 512–517 (2008)
4. Morita, T., Hidaka, T., Tanaka, A., Kato, Y.: System for Reminding a User of Information Obtained Through a Web Browsing Experience. In: Proceedings of the 16th International World Wide Web Conference, WWW 2007, pp. 1327–1328 (2007)
5. Takano, K., Li, K.F.: An Adaptive Personalized Recommender Based on Web-Browsing Behaviour Learning. In: Proceedings of the 2009 IEEE International Symposium on Mining and Web (MAW 2009), pp. 654–660 (2009)

BIDEL: An XML-Based System for Effective Fast Change Detection of Genomic and Proteomic Data

Song Yang[2] and Sourav S. Bhowmick[1,2]

[1] Singapore-MIT Alliance, Nanyang Technological University, Singapore
[2] School of Computer Engineering, Nanyang Technological University, Singapore
assourav@ntu.edu.sg

Abstract. A key issue to address in biological data integration is how to detect changes to the underlying biological data sources. In this demonstration, we present a novel system called BIDEL for detecting changes to genomic and proteomic data (sequences and annotations). We transform heterogeneous biological data to XML format (if necessary) and then detect changes between two versions of unordered XML representation of biological data. This demonstration will showcase the functionality of our system and the effectiveness of change detection in life sciences environment.

1 Introduction

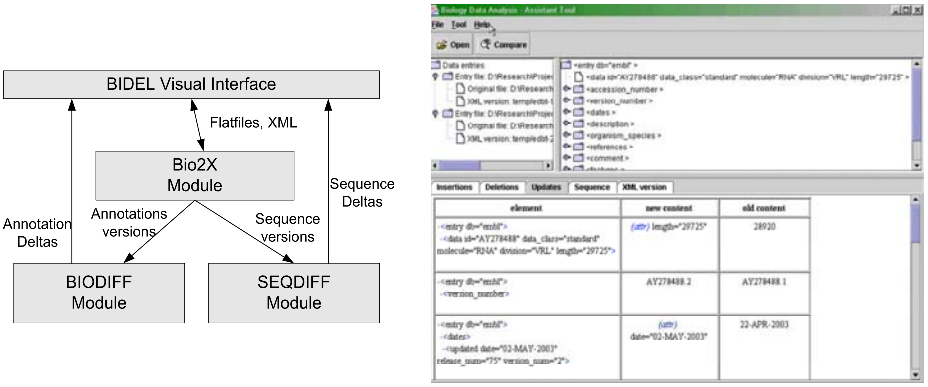
Detecting changes to the underlying biological data sources is a key challenge in biological data integration. In this demonstration, we present a system called BIDEL[1] (***Bi**ological **Del**ta Detector*) for detecting changes to old and new versions of genomic and proteomic data. In our system, we first transform heterogeneous genomic and proteomic data to XML format [4] (if necessary) and then detect changes between two versions of unordered XML representation of biological data [2,3]. Specifically, the BIODIFF [2] component of BIDEL detects *exact* changes to the *annotation* (non-sequence) data associated with gene or protein sequences. It *extends* X-Diff [6], a published unordered XML change detection algorithm, by addressing its limitations to exploit structural characteristics of underlying data (discussed in Section 3 and [2]). On the other hand, the SEQDIFF module detects changes to *sequence* data. Note that existing XML change detection techniques [6] are not designed to compute changes to sequences. To the best of our knowledge, *this is the first system to detect changes to both annotation and sequence data associated with biological entities.*

2 System Overview

Figure 1(a) shows the architecture of BIDEL and consists of the following modules.

The Visual Interface Module: Figure 1(b) depicts the screen dump of the visual interface of BIDEL. It consists of three panels. The top-left panel displays a list of versions of biological data (genomic and proteomic data in XML as well as flat file format) that we wish to compare in BIDEL. A user can view the details of a document in the top right panel by clicking on the correspond item in the list. Note that the transformation

[1] In Maltese, bidel means "to change".



element	new content	old content
-<entry db="embl"> -<data id="AY278488" data_class="standard" molecule="RNA" division="VRL" length="29725">	(attr) length="29725"	28920
-<entry db="embl"> -<version_number>	AY278488.2	AY278488.1
-<entry db="embl"> -<dates> -<updated date="02-MAY-2003" release_num="75" version_num="2">	(attr) date="02-MAY-2003"	22-APR-2003

(a) BIDEL Architecture **(b) GUI of BIDEL**

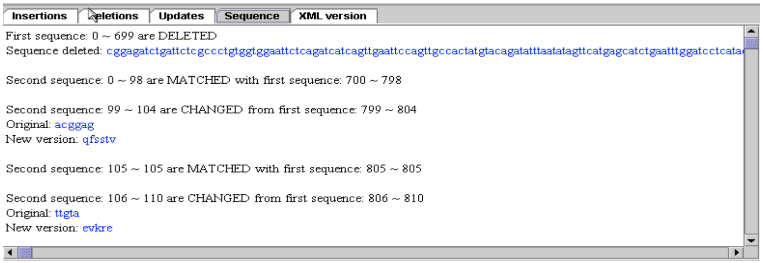
Fig. 1. Architecture and visual interface of BIDEL

of a flat file to XML format is achieved by clicking on the `Open` icon in the menu. It invokes the Bio2X [4] algorithm for the transformation (discussed below). Given an old and new versions of XML representation of biological data, the changes to the data are computed by clicking on the `Compare` icon in the menu. It invokes the BIODIFF [2] and SEQDIFF [3] algorithms to detect the changes to annotation (non-sequence) and sequence data, respectively. The bottom panel displays various types of changes that are computed by these two algorithms. A user can click on one of the four tabs (`Insertions`, `Deletions`, `Updates`, and `Sequence`) to view a specific type of changes. For instance, in Figure 1(b) clicking on the `Updates` tab results in the display of a list of updates detected by BIODIFF. Similarly, Figure 2 depicts the changes to sequence data when the `Sequence` tab is clicked. A user can also view the complete set of changes in XML format by clicking on the `XML version` tab.

The Bio2X Module: This module currently converts flat file data from GenBank, EMBL, Swiss-Prot, and PDB into XML format. The rule bases are designed in a consistent manner so that a single transformer is sufficient to parse any data file from any database. The transformer chooses a suitable rule base for parsing the input flat file based on its origin database and generates the XML data file. The rule base exploits the *hierarchical* structure of the source to constrain the data extraction problem. It allows for extraction of target patterns based on surrounding landmarks, *line types* and other lexical patterns in the flat files. It also allows for more advanced features such as disjunctive pattern definitions. Finally, it involves machine-learning techniques to refine the rules in order to improve the accuracy of the transformation. The reader may refer to [4] for details related to the transformer.

The BIODIFF Module: This module implements the algorithm BIODIFF [2] that identifies *exact* changes to the *annotations* associated with primary biological objects (gene and protein sequences). It takes as input two versions of unordered XML representation of annotations of a gene or a protein (the sequence data is excluded) from the *Bio2X* module, denoted by D_1 and D_2, and detect changes between them. The algorithm extends X-Diff [6] by addressing some of its limitation and consists of four phases, namely the *identifier checking* phase, the *parsing and hashing* phase, the *matching* phase, and the *edit script generation* phase. The *identifier checking phase* determines whether the two versions are identical by comparing the *version identifiers* of the biological data records. If the two entries are not identical, then BIODIFF parses D_1 and D_2 into DOM trees $tree1$ and $tree2$ in the *parsing and hashing phase*. This step is similar to the one in X-Diff [6]. The goal of the *matching phase* is to compute the minimum cost matching between $tree1$ and $tree2$. Each XML tree is divided into a set of smaller subtrees rooted at distinct first-level nodes. Note that each first-level element nodes resulted from *Bio2X* has a unique name and hierarchy. Each smaller tree is compared with another smaller tree from the second XML tree having the node with same name. This step makes it possible to use *different* methods of matching for subtrees having *different* characteristics (e.g, elements containing distinct or identical subelements). Note that these characteristics of the subtrees are extracted by the *Bio2X* module during XML transformation. BIODIFF employs four types of matching techniques for different subtree characteristics, namely *one-to-one* comparison, *identical subelement* comparison, *extended signature* comparison, and *bipartite matching*. Lastly, similar to X-Diff, the *edit script generation phase* generates a minimum-cost edit script for changes to annotation data based on the minimum cost matching found in the matching phase.

The SEQDIFF Module: This module implements a heuristic non-optimal algorithm called SEQDIFF [3] to detect changes between two versions of biological sequences of the same biological entity extracted by the *Bio2X* module. Specifically, it detects insert, delete, and update of a nucleotide or protein sequence at a specific position. The algorithm consists of two phases, namely the *sequence comparison* phase and the *edit script generation* phase. The first phase is based on the local alignment concept used in BLAST. That is, genes adjacent in one sequence should also remain near to each other in the new sequence. Based on this heuristic, the sequence is divided into segments (that act like sliding windows) and an optimal alignment is performed within each segment without any consideration for other choices from the other segment. Note that the algorithm computes the alignment twice by first matching the first sequence onto second one; and then matching the second one onto the first one. This is because it is not known *apriori* in which direction of match generates the higher score. In the second phase, the edit script is generated based on the alignment with a higher score. The alignment



```
First sequence: 0 ~ 699 are DELETED
Sequence deleted: cggagatctgattctcgccctgtggtggaattctcagatcatcagttgaattccagttgccactatgtacagatatttaatatagttcatgagcatctgaatttggatcctcata

Second sequence: 0 ~ 98 are MATCHED with first sequence: 700 ~ 798

Second sequence: 99 ~ 104 are CHANGED from first sequence: 799 ~ 804
Original: acggag
New version: qfsstv

Second sequence: 105 ~ 105 are MATCHED with first sequence: 805 ~ 805

Second sequence: 106 ~ 110 are CHANGED from first sequence: 806 ~ 810
Original: ttgta
New version: evkre
```

Fig. 2. Output of SEQDIFF module

result is traversed and the aligned segments with same values are *matched*. The aligned segments with different values are *updated*. The unaligned segments in the new version are *inserted*, while the ones in the old version are *deleted*. Figure 2 shows a screenshot of the edit script generated by the SEQDIFF module.

3 Related Systems and Novelty

A number of techniques for detecting changes to ordered and unordered XML data has been proposed (e.g., [6]). BIDEL differs from these approaches in the following ways. Firstly, since the min-cost max-flow algorithm for computing the bipartite mapping between two XML trees is the most time consuming part, it is desirable to reduce the size of data set during mapping. Existing XML change detection techniques fail to do so for biological data as it ignores the structural semantics of the underlying data. In contrast, BIDEL reduces the data size for bipartite mapping by exploiting the structural characteristics of XML representation of biological data. Consequently, BIODIFF shows better performance than X-Diff (up to 6 times faster) [2]. Secondly, none of these techniques are designed to detect changes to sequence data. The SEQDIFF component of BIDEL implements a heuristic strategy to detect different types of changes to old and new versions of sequence data. Lastly, to the best of our knowledge, none of the existing XML change detection system has been demonstrated in a major database conference.

Pairwise sequence alignment techniques [1,5] can be considered as the closest to the SEQDIFF module. Our change detection tool differs from these techniques in the following ways. First, sequence alignment techniques focus on finding *similarities* between the sequences whereas our technique focus on finding *differences* between a pair of sequences. Second, these approaches are designed to compare sequences among *different* biological entities. However, in change detection problem we are interested in detecting changes to two versions of a sequence of the *same* entry in terms of insertion, deletion, and update. Third, is the issue of performance. SEQDIFF trades off optimality for better performance. Specifically, SEQDIFF is significantly faster than DCLBDA [1], an optimal sequence alignment algorithm (highest observed factor being 350 times [3]).

4 Demonstration

Our demonstration aims to showcase the functionality and effectiveness of the BIDEL system in detecting changes to genomic and proteomic data. We will showcase the followings. (a) Demonstrate detection of different types of changes to annotation and sequence data using real-world datasets (EMBL, PDB, and Genebank). We will show how this process is simplified by the BIDEL visual interface. (b) Demonstrate the cases when the result quality of BIDEL is comparable to X-Diff as far as detection of changes to annotation data is concerned. (c) Demonstrate better efficiency and scalability of BIODIFF module compared to general unordered XML change detection algorithms (say X-Diff). We will also show cases where X-Diff fails to detect changes to annotation data due to lack of memory but BIODIFF is able to detect these changes.

References

1. Davidson, A.: A Fast Pruning Algorithm for Optimal Sequence Alignment. In: IEEE BIBE (2001)
2. Song, Y., Bhowmick, S.S.: BioDiff: An Effective Fast Change Detection Algorithm for Genomic and Proteomic Data. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 275–287. Springer, Heidelberg (2007)
3. Song, Y., Bhowmick, S.S.: SeqDiff: An Effective Fast Change Detection Algorithm for Biological Sequences. Technical Report (2008), www.cais.ntu.edu.sg/~assourav/TechReports/SeqDiff-TR.pdf
4. Song, Y., Bhowmick, S.S.: Bio2X: A Rule-based Approach for Semi-automatic Transformation of Semistructured Biological Data to XML. Data and Knowledge Engineering Journal 52(2), 249–271 (2003)
5. Tatusova, T.A., Madden, T.L.: BLAST 2 Sequence, a new tool for comparing protein and nucleotide sequences. FEMS Microbiology Letters 174, 247–250 (1999)
6. Wang, Y., DeWitt, D., Cai, J.-Y.: X-Diff: A Fast Change Detection Algorithm for XML Documents. In: ICDE, pp. 519–530 (2003)

DISTRO: A System for Detecting Global Outliers from Distributed Data Streams with Privacy Protection*

Ji Zhang[1], Stijn Dekeyser[1], Hua Wang[1], and Yanfeng Shu[2]

[1] Department of Mathematics and Computing,
The University of Southern Queensland, Australia
{Ji.Zhang,Stijn.Dekeyser,Hua.Wang}@usq.edu.au
[2] CSIRO ICT Centre, Hobart, Australia
Yanfeng.Shu@csiro.au

Abstract. In this demo proposal, we present a new system, called DISTRO (a.k.a **DI**stributed **STR**eam **O**utlier Detector), for detecting outliers from distributed data streams. DISTRO is able to effectively identify outliers from distributed data streams that are consistent with those generated by the centralized detection paradigm. DISTRO is also able to ensure high-level data privacy throughout the detection process. A number of optimization strategies are devised to further enhance its speed and communication performance. This proposal provides details on the motivation and technical challenges of detecting outliers from distributed data streams, presents an overview of DISTRO, and gives the plans for its system demonstration.

1 Introduction

Outlier detection from data streams is an important research problem in data mining that aims to find objects that are considerably dissimilar with the majority of data in the streams. In many cases, data streams are collected by multiple distributed agents (e.g., sensors in a sensor network) and outliers need to be detected from *all* the data that are collected. This kind of outliers are called *global outliers*.

The challenge for detecting global outliers lies in that centralized outlier detection (i.e., integrating the distributed data streams and carrying out outlier detection centrally) is intractable due to the potentially huge amount of data in the data streams and/or the privacy-related issues of data collected in different sites. The existing distributed outlier detection methods mainly suffer the following drawbacks: 1) The outlier-ness metrics they use are not updatable and are therefore not suitable for data streams [1][2][3]; and 2) The mediator they use needs to process a significant portion of the detailed stream data, thus the privacy of data may be compromised [4][5].

To address these problems, we present a new system, called DISTRO (short for **DI**stributed **STR**eam **O**utlier Detector) in this demo proposal. The innovative features and contributions of DISTRO are summarized as follows:

– DISTRO is able to produce outlier detection results that are consistent with those produced in a centralized environment without data integeration;

* This work is partly supported by the Tasmanian ICT Centre, which is jointly funded by the Australian Government through the Intelligent Island Program and CSIRO.

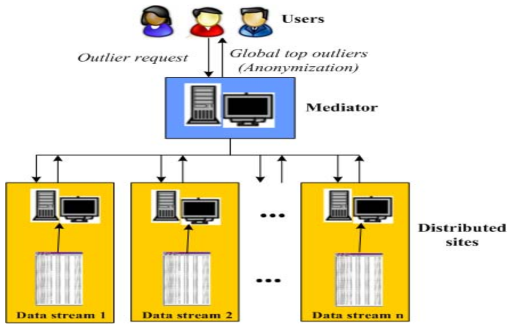


Fig. 1. System architecture of DISTRO

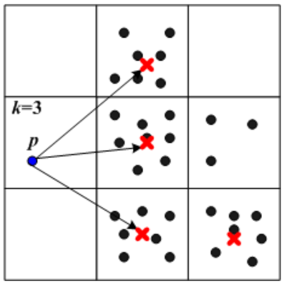


Fig. 2. Calculate k-ODF for data p ($k = 3$)

- The communication between the mediator and the distributed sites only involves compact summary-level information, leading to a low data transfer overhead;
- A number of optimization techniques have been incorporated into DISTRO to enable it to achieve even better speed and space performance;
- As end users and mediator have no access to the detailed distributed data except the final top outliers and data communication is prohibited amongst different distributed sites, DISTRO can, to the maximum extent, ensure privacy of proprietary data in the whole detection process. Various anonymization schemes can also be readily incorporated to anonymize the final outliers if necessary.

2 An Overview of DISTRO

In this section we present an overview of DISTRO and describe the algorithms that needs to be executed in various parts of the system. The system architecture diagram of DISTRO is given in Figure 1. There are three major parties in DISTRO system: the end users, the mediator and the distributed sites. The end users are the people who request outliers (typically the top n) for certain purpose. Requests from users are then passed to the mediator. Upon receiving a request, the mediator starts to execute the outlier detection process. The major role of the mediator is to generate the global data summary periodically and broadcast it to all the distributed sites for detecting local outliers. Each distributed site collects and processes the data stream it receives. The final global top n outliers are generated by the mediator and returned to end users through it. In DISTRO, the mediator can communicate with each distributed site but no communication is allowed amongst distributed sites themselves. Both the mediator and distributed sites have necessary computational capacity.

In order to compute the outlier-ness of data (as defined below) in the data streams efficiently, we introduce the grid structure to partition the data space in each distributed site. A multi-dimensional grid with equal-volumed cells is super-imposed in the data space. The gird structure facilitates the efficient generation of updatable local/global data summaries that are suitable for data stream applications.

Definition 1. **k-Outlying Degree Factor (k-ODF)**: k-ODF measures the strength of outlierness of data in the stream. ODF of a data point p is defined as the averaged distance between p and its k nearest dense cell centroids: $k\text{-}ODF(p) = \frac{\sum_{i=1}^{k} Dist(p, centroid(c_i))}{k}$, where k is a user-specified parameter and typically takes a small value. k-ODF measures outlier-ness for data by calculating its distance to its nearby dense regions, which is intuitive and consistent with human perception. Figure 2 presents an example of calculating k-ODF for data p when $k = 3$ (the red crosses in the figure indicate centroids of the dense cells).

The Algorithm of DISTRO. DISTRO detects global outliers from distributed data streams in the following steps:

Step 1: Assigning data into grid structure (distributed sites). Data in the data stream are read in sequentially and assigned into a cell in the grid. Instead of physically creating the grid structure, we only maintain the list of populated cells. The incoming data is mapped into one appropriate cell in this list. If the data falls into a cell that is not yet in the list, then a new cell will be added into the list. The density of a cell will be updated using a decaying function and sliding window techniques.

Step 2: Generating representative data (mediator). The density of cells will be aggregated periodically in the mediator. This operation is called aggregation. When aggregation is triggered, the local cell density needs to be transfered to the mediator. We only transfer the information of the populated cells rather than that of all the cells in the grid. Upon receiving the information of populated cells from all the distributed sites, the aggregation is carried out to aggregate density of populated cells from distributed sites and identify dense cells in the grid in order to calculate k-ODF for data. The dense cells are defined as the smallest number of most dense cells in the grid that contain no less than $q * 100\%$ of the total number of data that are assigned in the grid ($0 < q \leq 1$ and typically q is quite close to 1). The exact value of the number of the dense cells N_r is determined using the following inequity: $\sum_{i=1}^{N_r - 1} Dec(c_i) \leq q * 100\% N \leq \sum_{i=1}^{N_r} Dec(c_i)$. The centroids of the dense cells are called representative data which are considered as a global summary of the data streams. The list of representative data are then broadcast to all the distributed sites.

Step 3: Generating local top-n outliers (distributed sites). When each distributed site receives the representative data from mediator, the k-ODFs of data are calculated. The top-n local outliers can be picked up based on k-ODF for each distributed site. The local top-n outliers are then sent to mediator for producing the global top outliers.

Step 4: Generating global top-n outliers (mediator). When all the top local outliers are collected, the mediator will merge them and generate the global top-n outliers. The final results are retuned to end users.

3 Optimization Techniques for DISTRO

Besides the above algorithm, we have also devised a number of optimization techniques for effectively speeding up DISTRO and reducing the transferring overhead. For ease of referral, we call the algorithm we presented earlier in Section 2 as the *base algorithm*.

• **Using Outlier Candidates.** Outlier candidates are those data selected from the data stream in each distributed site which feature high outlying values. The local top-n outliers are selected only from these outlier candidates based on the k-ODF ranking. The number of outlier candidates is normally a few times (e.g. 4 or 5) of n in order to provide a sufficiently large pool for outlier selection. Suppose that the size of outlier candidates is $m * n$, where m is a positive real-value number provided by users. To generate $m * n$ outlier candidates, the data in the s most sparse cells whose total number of data exceeds $m * n$ will be selected as the outlier candidates. Here s is picked as the minimum integer satisfying this requirement. This strategy makes it possible for DISTRO to only evaluate k-ODF for a small number of data in the data stream.

• **Performing Local Pruning of Streaming Data.** In the base algorithm, all the streaming data arriving at each distributed site needs to be first archived in the limited-sized temporary storage space (memory or disc), awaiting aggregation to be performed. Nevertheless, when streaming data are arriving in a high rate, this limited storage space will be quickly filled up and aggregation has to be triggered for all the distributed sites in a relatively high frequency. As we have assumed that there is no communication amongst distributed sites, it is impossible to ship the unevenly distributed data amongst distributed sites to solve this problem. Alternatively, we devise a cell density estimation technique to significantly reduce the aggregation frequency without seriously compromising the effectiveness of outlier detection. Suppose that each distributed site has received a global summary during time t, then the estimated global density of cell c at time $t'(t' > t)$ at site i can be quantified as $Den_{estimate}(c, i, t') = Den(c, t) + f(c, i, t' - t)$, where $f(c, i, \Delta t)$ is the function that returns the estimated changes of density of c in a time duration of Δt in distributed site i based on statistics obtained in the previous aggregation cycles. By doing this, we can have an estimated global summary without aggregation, based on which the new locally estimated representative data can be generated. Using the locally estimated representative data, we can estimate the k-ODF of each local streaming data. We only need to store those data with relatively large k-ODF as local outlier candidates and discard the rest. The benefit of this strategy is that we can remarkably reduce the size of stream data that needs to be archived in the temporary storing space and noticeably prolonged the interval between two consecutive aggregation cycles. This contributes to a low data transferring overhead of DISTRO.

• **Implementing Global Top-n Outliers Merging Algorithm.** In the base algorithm, each distributed site will ship the top-n local outliers to the mediator to produce the global top-n outliers. This scheme can be further fine tuned to achieve a better transfer performance. The basic idea is that we first only transfer, from each distributed site, $\frac{n}{N_{sites}}$ to the mediator to ensure that there will be initially n outliers in mediator. These n outliers are merged to produce the initial top-n global outlier list. The minimum value of k-ODF of this list, denoted as min_{kODF} is extracted and broadcast to all the distributed sites. We can then prune away local outliers in each distributed whose k-ODF value is lower than min_{kODF} as it is guaranteed that they cannot possibly be included in the global top-n list. This pruning strategy is able to reduce the local outlier candidates that need to be transferred to the mediator.

4 Demonstration Plan

Our demonstration of DISTRO will consist of the following four parts:

First, we will describe to the audience the real-life application scenarios of outlier /anomaly detection in distributed multi databases/repository to motivate our DISTRO system. We will also introduce the pros and cons of the existing systems.

Second, we will showcase the system architecture of DISTRO. The Emphasis will be the introduction of various components in DISTRO regarding what their roles are and how they communicate with each other in the outlier detection process. The architecture demonstration is very useful to help the audience understand the algorithm of DISTRO that involves end users, mediator and distributed sites;

Third, the experimental evaluation results will be presented to the audience to show the effectiveness and efficiency of DISTRO. We will show that the the result of DISTRO is consistent with that generated in a centralized environment. Component analysis will also be used to show how each optimization technique assists DISTRO to achieve better speed and space performance;

Last but not least, an on-site demonstration of DISTRO will be played to the audience. The audience will be encouraged to interact with the demo themselves. We will provide on-site assistance to the audience to use the prototype upon request.

References

1. Branch, J.W., Szymanski, B.K., Giannella, C., Wolff, R., Kargupta, H.: In-Network Outlier Detection in Wireless Sensor Networks. In: ICDCS 2006, p. 51 (2006)
2. Chhabra, P., Scott, C., Kolaczyk, E.D., Crovella, M.: Distributed Spatial Anomaly Detection. In: INFOCOM 2008, pp. 1705–1713 (2008)
3. Dutta, H., Giannella, C., Borne, K.D., Kargupta, H.: Distributed Top-K Outlier Detection from Astronomy Catalogs using the DEMAC System. In: SDM 2007 (2007)
4. Sheng, B., Li, Q., Mao, W., Jin, W.: Outlier detection in sensor networks. In: MobiHoc 2007, pp. 219–228 (2007)
5. Su, L., Han, W., Yang, S., Zou, P., Jia, Y.: Continuous Adaptive Outlier Detection on Distributed Data Streams. In: HPCC 2007, pp. 74–85 (2007)

Introduction to Social Computing

Irwin King

Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, NT, Hong Kong
king@cse.cuhk.edu.hk
http://www.cse.cuhk.edu.hk/~king

Abstract. With the advent of Web 2.0, Social Computing has emerged as one of the hot research topics recently. Social Computing involves the collecting, extracting, accessing, processing, computing, visualizing, etc. of social signals and information. More specifically, this tutorial places special emphases in machine learning, data mining, information retrieval, and other computational techniques involved in collective intelligence processing of social behavior data collected from blogs, wikis, clickthrough data, query logs, tags, etc., and from areas such as social networks, social search, social media, social bookmarks, social news, social knowledge sharing, and social games. In this tutorial, I plan to give an introduction to Social Computing and elaborate on how the various characteristics and aspects are involved in the social platforms for collective intelligence. The topics include social network theory and modeling, graph mining, query log processing, learning to rank, recommender systems, human computation, etc. The tutorial is prepared for machine learning, web mining, and information retrieval researchers who are interested in computational approaches to social computing.

Brief Profile

Irwin King's research interests include machine learning, web intelligence & social computing, and multimedia processing. In these research areas, he has over 200 technical publications in journals (JMLR, ACM TOIS, IEEE TNN, Neurocomputing, NN, IEEE BME, PR, IEEE SMC, JAMC, JASIST, IJPRAI, DSS, etc.) and conferences (NIPS, IJCAI, CIKM, SIGIR, KDD, PAKDD, ICDM, WWW, WI/IAT, WCCI, IJCNN, ICONIP, ICDAR, etc.). In addition, he has contributed over 20 book chapters and edited volumes. Moreover, Irwin King has over 30 research and applied grants. One notable system he has developed is the VeriGuide System, previously known as the CUPIDE (Chinese University Plagiarism IDentification Engine) system, which detects similar sentences and performs readability analysis of text-based documents in both English and in Chinese to promote academic integrity and honesty.

Irwin King is an Associate Editor of the IEEE Transactions on Neural Networks (TNN) and IEEE Computational Intelligence Magazine (CIM). He is a member of the Editorial Board of the Open Information Systems Journal, Journal of Nonlinear Analysis and Applied Mathematics, and Neural Information ProcessingLetters and Reviews Journal (NIP-LR). He has also served as Special Issue Guest Editor for Neurocomputing, International Journal of Intelligent Computing and Cybernetics (IJICC), Journal of Intelligent Information Systems (JIIS), and International Journal of Computational Intelligent Research (IJCIR). He is a senior member of IEEE and a member of ACM, International Neural Network Society (INNS), and Asian Pacific Neural Network Assembly (APNNA). Currently, he is serving the Neural Network Technical Committee (NNTC) and the Data Mining Technical Committee under the IEEE Computational Intelligence Society (formerly the IEEE Neural Network Society). He is also a Vice-President and Governing Board Member of the Asian Pacific Neural Network Assembly (APNNA).

Selected References

1. Xin, X., King, I., Lyu, M.R., Deng, H.: A social recommendation framework based on multi-scale continuous conditional random fields. In: Proceedings to the ACM 18th Conference on Information and Knowledge Management (CIKM2009), Hong Kong, China, November 2-9, pp. 1247–1256. ACM, New York (2009)
2. Ma, H., King, I., Lyu, M.R., Yang, H.: Semi-nonnegative matrix factorization with global statistical consistency in collaborative filtering. In: Proceedings to the ACM 18th Conference on Information and Knowledge Management (CIKM 2009), Hong Kong, China, November 2-9, pp. 767–775. ACM, New York (2009)
3. Lin, Z., King, I., Lyu, M.R.: Matchsim: A novel neighbor-based similarity measure with maximum neighborhood matching. In: Proceedings to the ACM 18th Conference on Information and Knowledge Management (CIKM2009), Hong Kong, China, November 2-9, pp. 1613–1616. ACM, New York (2009)
4. Deng, H., King, I., Lyu, M.R.: Enhancing expertise retrieval using community-aware strategies. In: Proceedings to the ACM 18th Conference on Information and Knowledge Management (CIKM 2009), Hong Kong, China, November 2-9, pp. 1733–1736. ACM, New York (2009)
5. Zhou, T.C., King, I.: Automobile, car and BMW: Horizontal and hierarchical approach in social tagging systems. In: Workshop Proceedings to the Social Web Search and Mining (SWSM 2009) at the ACM 18th Conference on Information and Knowledge Management (CIKM2009), Hong Kong, China, November 2-9, pp. 25–32. ACM, New York (2009)
6. Ma, H., Lyu, M., King, I.: Learning to recommend with trust and distrust relationships. In: Proceedings to the 3rd ACM Conference on Recommender Systems, New York City, NY, USA, October 22-25 (accepted)
7. Yuen, M.-C., Chen, L.-J., King, I.: A survey of human computation systems. In: Proceedings of the 2009 International Symposium on Social Computing Applications (SCA 2009), 12th IEEE International Conference on Computational Science and Engineering (CSE2009), Vancouver, Canada, August 29-31, pp. 723–728. IEEE Computer Society, Los Alamitos (2009)
8. Ma, H., King, I., Lyu, M.R.: Learning to recommend with social trust ensemble. In: Allan, J., Aslam, J.A., Sanderson, M., Zhai, C., Zobel, J. (eds.) Proceedings to the 32nd Annucal ACM SIGIR Conference (SIGIR 2009), Boston, MA, July 19-23, pp. 203–210. ACM Press, New York (2009)

9. Deng, H., King, I., Lyu, M.R.: Entropy-biased models for query representation on the click graph. In: Allan, J., Aslam, J.A., Sanderson, M., Zhai, C., Zobel, J. (eds.) Proceedings to the 32nd Annucal ACM SIGIR Conference (SIGIR2009), Boston, MA, July 19-23, pp. 339–346. ACM Press, New York (2009)
10. Deng, H., Lyu, M.R., King, I.: A generalized co-hits algorithm and its application to bipartite graphs. In: The Proceedings to the 15th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2009), Paris, France, June 28-July 1 (2009)
11. Deng, H., King, I., Lyu, M.R.: Effective latent space graph-based re-ranking model with global consistency. In: Proceedings to the Second ACM International Conference on Web Search and Data Mining (WSDM 2009), Barcelona, Spain, February 9-12 (2009)
12. Huang, K., Xu, Z., King, I., Lyu, M.R.: Semi-supervised learning from general unlabeled data. In: Proceedings to the Eighth IEEE International Conference on Data Mining (ICDM 2008), Pisa, Italy, December 15-19 (2008)
13. Huang, K., King, I., Lyu, M.R.: Direct zero-norm optimization for feature selection. In: Proceedings to the Eighth IEEE International Conference on Data Mining (ICDM 2008), Pisa, Italy, December 15-19 (2008)

Mining Moving Object, Trajectory and Traffic Data

Jiawei Han, Zhenhui Li, and Lu An Tang

Department of Computer Science
University of Illinois at Urbana-Champaign, USA
{hanj,zli28,tang18}@uiuc.edu

With the wide availability of satellite, RFID, GPS, sensor, wireless, and video technologies, moving-object data have been collected in massive scale and are becoming increasingly rich, complex, and ubiquitous. There is an imminent need for scalable and flexible data analysis over moving-object information; and thus mining moving-object data has become one of major challenges in data mining. There have been considerable research efforts on data mining for moving object, trajectory, and traffic data sets. However, there has been few systematic tutorial on knowledge discovery from such moving-object data sets. This tutorial presents a comprehensive, organized, and state-of-the-art survey on methodologies and algorithms on analyzing different kinds of moving-object data sets, with an emphasis on several important mining tasks: *pattern-mining, clustering, classification, outlier analysis*, and *multidimensional analysis*. Besides a thorough survey of the recent research work on this topic, we also show how real-world applications can benefit from data mining of moving object, trajectory, and traffic data sets. The tutorial consists of three parts: (1) moving object pattern mining, (2) trajectory data mining, and (3) traffic data mining.

In the first part, *moving object pattern mining*, we introduce different pattern mining algorithms for various moving object patterns. *Frequent pattern* is one of the most basic patterns that detects frequently visited routes, such as "Railway Station → Time Square → Central Park" for New York city travelers. The challenge of frequent pattern lies in the approximation of locations and transition time. *Periodic pattern mining* is another interesting topic since periodicity is an intrinsic nature of moving objects. For example, people have weekly working pattern and animals have yearly migration pattern. But how to detect periodicity for moving objects in 2-dimensional space remains a difficult problem. *Moving object grouping pattern* discovers the social behaviors of moving objects in groups. Research have been conducted in different definitions, such as *moving cluster*, *flock* and *convoy*. But strict temporal and spatial constraint may result in failure of finding meaningful patterns. Thus, a concept called *swarm* is further proposed to suit for more realistic cases. Other interesting patterns including *leadership*, *following*, and *meeting* are studied as well. Leadership and following patterns discovers a small number of objects (*e.g.*, suspect, wolf) that follow one or a set of given moving objects (*e.g.*, people, sheep). Meeting could describe the movement that suspects meet to plot an attack or animals meet together for the same food resources.

In the second part, *trajectory mining*, we focus more on trajectory clustering, classification and outlier detection. For *trajectory clustering*, many high dimensional data clustering methods can be easily adapted if we treat each timestamp as one dimension. Studies related to different distance measures in high dimensional space and efficient computation of these distance functions haven been conducted. Probabilistic methods are also proposed by modeling a set of trajectories as individual sequences of points generated from a finite mixture model. While these methods cluster trajectories as a whole, they ignore sub-trajectory clusters. A partition-and-group framework is proposed to solve this problem. It first partitions a trajectory into multiple line segments and then cluster the line segments based on density. Outlier could be a natural byproduct of clustering result. The objects that are distant from any cluster can be considered as outliers. Recently, methods specifically designed for more complex cases are developed, such as integration of multi-dimensional information and partial trajectory outlier detection. While there are many clustering and outlier detection methods, few *classification method* has been developed for moving objects. A related area could be time series classification, in which 1-Nearest-Neighbor has been the most popular method and a shapelet-based classification method has recently shown to be effective. Time series studies mainly deal with 1-dimensional data whereas moving object classification has to face more complicated 2-dimensional spatial data. Recently, a trajectory classification method based on regions and trajectory clusters has shown satisfactory result. This method extracts discriminative regions and trajectory clusters for some class as classification features.

Finally, we introduce some state-of-the-art traffic data mining methods. Traffic data, different from free space movement, is confined to road networks. An important task in traffic analysis is the *shortest/fastest path computation*. Classical shortest path problem focuses on efficient computation in a large road network. However, historical traffic data may discover real fastest path and thus an adaptive fastest path computation method is proposed. Another interesting topic is to *predict destination* of moving vehicles. Methods based on Bayesian classification and frequent pattern have been developed to efficiently and effectively predict the recent or distant movement. In real life, people are also concerned with *road conditions*, such as hot/jammed roads and abnormal events on some road segments. To monitor road conditions, density-based routes clustering method is developed for hot routes discovery and temporal outlier detection in vehicle data is used to find abnormal road segments. Lastly, since road network naturally forms hierarchical structures and different granularity is embedded in temporal data, it is necessary to analyze moving object data in a *multidimensional way*, such as multidimensional traffic anomaly detection on highways and traffic cube and mining in traffic cube space.

In summary, this tutorial presents the state-of-the-art research on moving object data analysis including pattern mining, trajectory clustering, classification, outlier detection, and traffic analysis. It shows the confluence of multiple scientific and engineering disciplines, including data mining, database systems, geographic information system, statistical analysis, and machine learning, and links to multiple applications. We also discuss several promising research directions.

Querying Large Graph Databases

Yiping Ke[1], James Cheng[2], and Jeffrey Xu Yu[1]

[1] Department of Systems Engineering and Engineering Management
The Chinese University of Hong Kong
Shatin, New Territories, Hong Kong
{ypke,yu}@se.cuhk.edu.hk
[2] Division of Information Systems
School of Computer Engineering
Nanyang Technological University, Singapore
jamescheng@ntu.edu.sg

Abstract. Graph exists ubiquitously in a wide spectrum of application domains, such as protein structures in biology, chemical compounds in chemistry, food webs in ecology, social networks, Web graphs, P2P networks, and many more. With the increasing popularity of graph databases, how to assess graph data effectively and efficiently becomes an important research problem. Considerable research efforts have been devoted to developing advanced query processing techniques on graph databases. This tutorial presents a comprehensive survey on methodologies and techniques for querying large graph databases, including subgraph and supergraph query processing, structural similarity query processing, correlation search in transaction graph databases, connection query processing and approximate matching in large graphs. The tutorial is prepared for database and data mining researchers who are interested in complex data types that can be generally modeled as graphs.

1 Introduction

This tutorial provides a comprehensive overview of the main methodologies and techniques for querying large graph databases. The tutorial starts with *subgraph query processing*, which has been well-studied in the literature. Representative techniques include GraphGrep [1], gIndex [2], C-tree [3], FG-index [4], TreePi [5], GString [6], GDIndex [7], Tree+Δ [8], GCoding [9], and QuickSI [10]. It then continues with a newly introduced counterpart, *supergraph query processing*, with two existing techniques, cIndex [11] and GPTree [12]. The next part focuses on *structural similarity search*, which is a special type of subgraph queries by allowing structural relaxation. Systems that support structural similarity search include SUBDUE [13], RASCAL [14], Grafil [15], C-tree [3], and GDIndex [7]. The tutorial also covers an emerging type of graph queries, namely *correlation queries*, which discovers subgraphs with similar occurrence distributions. Typical techniques include CGSearch [16], TopCor [17], and FCP-Miner [18]. Finally, the tutorial ends with *query processing on large graphs*, including connection subgraph [19], center-piece subgraph [20], proximity subgraph [21], context-aware object connection discovery [22], and approximate matching queries [23,24].

References

1. Shasha, D., Wang, J.T.L., Giugno, R.: Algorithmics and applications of tree and graph searching. In: PODS, pp. 39–52 (2002)
2. Yan, X., Yu, P.S., Han, J.: Graph indexing based on discriminative frequent structure analysis. ACM Trans. Database Syst. 30(4), 960–993 (2005)
3. He, H., Singh, A.K.: Closure-tree: An index structure for graph queries. In: ICDE, p. 38 (2006)
4. Cheng, J., Ke, Y., Ng, W., Lu, A.: Fg-index: towards verification-free query processing on graph databases. In: SIGMOD, pp. 857–872 (2007)
5. Zhang, S., Hu, M., Yang, J.: Treepi: A novel graph indexing method. In: ICDE, pp. 966–975 (2007)
6. 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)
7. Williams, D.W., Huan, J., Wang, W.: Graph database indexing using structured graph decomposition. In: ICDE, pp. 976–985 (2007)
8. Zhao, P., Yu, J.X., Yu, P.S.: Graph Indexing: Tree + Delta >= Graph. In: VLDB, pp. 938–949 (2007)
9. Zou, L., Chen, L., Yu, J.X., Lu, Y.: A novel spectral coding in a large graph database. In: EDBT, pp. 181–192 (2008)
10. Shang, H., Zhang, Y., Lin, X., Yu, J.X.: Taming verification hardness: An efficient algorithm for testing subgraph isomorphism. In: VLDB, pp. 364–375 (2008)
11. 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)
12. 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)
13. Holder, L., Cook, D., Djoko, S.: Substucture Discovery in the SUBDUE System. In: KDD Workshop, pp. 169–180 (1994)
14. Raymond, J.W., Gardiner, E.J., Willett, P.: RASCAL: calculation of graph similarity using maximum common edge subgraphs. Comput. J. 45(6), 631–644 (2002)
15. Yan, X., Yu, P.S., Han, J.: Substructure similarity search in graph databases. In: SIGMOD Conference, pp. 766–777 (2005)
16. Ke, Y., Cheng, J., Ng, W.: Correlation search in graph databases. In: KDD, pp. 390–399 (2007)
17. Ke, Y., Cheng, J., Yu, J.X.: Top-k correlative graph mining. In: SDM, pp. 1038–1049 (2009)
18. Ke, Y., Cheng, J., Yu, J.X.: Efficient discovery of frequent correlated subgraph pairs. In: ICDM, pp. 239–248 (2009)
19. Faloutsos, C., McCurley, K.S., Tomkins, A.: Fast discovery of connection subgraphs. In: KDD, pp. 118–127 (2004)
20. Tong, H., Faloutsos, C.: Center-piece subgraphs: problem definition and fast solutions. In: KDD, pp. 404–413 (2006)
21. Koren, Y., North, S.C., Volinsky, C.: Measuring and extracting proximity in networks. In: KDD, pp. 245–255 (2006)
22. Cheng, J., Ke, Y., Ng, W., Yu, J.X.: Context-aware object connection discovery in large graphs. In: ICDE, pp. 856–867 (2009)
23. Tian, Y., Patel, J.M.: Tale: A tool for approximate large graph matching. In: ICDE, pp. 963–972 (2008)
24. Tong, H., Faloutsos, C., Gallagher, B., Eliassi-Rad, T.: Fast best-effort pattern matching in large attributed graphs. In: KDD, pp. 737–746 (2007)

Future Directions of Innovative Integration between Multimedia Information Services and Ubiquitous Computing Technologies

Yasushi Kiyoki[1] and Virach Sornlertlamvanich[2]

[1] Graduate School of Media and Governance, Keio University, SFC
5322 Endoh, Fujisawa, Kanagawa, 252-0816, Japan
kiyok@sfc.keio.ac.jp
[2] National Electronics and Computer Technology Center, Digitized Thailand Project,
National Science and Technology Development Agency, Thailand
virach@tcllab.org

Abstract

Over the past decade, the success of multimedia and database technologies results a large extent to use various data resources provided to create innovative applications. Rapid progress in this technology and its applications has been seen especially in the Internet-based social aspects. Databases, knowledge bases, data mining and multimedia data managements have become important subjects not only in academic communities related to information systems and computer science but also in business areas.

This panel session is the opportunity to address world-wide research issues for the exchange of scientific aspects and experiences achieved in multimedia data managements, ubiquitous computing, and other related disciplines. Basic system platforms will be discussed in the context of multimedia information services and ubiquitous computing technologies. The main topics of this panel session target the themes in the interdisciplinary domain between multimedia information modelling, multimedia systems and ubiquitous computing technologies.

The rapid progress of multimedia data management technology has realized the large scale of media data transfer and resource-accumulation in the world. The ubiquitous computing technology has also been creating new information provision environments in the world-wide scope. Innovative integrations of large scale multimedia data and ubiquitous computing resources will lead to a new information society. This panel session focuses on promising integration between multimedia information services and ubiquitous computing in regard to integrated system architectures, media-data analysis, community-based media data-creation, user-generated multimedia content, large-scale storage systems, mobile computing and new sensing device technologies. To this end much attention is paid also to various applications including e-community, e-learning and e-commerce. For construction of the integrated systems of multimedia information services and ubiquitous computing, further innovative technologies are expected. This panel session will discuss opportunities for explorations of significant research and development.

Author Index