

Dimitris Papadias  
Donghui Zhang  
George Kollios (Eds)

LNCS 4605

# Advances in and Tempo

10th International Symposium  
Boston, MA, USA, July 2007  
Proceedings

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

David Hutchison

*Lancaster University, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*University of Dortmund, Germany*

Madhu Sudan

*Massachusetts Institute of Technology, MA, USA*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Moshe Y. Vardi

*Rice University, Houston, TX, USA*

Gerhard Weikum

*Max-Planck Institute of Computer Science, Saarbruecken, Germany*

Dimitris Papadias Donghui Zhang  
George Kollios (Eds.)

# Advances in Spatial and Temporal Databases

10th International Symposium, SSTD 2007  
Boston, MA, USA, July 16-18, 2007  
Proceedings

Volume Editors

Dimitris Papadias  
Hong Kong University of Science and Technology  
Department of Computer Science and Engineering  
Clearwater Bay, Hong Kong, China  
E-mail: dimitris@cs.ust.hk

Donghui Zhang  
Northeastern University, College of Computer & Information Science  
360 Huntington Avenue, #202WVH, Boston, MA 02115, USA  
E-mail: donghui@ccs.neu.edu

George Kollios  
Boston University, Computer Science Department  
111 Cummington Street, Boston, MA 02215, USA  
E-mail: gkollios@cs.bu.edu

Library of Congress Control Number: 2007929929

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

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

ISSN 0302-9743  
ISBN-10 3-540-73539-9 Springer Berlin Heidelberg New York  
ISBN-13 978-3-540-73539-7 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media  
springer.com

© Springer-Verlag Berlin Heidelberg 2007  
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India  
Printed on acid-free paper SPIN: 12088263 06/3180 5 4 3 2 1 0

# Preface

SSTD 2007 was the tenth in a series of biannual events that discuss new and exciting research in spatio-temporal data management and related technologies. Previous symposia were successfully held in Santa Barbara (1989), Zurich (1991), Singapore (1993), Portland (1995), Berlin (1997), Hong Kong (1999), Los Angeles (2001), Santorini, Greece (2003) and Angra dos Reis, Brazil (2005). Before 2001, the series was devoted solely to spatial database management, and called SSD. From 2001, the scope was extended in order to accommodate also temporal database management, in part due to the increasing importance of research that considers spatial and temporal aspects jointly.

SSTD 2007 received 76 submissions from 19 countries (based on the affiliation of the first author). A thorough review process led to the acceptance of 26 high-quality papers, geographically distributed as follows: USA 10, Germany 3, Denmark 2, Hong Kong 2, Singapore 2, Brazil 1, Canada 1, France 1, Greece 1, Israel 1, South Korea 1, and Taiwan 1. The papers are classified in the following categories, each corresponding to a conference session: (1) Continuous Monitoring, (2) Indexing and Query Processing, (3) Mining, Aggregation and Interpolation, (4) Semantics and Modeling, (5) Privacy, (6) Uncertainty and Approximation, (7) Streaming Data, (8) Distributed Systems, and (9) Spatial Networks.

The success of SSTD 2007 was the result of team effort. First, we would like to thank the authors for providing the content of the program. We would also like to apologize to the authors of rejected papers, as some good submissions had to be left out. Second, we are grateful to the members of the Program Committee (and the external reviewers) for their thorough and timely reviews. We were impressed by the fact that 95% of the reviews were submitted on time, despite a reviewing process that lasted less than a month. Third, we are grateful to Ellen Grady and the students at Boston University and Northeastern University for their help with organizing and running the conference. Finally, we would like to thank Oracle Spatial, ESRI, and Microsoft Research for their generous support.

We believe that SSTD 2007 continued the successful tradition of the series, providing an interesting program and lively discussions in a pleasant environment.

May 2007

Dimitris Papadias  
Donghui Zhang  
George Kollios

# Organization

SSTD 2007 was organized by Boston University.

## Executive Committee

General Chair	George Kollios (Boston University)
Program Chairs	Dimitris Papadias (Hong Kong University of Science and Technology)
	Donghui Zhang (Northeastern University)
Organizing Chair	Ellen Grady (Boston University)
Web Master	Ling Hu (Northeastern University)

## Program Committee

Walid Aref	Purdue University (USA)
Lars Arge	University of Aarhus (Denmark)
Spiros Bakiras	John Jay College, CUNY (USA)
Elisa Bertino	Purdue University (USA)
Thomas Brinkhoff	Oldenburg University of Applied Sciences (Germany)
Reynold Cheng	Hong Kong Polytechnic University (Hong Kong)
Max Egenhofer	National Center for Geographic Information and Analysis, Maine (USA)
Amr El Abbadi	University of California, Santa Barbara (USA)
Dimitrios Gunopulos	University of California, Riverside (USA)
Ralf Hartmut Güting	Fernuniversität Hagen (Germany)
Marios Hadjieleftheriou	AT&T Labs Inc. (USA)
Erik Hoel	ESRI (USA)
Christian S. Jensen	Aalborg University (Denmark)
Panos Kalnis	National University of Singapore (Singapore)
George Kollios	Boston University (USA)
Ravi Kothuri	Oracle Spatial (USA)
Nick Koudas	University of Toronto (Canada)
Ki-Joune Li	Pousan National University (South Korea)
David Lomet	Microsoft Research (USA)
Nikos Mamoulis	Hong Kong University (Hong Kong)
Yannis Manolopoulos	Aristotle University (Greece)
Claudia Bauzer Medeiros	University of Campinas (Brazil)
Mohamed Mokbel	University of Minnesota (USA)
Kyriakos Mouratidis	Singapore Management University (Singapore)

## VIII Organization

Mario Nascimento	University of Alberta (USA)
Gultekin Ozsoyoglu	Case Western Reserve University (USA)
Spiros Papadimitriou	IBM T.J. Watson Research Center (USA)
Dieter Pfoser	Computer Technology Institute (Greece)
Siva Ravada	Oracle Spatial (USA)
John Roddick	Flinders University (Australia)
Simonas Šaltenis	Aalborg University (Denmark)
Markus Schneider	University of Florida (USA)
Bernhard Seeger	Philipps-Universität Marburg (Germany)
Timos Sellis	National Technical University of Athens (Greece)
Cyrus Shahabi	University of Southern California (USA)
Shashi Shekhar	University of Minnesota (USA)
Richard Snodgrass	University of Arizona (USA)
Jianwen Su	University of California, Santa Barbara (USA)
Kian-Lee Tan	National University of Singapore (Singapore)
Yufei Tao	Chinese University of Hong Kong (Hong Kong)
Yannis Theodoridis	University of Piraeus (Greece)
Vassilis J. Tsotras	University of California, Riverside (USA)
Agnès Voisard	Fraunhofer ISST and Freie Universität Berlin (Germany)
Kyu-Young Whang	KAIST (South Korea)
Jun Yang	Duke University (USA)
Ke Yi	AT&T Labs Inc. (USA)
Man Lung Yiu	Aalborg University (Denmark)

## External Reviewers

Ping Wu	Jeffrey Xie	Mehrdad Jahangiri
Stacy Patterson	Chuck Freiwald	Leyla Kazemi
Dan Lin	Shyam Antony	Hua Lu
Hai Yu	Elias Frentzos	Maria M. Ruxanda
Ahmed Metwally	Jinchuan Chen	Luiz Celso Gomes Jr
Hui Ding	Gabriel Ghinita	Gilberto Pastorello
Mohamed Shehab	Kiyoung Yang	Jaudete Daltio
Huagang Li	Hyunjin Yoon	

## Sponsors

Oracle Spatial (Platinum Sponsor)  
ESRI (Silver Sponsor)  
Microsoft Research (Silver Sponsor)

# Table of Contents

Continuous Monitoring of Exclusive Closest Pairs . . . . .	1
<i>Leong Hou U, Nikos Mamoulis, and Man Lung Yiu</i>	
Continuous Evaluation of Fastest Path Queries on Road Networks . . . . .	20
<i>Chia-Chen Lee, Yi-Hung Wu, and Arbee L.P. Chen</i>	
Continuous Medoid Queries over Moving Objects . . . . .	38
<i>Stavros Papadopoulos, Dimitris Sacharidis, and Kyriakos Mouratidis</i>	
Efficient Index Support for View-Dependent Queries on CFD Data . . . . .	57
<i>Christoph Brochhaus and Thomas Seidl</i>	
Generalizing the Optimality of Multi-step $k$ -Nearest Neighbor Query Processing . . . . .	75
<i>Hans-Peter Kriegel, Peer Kröger, Peter Kunath, and Matthias Renz</i>	
S-GRID: A Versatile Approach to Efficient Query Processing in Spatial Networks . . . . .	93
<i>Xuegang Huang, Christian S. Jensen, Hua Lu, and Simonas Šaltenis</i>	
Efficiently Mining Regional Outliers in Spatial Data . . . . .	112
<i>Richard Frank, Wen Jin, and Martin Ester</i>	
A Two Round Reporting Approach to Energy Efficient Interpolation of Sensor Fields . . . . .	130
<i>Brian Harrington and Yan Huang</i>	
Online Amnesic Summarization of Streaming Locations . . . . .	148
<i>Michalis Potamias, Kostas Patroumpas, and Timos Sellis</i>	
Spatial Partition Graphs: A Graph Theoretic Model of Maps . . . . .	167
<i>Mark McKenney and Markus Schneider</i>	
Geographic Ontology Matching with IG-MATCH . . . . .	185
<i>Guillermo Nudelman Hess, Cirano Iochpe, and Silvana Castano</i>	
Local Topological Relationships for Complex Regions . . . . .	203
<i>Mark McKenney, Alejandro Pauly, Reasey Praing, and Markus Schneider</i>	
MOBIHIDE: A Mobile Peer-to-Peer System for Anonymous Location-Based Queries . . . . .	221
<i>Gabriel Ghinita, Panos Kalnis, and Spiros Skiadopoulos</i>	



Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy . . . . .	239
<i>Ali Khoshgozaran and Cyrus Shahabi</i>	
Enabling Private Continuous Queries for Revealed User Locations . . . . .	258
<i>Chi-Yin Chow and Mohamed F. Mokbel</i>	
Computing a $k$ -Route over Uncertain Geographical Data . . . . .	276
<i>Eliyahu Safra, Yaron Kanza, Nir Dolev, Yehoshua Sagiv, and Yerach Doytsher</i>	
Querying Objects Modeled by Arbitrary Probability Distributions . . . . .	294
<i>Christian Böhm, Peter Kunath, Alexey Pryakhin, and Matthias Schubert</i>	
Invisible Graffiti on Your Buildings: Blind and Squaring-Proof Watermarking of Geographical Databases . . . . .	312
<i>Julien Lafaye, Jean Béguec, David Gross-Amblard, and Anne Ruas</i>	
Transformation of Continuous Aggregation Join Queries over Data Streams . . . . .	330
<i>Tri Minh Tran and Byung Suk Lee</i>	
Continuous Constraint Query Evaluation for Spatiotemporal Streams . . . . .	348
<i>Marios Hadjieleftheriou, Nikos Mamoulis, and Yufei Tao</i>	
Collaborative Spatial Data Sharing Among Mobile Lightweight Devices . . . . .	366
<i>Zhiyong Huang, Christian S. Jensen, Hua Lu, and Beng Chin Ooi</i>	
A Study for the Parameters of a Distributed Framework That Handles Spatial Areas . . . . .	385
<i>Verena Kantere and Timos Sellis</i>	
Distributed, Concurrent Range Monitoring of Spatial-Network Constrained Mobile Objects . . . . .	403
<i>Hua Lu, Zhiyong Huang, Christian S. Jensen, and Linhao Xu</i>	
Compression of Digital Road Networks . . . . .	423
<i>Jonghyun Suh, Sungwon Jung, Martin Pfeifle, Khoa T. Vo, Marcus Oswald, and Gerhard Reinelt</i>	
Traffic Density-Based Discovery of Hot Routes in Road Networks . . . . .	441
<i>Xiaolei Li, Jiawei Han, Jae-Gil Lee, and Hector Gonzalez</i>	
Spatio-temporal Network Databases and Routing Algorithms: A Summary of Results . . . . .	460
<i>Betsy George, Sangho Kim, and Shashi Shekhar</i>	
<b>Author Index . . . . .</b>	<b>479</b>

# Continuous Monitoring of Exclusive Closest Pairs<sup>\*</sup>

Leong Hou U<sup>1</sup>, Nikos Mamoulis<sup>1</sup>, and Man Lung Yiu<sup>2</sup>

<sup>1</sup> Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong

{hleongu, nikos}@cs.hku.hk

<sup>2</sup> Department of Computer Science, Aalborg University, DK-9220 Aalborg, Denmark

mly@cs.aau.dk

**Abstract.** Given two datasets  $A$  and  $B$ , their exclusive closest pairs (ECP) join is a one-to-one assignment of objects from the two datasets, such that (i) the closest pair  $(a, b)$  in  $A \times B$  is in the result and (ii) the remaining pairs are determined by removing objects  $a, b$  from  $A, B$  respectively, and recursively searching for the next closest pair. An application of exclusive closest pairs is the computation of (car, parking slot) assignments. In this paper, we propose algorithms for the computation and continuous monitoring of ECP joins in memory, given a stream of events that indicate dynamic assignment requests and releases of pairs. Experimental results on a system prototype demonstrate the efficiency of our solutions in practice.

## 1 Introduction

Due to the increasing popularity of location-based services, continuous monitoring of spatial queries emerges as an important research topic. Existing work [18, 11, 21, 13, 16, 17] focuses on *range* or *k nearest neighbor* ( $k$ NN) queries on moving objects. These problems can also be viewed as continuous joins between queries and data objects, according to their spatial relationship. However, there has not been much research done related to the continuous monitoring of spatial join results. Several variants of spatial join queries exist, such as the intersection join [2], the distance (or similarity) join [14], the all  $k$  nearest neighbors join [24], and the  $k$  (inclusive) closest pairs query ( $k$ ICP) [9, 4].

In this paper, we study an interesting type of spatial joins that has received little attention in the past. We call this operation the *k exclusive closest pairs* join ( $k$ ECP).  $k$ ECP produces  $k$  one-to-one assignments of objects between two datasets  $A$  and  $B$ , such that (i) the closest pair  $(a, b)$  in  $A \times B$  belongs to the result and (ii) the remaining pairs are determined by removing objects  $a, b$  from  $A, B$  respectively, and recursively searching for the next closest pair. Thus, each object appears only once in the result.

A real-life application of a  $k$ ECP query is the car-parking assignment problem. Consider a set  $A$  of car drivers that request for a parking slot and another set  $B$  of available slots. The well-known assignment problem [19] searches for the 1-to-1 assignment of cars to parking spaces, such that the sum of travel distances is minimized. However, in a world of selfish users, it is more reasonable to assign each car  $c \in A$  to the parking

---

<sup>\*</sup> Supported by grant HKU 7160/05E from Hong Kong RGC.

space  $p \in B$  that may not be taken by another driver  $c'$ , which happens to be closer to  $p$  than  $c$  is. Therefore, our formulation of the  $k$ ECP query (assuming that  $k$  is the minimum of cardinalities  $|A|$  and  $|B|$ ) searches for a practical solution to the problem.

We propose a technique for computing the ECP pairs efficiently given a set of cars and a set of parking slots. In addition, we extend it to monitor the ECP results, in a dynamic environment, where parking requests from cars and availability events from parking slots arrive from a data stream. Due to such events, ECP assignments must be deleted (i.e., when a car un-parks), new assignments must be added (i.e., when a new car requests parking), and current assignments may have to be changed. For instance, assume that pair  $(c, p)$  is in the current assignment and a new parking slot  $p'$  becomes available which is closer to  $c$  than  $p$  is. In this case,  $c$  must be re-assigned to  $p'$  and  $p$  should become available for other cars. This change may trigger a “chaining” effect which could alter the whole assignment. Our method processes incoming events in an appropriate order, such that the correct ECP results are maintained correctly and efficiently.

We assume that a centralized server monitors the locations of objects. When an object moves to another location, it informs the server about its new location. Since the frequent updates render disk-based management techniques inefficient, our solution is based on a memory grid-based indexing approach [16, 18].

Our contributions can be summarized as follows:

- We identify ECP as a new type of spatial join that finds application in real-life dynamic allocation problems (e.g., car/parking assignment).
- We show that the  $k$ ECP for  $k = \min\{|A|, |B|\}$  is equivalent to a special case of the stable marriage problem [7], where assignment preferences are derived from the distance function. Based on this observation, we adapt the Gale-Shapley algorithm [6] to solve  $k$ ECP queries by computing only a small fraction of the distances dynamically and on-demand.
- We define a dynamic version of ECP for moving objects and streaming events that indicate (i) availability of slots and (ii) demand for new  $k$ ECP pairs. We propose an appropriate extension of our static  $k$ ECP query evaluation algorithm that solves this continuous  $k$ ECP query.
- We conduct a set of experiments to verify the efficiency of the proposed methods for a wide range of problem parameters.

The rest of the paper is organized as follows. Section 2 surveys related work on closest pair queries in spatial data, continuous monitoring problems, and the stable marriage problem. Section 3 formally defines ECP and presents our solution to it for a static input. Section 4 presents an update framework for ECP calculation with two optimizations to improve its performance. Our solutions are evaluated in Section 5. Finally, Section 6 concludes the paper, giving directions for future work.

## 2 Background and Related Work

### 2.1 Closest Pairs Queries in Spatial Databases

Computation of closest pairs queries have been studied for several decades. Main-memory algorithms, such as the Neighbor Heuristic [1] and Fast Pair [3, 5], focus on

1CP problems. Fast Pair was shown to have the best overall performance. However, this method is not directly applicable to: (i)  $k$ CP queries for arbitrary values of  $k$ , and (ii) other variants of CP queries.

Some previous work [4, 9, 22] employ spatial indexes to solve  $k$ ICP queries in secondary memory. [4, 9] assume that the datasets are indexed by R-trees [8]. On the other hand, Yang et al. [22] extended the R-tree to a b-Rdnn tree, by augmenting each non-leaf entry with the maximum nearest neighbor distance (with respect to the other dataset) of points in its subtree. During query evaluation, such distances are utilized for reducing the search space. [22] showed that their approach outperforms previous R-tree based methods. Since these methods operate on indexed data they may not be applied in a dynamic environment. A high rate of streaming events imposes a high burden to the update of the indexes, which in combination with the expensive refreshing of the query results, renders the overall approach inefficient or impossible. In addition, although an  $k$ ICP algorithm can be tuned to process the  $k$ ECP query (i.e., by remembering assigned points and avoiding their re-assignment), such an approach would require a large amount of memory (for  $k = \min\{|A|, |B|\}$ , as large as the size of a dataset).

## 2.2 Continuous Monitoring of Spatial Queries

Various spatial applications, like the car-parking problem of the Introduction, handle large amounts of information at fast arrival rate. Several extensions of R-trees have been developed for supporting frequent updates of spatial data. Lee et al. [15] proposed the FUR-tree (Frequent Update R-tree), which uses localized bottom-up update strategies into the traditional R-tree. Recently, Xiong et al. [20] developed the RUM-tree (R-tree with Update Memo), which was shown to have better update performance than FUR-tree. [12] applied an event-driven approach to maintain query results for  $k$ NN and spatial join queries, with the assumption that moving objects can be modeled by linear motion functions.

Continuous monitoring of *multiple* spatial queries (e.g., range [16, 17] and  $k$ NN [21, 23, 18]) adopt the *shared execution paradigm* to reduce the processing cost. Instead of monitoring the results for different queries separately, the problem is viewed as a large spatial join between the query objects and data objects. As illustrated in Figure 1, grid cells (of cell length  $\delta$ ) are employed for indexing the objects. In practice,

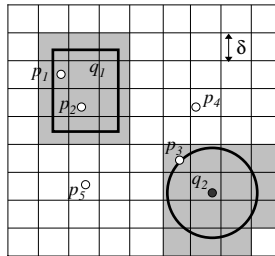


Fig. 1. Monitoring spatial queries

memory grid cells [23,18] are used (instead of disk-based structures) in order to handle very high update rate.  $q_1$  corresponds to a range query (shown in bold rectangle) and its *influence region* consists of the (gray) cells that intersect with  $q_1$ . Since data object updates outside the influence region cannot affect the query result, the processing cost is significantly reduced. As another example,  $q_2$  represents a NN query (shown in bold circle). Its difference from  $q_1$  is that its influence region is a circular region centered at  $q_2$  with dynamic radius equal its NN distance. For example, when the NN of  $q_2$  moves closer to (further from)  $q_2$ , then the influence region of  $q_2$  shrinks (grows).

Observe that continuous monitoring of range/ $k$ NN queries is different from that of  $k$ ECP queries. For range and  $k$ NN queries, only query results near a change triggered by a streaming event (i.e., appearance, disappearing, or movement of an object) need to be updated (i.e., only for queries whose influence region intersects the location of the change). On the other hand, as we will discuss in Section 4, a streaming event can generate a sequence of changes in the  $k$ ECP result. Thus, the idea of influence regions is not appropriate for  $k$ ECP monitoring, which calls for novel techniques.

### 2.3 The Stable Marriage Problem

The  $k$ ECP join is closely related to the classic stable marriage problem [6,7]. Given two set of objects  $A$  and  $B$ ,  $M$  is said to be a *matching* between  $A$  and  $B$  if (i)  $M$  is a set of  $\min\{|A|, |B|\}$  pairs of objects  $(a, b)$  where  $a \in A$ ,  $b \in B$ , and (ii) each object  $a \in A$  ( $b \in B$ ) appears in at most one pair in  $M$ . A matching  $M$  is *stable* if there are no pairs  $(a, b)$  and  $(a', b')$  in  $M$  such that  $a$  prefers  $b'$  to  $b$  and  $b'$  prefers  $a$  to  $a'$ . Given the preference lists of all objects  $a \in A$  and  $b \in B$ , the stable marriage problem seeks for a stable matching. In our context, the preference list of an object  $a$  is implicitly defined by the total order defined by the Euclidean distance; if  $a$  is closer to  $b$  than to  $b'$ , then  $a$  prefers  $b$  to  $b'$ .

[7] is a nice reference text that introduces the stable marriage problem and presents solutions to it, for special cases of the input. For the generic problem, Gale and Shapley [6] proved that, if  $|A| = |B|$ , it is always possible to find a solution and provided an algorithm for this. For the ease of discussion, we call the objects in  $A$  and  $B$  as senders and receivers, respectively. In the first round, each sender (in  $A$ ) calls its most preferred receiver (in  $B$ ). If a receiver hears from at least one sender, then the receiver matches with the best sender (according to the receiver's preference) and the corresponding sender is removed from  $A$ . The above procedure is applied iteratively in subsequent rounds, but with an additional rule: if a receiver has been assigned a sender  $a_{old}$  (in previous rounds) and now it hears from a better sender  $a_{new}$  (in the current round), then the receiver matches with the new sender and the remaining set of senders becomes  $A := \{a_{old}\} \cup A - \{a_{new}\}$ . Eventually, the stable matching between  $A$  and  $B$  is obtained after all objects in  $A$  or  $B$  have been removed.

For example, Table 1 illustrates a set  $A$  of three jobs and a set  $B$  of three applicants, such that the applicants (jobs) can be totally ordered based on their qualification (preference) for the job (applicant). In the first round of the Gale-Shapley algorithm, both jobs  $a_1$  and  $a_2$  call the applicant  $b_1$ , who prefers  $a_1$  to  $a_2$ . Thus,  $b_1$  matches with  $a_1$  and  $a_1$  is removed from  $A$ . Also,  $a_3$  calls  $b_2$ ,  $b_2$  matches with  $a_3$  and  $a_3$  is removed from  $A$ . In the second round,  $a_2$  calls  $b_2$ . Since  $b_2$  prefers the new job  $a_2$  to its old

**Table 1.** Example of stable marriage

Job	Preference	Applicant	Preference
$a_1$	$b_1 \succ b_3 \succ b_2$	$b_1$	$a_1 \succ a_3 \succ a_2$
$a_2$	$b_1 \succ b_2 \succ b_3$	$b_2$	$a_2 \succ a_3 \succ a_1$
$a_3$	$b_2 \succ b_3 \succ b_1$	$b_3$	$a_2 \succ a_1 \succ a_3$

job  $a_3$ ,  $b_2$  now matches with  $a_2$  instead and the job  $a_3$  is added back to  $A$ . In the third round,  $a_3$  calls  $b_3$  and  $b_3$  matches with  $a_3$ . Thus, the stable matching contains the pairs  $(a_1, b_1)$ ,  $(a_2, b_2)$ ,  $(a_3, b_3)$ . Note that at least one pair is finalized at each round, thus the worst-case time complexity of the algorithm is  $O(|A| \times |B|)$ .

The stable marriage algorithm is asymmetric; if the roles of  $A$  and  $B$  are reversed, a different solution may be found. Furthermore, it has been shown that it is sender-optimal (i.e.,  $A$ -optimal if  $A$  is the sender dataset); its execution will derive the optimal pair in  $B$  for any  $a \in A$ , for any order of examined objects from  $A$ . Thus, there is a unique solution when taking  $A$  as the sender input and another unique solution when taking  $B$  as the sender. We now prove that if the preference list is derived by a symmetric weight function  $w$  (e.g., Euclidean distance), such that  $w(a, b) = w(b, a)$ ,  $\forall a \in A, b \in B$ , then these two solutions are identical.

**Theorem 1.** *If preferences are defined by a weight function  $w$ , such that  $a$  prefers  $b$  to  $b'$  if and only if  $w(a, b) < w(a, b')$  and  $b$  prefers  $a$  to  $a'$  if and only if  $w(b, a) < w(b, a')$ , and  $w(a, b) = w(b, a)$ , for any  $a, a' \in A, b, b' \in B$  then the optimal stable marriage result is unique independently on whether  $A$  or  $B$  is the sender set.*

*Proof.* Without loss of generality, assume that  $|A| = |B| = n$ . Let  $M_A = \{(a_{(1)}, b_{(1)})\}, \{(a_{(2)}, b_{(2)})\}, \dots, \{(a_{(n)}, b_{(n)})\}\}$  be the  $A$ -optimal matching, such that  $(a_{(i)}, b_{(i)})$  models the pair which is finalized at the  $i$ -th loop of the Gale-Shapley algorithm. Let the  $B$ -optimal matching, generated by the Gale-Shapley algorithm, be  $M_B = \{(b'_{(1)}, a'_{(1)})\}, \{(b'_{(2)}, a'_{(2)})\}, \dots, \{(b'_{(n)}, a'_{(n)})\}\}$ . We will first prove that  $a_{(1)} = a'_{(1)}$  and  $b_{(1)} = b'_{(1)}$ , i.e., the first assignments output by the two runs of the algorithm are identical. Since  $(a_{(1)}, b_{(1)})$  is the first finalized pair of the  $A$ -sender run,  $w(a_{(1)}, b_{(1)})$  should be the smallest  $w(a, b)$ , for any  $a \in A, b \in B$ . Similarly,  $w(b'_{(1)}, a'_{(1)})$  should be the smallest  $w(b, a)$ , for any  $a \in A, b \in B$ . Since  $w(a, b) = w(b, a)$ , it must be  $a_{(1)} = a'_{(1)}$  and  $b_{(1)} = b'_{(1)}$ . By induction, we can prove that  $a_{(i)} = a'_{(i)}$  and  $b_{(i)} = b'_{(i)}$ , for  $1 \leq i \leq n$ , since by removing pairs  $\{(a_{(1)}, b_{(1)}), (a_{(2)}, b_{(2)}), \dots, (a_{(i)}, b_{(i)})\}$  from the problem we showed that the first pair  $(a_{(i+1)}, b_{(i+1)})$  in the resulting subproblem is identical for both  $A$ -sender and  $B$ -sender runs.  $\square$

A subtle issue to note is that the uniqueness argument for the  $A$ -sender (or  $B$ -sender) Gale-Shapley's output and Theorem 1 holds only for cases where the preference lists are *unique*, strictly total orders. Non-unique orders can be derived from weight functions

<sup>1</sup> Without loss of generality, we assume that only one pair is finalized at each loop. If there are multiple such pairs we could modify the algorithm to output only the one with the smallest  $w(a, b)$ , without affecting the correctness of the result.

$w$ , for which there exist pairs  $(a, b)$  and  $(a', b')$ , such that  $w(a, b) = w(a', b')$  and  $(a = a' \wedge b \neq b')$  or  $(a \neq a' \wedge b = b')$ . In such cases, e.g.,  $a = a' \wedge b \neq b'$ , object  $a$  has the same preference to  $b$  and  $b'$ , therefore the stable marriage result may not be unique; there could be a stable solution that includes  $(a, b)$  and another that includes  $(a, b')$ .

### 3 The Static $k$ ECP Query

In this section, we define and solve the *static* case of the  $k$ ECP query, where the  $k$ ECP result is requested for two sets of static points. For completeness, we also provide the definition of the  $k$  *inclusive* closest pairs ( $k$ ICP) query.

**Definition 1.** Given two set of points  $A, B$  and a  $k < |A \times B|$ , the  $k$  inclusive closest pairs  $kICP(A, B)$  is defined as the set  $S \subset A \times B$ , such that  $|S| = k$  and  $\forall (a, b) \in S, (a', b') \in (A \times B) - S, d(a, b) \leq d(a', b')$ .

**Definition 2.** Given two set of points  $A$  and  $B$ , the  $k$  exclusive closest pairs  $kECP(A, B)$  is recursively defined as:

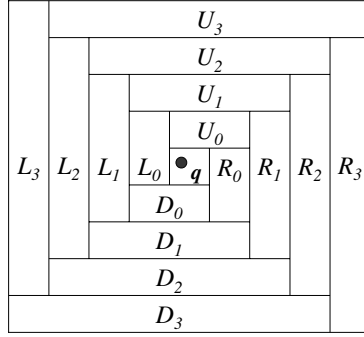
$kECP(A, B) = kICP(A, B)$ , for  $k = 1$ , and  
 $kECP(A, B) = 1ECP(A, B) \cup (k - 1)ECP(A - \{a\}, B - \{b\})$ , otherwise.

Note that the maximum possible value for  $k$  is  $\min\{|A|, |B|\}$  in  $kECP$  and  $k \leq |A| \cdot |B|$  in  $kICP$ . It is easy to prove that  $kECP$ , for  $k = \min\{|A|, |B|\}$  is a special case of the stable marriage problem, where the preference order is derived by the weight function  $w(a, b) = d(a, b)$  ( $d$  denotes Euclidean distance). Therefore the Gale-Shapley stable marriage algorithm (SMA) can be applied to solve  $kECP$  queries; the preference list of a point  $a \in A$  is constructed by placing points in  $B$  in ascending order of their distances to  $a$ . The preference lists of points  $b \in B$  are generated symmetrically. After running SMA on the preference lists, the obtained results correspond to the results of ECP. Since  $d(a, b) \equiv d(b, a)$ , and assuming that the distances between a point  $a \in A$  and the points in  $B$  are distinct (and vice versa<sup>2</sup>), SMA will derive the unique ECP result according to Theorem 1, no matter whether we take  $A$  or  $B$  as the sender set.

Nevertheless, the direct application of SMA requires the computation of a large number of distances and large space to store them (for  $|A| \cdot |B|$  distances), thus it does not scale well for large problems. We conjecture that the spatial properties of the query, in combination with appropriate indexes can be utilized to accelerate SMA. For example, we need not compute the distance of a point  $a \in A$  to *all* in  $B$  before running SMA; instead, we can applying spatial ranking techniques [10, 13] to generate the preference list of  $a$  incrementally and on-demand.

We adopt CPM; the grid-based technique of [13] for indexing data points in our problem, due to its good performance in environments with frequent updates. CPM is the state-of-the-art grid-based index for monitoring NN queries. Each query point is associated with a heap such that the objects and grid cells are visited in ascending order of their distances from the query point. In this way, query results can be computed fast and unnecessary accesses to other points are avoided. In particular, [13] propose a

<sup>2</sup> This is a realistic assumption since distances are real numbers and they are unlikely to coincide.



**Fig. 2.** Conceptual Partitioning Monitoring (CPM) space division

**Table 2.** Notation

Symbol	Description
$A, B$	a set of points (cars), a set of points (parking slots)
$a (b)$	a point in $A (B)$
$d(a, b)$	Euclidean distance between $a$ and $b$
$d_{min}(r, a)$	minimum distance between rectangle $r$ and point $a$
$DIR_{lvl}$	rect. of direction ( $DIR$ ) in level $lvl$ (in CPM)
$a.\psi (b.\psi)$	$a$ 's ( $b$ 's) current ECP point
$a.\lambda (b.\lambda)$	the distance between $a (b)$ and its ECP point

conceptual partitioning of the cells (see Figure 2) for reducing distance computations. Each rectangle  $DIR_{lvl}$  is associated with a direction  $DIR$  and a level number  $lvl$ . The direction can be  $U$  (up),  $D$  (down),  $L$  (left), or  $R$  (right). The level number denotes the number of rectangles between  $DIR_{lvl}$  and the cell containing the query point  $q$ .

Our static ECP algorithm (see Algorithm 1) uses the CPM index to search for the optimal matching, and (due to the hardness of the ECP problem) it is more sophisticated compared to the simple NN algorithm of [18]. Recall that we have two datasets  $A$  and  $B$  in our problem. In order to optimize performance, we consider the smallest dataset as a *query* set (that will generate nearest neighbor lists to be used as preference lists in the stable marriage evaluation). Accordingly, the other dataset represents an *objects* set. For the ease of exposition, let  $A$  be the query set and  $B$  be the objects set. For each point  $o$  in  $A$  and  $B$  keep track of the following information: (i) its current ECP object ( $o.\psi$ ), and (ii) the distance ( $o.\lambda$ ) to that object. Initially,  $o.\lambda$  is set to  $\infty$ , and  $o.\psi$  is set to NULL. Table 2 summarizes the notation used in our algorithm description.

In its initialization phase (Lines 1–5), SECP allocates a min-heap  $a.H$  for each object  $a \in A$ , and inserts in it  $Cell(a)$  (i.e., the cell containing  $a$ ) and all 0-level CPM rectangles (see Figure 2) that surround  $a$ . During SECP,  $a.H$  contains cells, rectangles, and/or objects from  $B$  and can identify the one with the smallest  $d_{min}$  to  $a$  in  $O(1)$



time<sup>3</sup>. In addition, all points in  $A$  are inserted to a *patients* set  $P$ , containing query points that have not found their exclusive closest pair yet.

SECP then starts a sequence of iterations (Lines 7-16); after each loop a number of ECP pairs are identified and inserted to the result. At the  $i$ -th iteration, for each query point  $a \in P$ , SECP incrementally retrieves from  $B$  nearest neighbors of  $a$  which are no further than the  $i$ -th level rectangle of the CPM partition and attempts to find the ECP pair of  $a$  in them (Lines 10-12).

---

**Algorithm 1.** SECP
 

---

```

V, P, P' : Queue
Result : Heap
algorithm SECP(Integer  $k$ )
1: for all  $a \in A$  do
2:   insert  $\langle Cell(a), d_{min}(a, Cell(a)) \rangle$  into  $a.H$ 
3:   for each direction  $DIR$  do
4:     insert  $\langle DIR_0, d_{min}(a, DIR_0) \rangle$  into  $a.H$ 
5:   insert  $a$  into  $P$ 
6:  $loop := 0$ 
7: while  $|Result| < k$  do
8:    $loop := loop + 1$ 
9:    $maxdist := (loop - 1/2) \cdot \delta$ 
10:  while  $P \neq \emptyset$  do
11:    dequeue an object  $a$  from  $P$ 
12:     $\epsilon$ -INNECP( $a, maxdist, V, P, P'$ )
13:  for all  $b \in V$  do
14:    if  $b = (b.\psi). \psi$  then
15:      insert  $(b.\psi, b)$  into  $Result$ 
16:   $P := P'; P' := \emptyset$ 

```

---

We now describe in more detail the core search module of SECP which is called at Line 12. Algorithm 2 is a pseudo-code for this  $\epsilon$ -bounded incremental nearest neighbor search with integrated ECP assignment ( $\epsilon$ -INNECP).  $\epsilon$ -INNECP browses the nearest neighbors of a query point  $a$  incrementally, subject to the constraint that their distance to  $a$  is not greater than  $\epsilon$ . At Lines 3-9, it processes the element on top of the  $a.H$  heap, if it is a rectangle or a cell, exactly like the original NN algorithm of [18]. If the next  $a.H$  entry is an object  $b$ , it is processed according to Lines 11-19. If  $d(a, b)$  is smaller than  $b.\lambda$  (this happens if  $b$  is unassigned or  $b$  has been previously assigned to a further query point), then the current ECP of  $a$  (resp.  $b$ ) is tentatively set to  $b$  (resp.  $a$ ). If  $b$  is unassigned, we insert it into a *candidates* list  $V$ . Otherwise, the previous assigned pair of  $b$  ( $b.\psi \in A$ ), is added to  $P$  and marked as unassigned. Then,  $b.\lambda$  and  $b.\psi$  are updated as  $d(a, b)$  and  $a$  respectively. Search terminates if  $a$  is assigned to a point  $b \in B$  (while-loop break of Line 19) or if  $a$  has not been assigned after all its  $\epsilon$ -bounded nearest neighbors in  $B$  have been examined. In the latter case,  $a$  is inserted

<sup>3</sup> Given a point  $p$  and a rectangle  $r$ ,  $d_{min}(p, r)$  is the minimum distance between  $p$  and any possible point in  $r$ .

into next loop's patients list  $P'$  (Line 21). Note that  $\epsilon$ -INNECP does not search for neighbors of  $a$  beyond  $\epsilon$  distance from  $a$ , and  $\epsilon$  is increased at each loop.

After each loop of SECP has examined all points in  $P$ , for each  $b$  in the candidate list  $V$ , it checks whether  $a = b.\psi$  has also  $a.\psi = b$  (Lines 13-15 of SECP). In this case  $(a, b)$  is definitely a pair in the ECP result. The reason is that  $d(a, b) \leq \epsilon$  and there could not be an unassigned neighbor to  $a$  (or  $b$ ) with a smaller distance (those have already been retrieved by  $\epsilon$ -INNECP). The algorithm terminates when the number of results reaches  $k$ . Otherwise,  $\epsilon$ -INNECP is invoked again with a new distance  $\epsilon = (\text{loop} - 0.5) \cdot \delta$ , where  $\text{loop}$  is the current loop and  $\delta$  is the extent of a grid cell.

---

**Algorithm 2.**  $\epsilon$ -bounded INN search and tentative ECP assignment
 

---

```

algorithm  $\epsilon$ -INNECP(Object  $a$ , Distance  $\epsilon$ , Queue  $V$ ,  $P$ ,  $P'$ )
1: while  $a.H \neq \emptyset$  and  $a.H$ 's top entry's distance  $\leq \epsilon$  do
2:    $\langle o, o_{dist} \rangle := deheap(a.H)$ 
3:   if  $o$  is a cell  $c$  then
4:     for all objects  $b' \in c$  do
5:       insert  $\langle b', d(a, b') \rangle$  into  $a.H$ 
6:   else if  $o$  is a rectangle  $DIR_{lvl}$  then
7:     for each cell  $c'$  in  $DIR_{lvl}$  do
8:       insert  $\langle c', d_{min}(a, c') \rangle$  into  $a.H$ 
9:     insert  $\langle DIR_{lvl+1}, d_{min}(a, DIR_{lvl+1}) \rangle$  into  $a.H$ 
10:  else  $\triangleright o$  is an object  $b$ 
11:    if  $b.\lambda > o_{dist}$  then  $\triangleright b$  prefers  $a$  to its previous pair
12:      set  $a.\psi := b$  and  $a.\lambda := o_{dist}$   $\triangleright$  update ECP for  $a$ 
13:      if  $b.\psi$  is NULL then
14:        insert  $b$  into  $V$   $\triangleright$  insert to ECP candidates  $V$ 
15:      else
16:         $(b.\psi).\psi := \text{NULL}; (b.\psi).\lambda := \infty$   $\triangleright$  unset pair of  $b$ 
17:        insert  $b.\psi$  into  $P$ 
18:        set  $b.\psi := a$  and  $b.\lambda := o_{dist}$   $\triangleright$  update ECP for  $b$ 
19:        break  $\triangleright$  break while-loop
20:  if  $a.\psi = \text{NULL}$  then  $\triangleright a$  has not been assigned in this loop
21:    insert  $a$  into  $P'$ 

```

---

Figure 3 exemplifies how SECP algorithm works. Assume that 4 cars (in  $A$ ) and 3 parking slots (in  $B$ ) remain unassigned after the first loop. Then,  $\epsilon$  is set to  $(2 - 0.5) * \delta$ , thus the maximum search range around each  $a \in P$  is shown by the gray circles in Figure 3a. Assume that the order of points in  $P$  is  $(a_1, a_2, a_3, a_4)$ . Figure 3b shows the running steps of this example in  $\text{loop}=2$ . At the first call of  $\epsilon$ -INNECP,  $a_1$  is assigned to  $b_1$ , since  $b_1$  is the NN of  $a_1$  and  $b_1$  is currently unassigned. Then,  $a_2$  takes  $b_1$  and  $a_1$  is put back to  $P$  (Lines 16-17 of  $\epsilon$ -INNECP). This happens because (i)  $b_1$  is the NN of  $a_2$  and (ii)  $a_1$  is the current ECP pair of  $b_1$  and  $d(a_2, b_1) < d(a_1, b_1)$ . The algorithm continues and eventually outputs the assignments  $(a_2, b_1)$  and  $(a_1, b_2)$ , whereas  $P' = \{a_3, a_4\}$  are moved to the next loop (so is  $b_3$ ). Although  $\epsilon$ -INNECP runs with a larger searching area in the next loop, it avoids accessing unnecessary elements, because it continues searching using the current min-heap  $a.H$  for each  $a \in P$ .

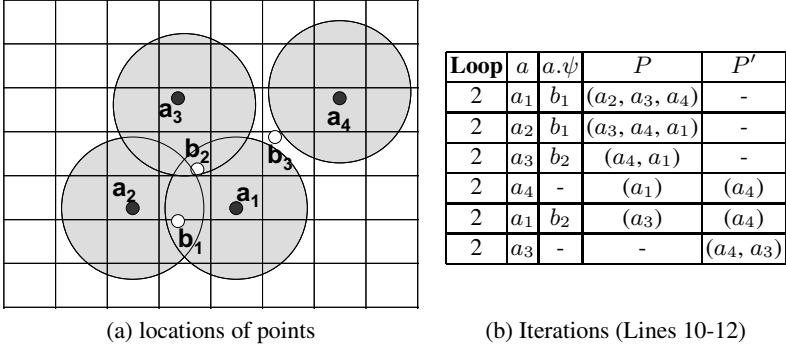


Fig. 3. An example of SECP ( $loop=2$ )

## 4 Continuous Monitoring of ECP Pairs

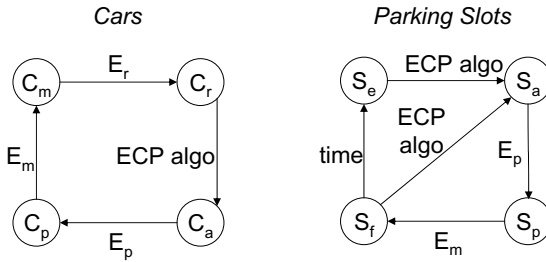
In this section, we set up the problem of monitoring ECP pairs dynamically and propose a solution that uses the SECP algorithm presented in the previous section. To motivate our problem setting, we base it on a realistic application, where the ECP join between a set of moving cars ( $C$ ) and a set of static parking slots ( $S$ ) is to be computed and incrementally maintained. When the car-parking assignment system starts up, it receives a number of events  $E_r$  from cars ( $c \in C$ ) in the monitored area, corresponding to assignment requests. It then runs a static ECP join algorithm to determine the slots to be assigned to these cars.

While the system is running, it receives events from cars and pushes them into a buffer  $Buf$ . At regular time intervals (e.g., every few seconds), the events collected in  $Buf$  are handled in batch. Three types of events are collected in  $Buf$ :  $E_r$  events from cars that have just requested to park,  $E_p$  events from cars that have just parked to their assigned slot, and  $E_m$  events from cars that have just unparked and they are moving. Accordingly, we can divide the sets of cars (and slots) into four classes based on their current state, as specified in Table 3. Figure 4 shows how streaming events or system decisions define the transitions of cars and parking slots among states. We assume that at each timestamp the system receives a number of  $E_r$ ,  $E_p$ , and  $E_m$  events from cars. First, all  $E_p$  events are processed, which change the statuses of the corresponding cars and slots from  $C_a$  to  $C_p$  and  $S_a$  to  $S_p$ , respectively. Then, the  $E_m$  events are processed and the corresponding cars in  $C_p$  and slots in  $S_p$  will move to classes  $C_m$  and  $S_f$ , respectively (we will explain the role of  $S_f$  shortly). Finally, the  $E_r$  events move cars from  $C_m$  state to  $C_r$  state. Unassigned cars in  $C_r$  and currently assigned cars in  $C_a$  must be processed by a *continuous* ECP algorithm based on the following.

- If an assigned car  $c \in C_a$  can be assigned a better slot (due to the availability of a new free parking slot which is closer) then perform this change.
- For all cars in  $c \in C_r$ , find their ECP pairs after having considered the optimal re-assignments for cars in  $C_a$ .

**Table 3.** Classification of objects based on their current status

Symbol	Description
$C_m$	set of cars which move and do not want to park
$C_r$	set of cars which move and request to park
$C_a$	set of cars which move and are assigned to a parking slot
$C_p$	set of parked cars
$S_e$	set of slots which are unoccupied and unassigned
$S_a$	set of slots which are assigned but not occupied
$S_p$	set of slots which are currently occupied
$S_f$	set of slots which are set free at the current timestamp

**Fig. 4.** State transition diagrams for objects

Note that a re-run of the ECP join for the union of  $C_r \cup C_a$  cars could result in the unfavorable assignment of a  $c \in C_a$  to a slot which is further than its currently assigned slot. In order to avoid such situations<sup>4</sup>, we must run a special version of ECP that handles cars in  $C_a$  separately.

Our continuous ECP algorithm (CECP) (see Algorithm 3) is based on the realistic assumption that only slots in  $S_f$  can change a current assignment  $(c_a, c_a.\psi)$  for  $c_a \in C_a$  to a better one. The rationale is that once assigned to its slot,  $c_a$  will have moved towards it, so it is unlikely for a slot in  $S_e$  (i.e., the empty slots from the previous timestamp) will suit  $c_a$  now (since it did not suit it in the previous timestamp). Based on this assertion, we examine all slots in  $S_f$  to see if any of them could change the current assignment of a  $c_a \in C_a$  to a better one. If a slot  $s_f \in S_f$  can replace the current assignment  $c_a.\psi$  of a car  $c_a$ , we perform this change and push  $c_a.\psi$  to  $S_f$  (since it could update the assignment of another car). Otherwise, we put  $s_f$  to  $S_e$  (the set of empty slots). After all slots in  $S_f$  have been examined and the set becomes empty, we perform a static ECP join for the pair of requesting cars and empty slots  $(C_r, S_e)$ . For this join, we use the SECP algorithm described in Section 3. We now discuss two optimization techniques for speeding up the search operation at Line 3 of CECP.

<sup>4</sup> Imagine that you've been assigned to a parking and while moving towards it, the system informs you that you have to change to a further slot!

**Algorithm 3.** Continuous ECP

---

```

algorithm CECP( $C, S$ )
1: while  $S_f \neq \emptyset$  do ▷ first phase
2:    $s_f :=$  remove slot  $s_f$  from  $S_f$ 
3:   if for a  $c_a \in C_a$   $d(c_a, c_a.\psi) > d(c_a, s_f)$  then
4:     move  $c_a.\psi$  to  $S_f$ ; set  $c_a.\psi := s_f$ 
5:     move  $s_f$  to  $S_a$ ;
6:   else
7:     move  $s_f$  to  $S_e$ ;
8: SECP( $C_r, S_e$ ) ▷ second phase

```

---

**4.1 Distance-Bounded Search**

For each  $s_f$ , CECP scans  $C_a$  to find a car  $c_a \in C_a$  for which  $s_f$  can replace  $c_a.\psi$  or verify that no such car exists in  $C_a$ . This search can be accelerated if the cars in  $C_a$  are checked in increasing distance from  $s_f$ . Therefore, before CECP begins for the current timestamp, we organize the existing  $C_a$  (from the previous timestamp) in a CPM index. In addition, we compute the maximum distance  $\Gamma$  of any assigned pair in  $C_a$  (i.e.,  $\Gamma = \max\{d(c_a, c_a.\psi) | c_a \in C_a\}$ ). This preprocessing phase requires a only single pass over  $C_a$ , whereas the resulting index can be used for any  $s_f \in S_f$ .

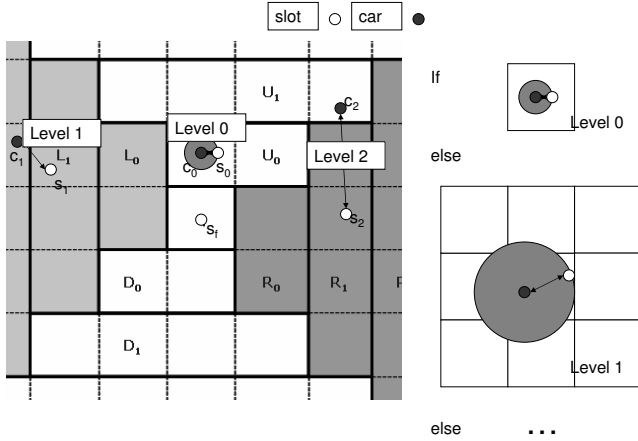
For each  $s_f$ , we examine the objects  $c_a \in C_a$  incrementally according to their distance to  $s_f$  (i.e., we perform a NN search on the CPM-index [18]). This way, the chances to find an assignment for  $s_f$  early are maximized because assigned cars close to  $s_f$  are examined earlier. More importantly, NN search can terminate as soon as  $d(s_f, c_a) \geq \Gamma$ , for a neighbor  $c_a$  of  $s_f$ .

**4.2 Partitioning in CPM Cells**

Recall that each  $s_f \in S_f$  attempts to find any  $c_a \in C_a$ , for which  $\text{dist}(c_a, s_f) < \text{dist}(c_a, c_a.\psi)$ . If the distance between  $c_a$  and its assigned slot  $s_a$  ( $c_a.\psi$ ) is smaller than the minimum distance between  $c_a$  and the boundary of the CMP cell  $\text{Cell}(c_a)$  which encloses  $c_a$  (i.e.,  $d(c_a, c_a.\psi) \leq d_{\min}(c_a, \text{Cell}(c_a))$ ), then  $c_a$  cannot be re-assigned to any  $s_f$  outside  $\text{Cell}(c_a)$ . For example, consider three assigned pairs  $(c_0, s_0)$ ,  $(c_1, s_1)$ ,  $(c_2, s_2)$ , and a newly available slot  $s_f$ , as shown in Figure 5. Since  $d(c_0, s_0) \leq d_{\min}(c_0, \text{Cell}(c_0))$  and  $s_f \notin \text{Cell}(c_0)$ , we know that  $c_0$  cannot be re-assigned to  $s_f$ .

We can extend this argument for arbitrary cars as follows. For each  $c_a \in C_a$ , we define  $\text{level}(c_a)$  to be the minimum number of CPM levels around  $\text{Cell}(c_a)$  such that  $c_a$  cannot be re-assigned to  $s_f$ , for any  $s_f$  further than these levels. This can be computed by comparing  $d(c_a, c_a.\psi)$  to  $d_{\min}(c_a, L)$  where  $L$  is the boundary (MBR) of successive cell layers around  $c_a$ . For example, in Figure 5,  $\text{level}(c_0) = 0$ ,  $\text{level}(c_1) = 1$ , and  $\text{level}(c_2) = 2$ .

The idea behind our second optimization is to partition the cars  $c_a$  in each cell, based on their  $\text{level}(c_a)$ . For example, in Figure 5,  $c_0$  belongs to the level-0 partition of  $\text{Cell}(c_0)$ ,  $c_1$  belongs to the level-1 partition of  $\text{Cell}(c_1)$ , and  $c_2$  belongs to the level-2 partition of  $\text{Cell}(c_2)$ . Then, for each  $s_f$ , when we examine a cell  $C$  during NN search, we only check all  $c_a \in C$ , for which  $\text{level}(c_a) \geq s_f.\text{cpmlevel}$ , where  $s_f.\text{cpmlevel}$  is



**Fig. 5.** Partitioning of objects to levels

the current search level around  $s_f$ . The further  $C$  is from  $s_f$  the more partitions inside it will be pruned. For example, in Figure 5, while searching for a better assignment containing  $s_f$ , when visiting  $Cell(c_0)$ , we don't have to check its level-0 partition (which contains  $c_1$ ). Similarly, when visiting  $Cell(c_2)$ , we can prune its level-0 and level-1 partitions (but not the level-2 partition which contains  $c_2$ ; therefore  $c_2$  has to be examined).

## 5 Experimental Evaluation

In this section we experimentally evaluate the efficiency of our proposed ECP algorithms using synthetic data. First, we compare the SECP algorithm proposed in Section 3 with two alternative approaches to the same problem. Second, we validate CECP; the algorithm for continuous monitoring of ECP pairs proposed in Section 4. The algorithms were implemented in C++ and all experiments were performed on a Pentium IV 1.8GHz machine with 512MB memory, running Windows XP.

### 5.1 ECP Computation

To our knowledge, this is the first paper studying ECP computation, so there are no previous approaches to compare SECP with. Clearly, computing the distances between all pairs of points and running the stable-marriage algorithm (SMA) would be very inefficient. Alternatively, we compare SECP with two alternative methods, which (like SECP) avoid computing all distances:

- **$p$ INN ECP search.** This method is similar to our SECP. The difference is that at each step it (incrementally) fills a list with the next  $p$  nearest neighbors in  $B$  for each unassigned  $A$ -point, where  $p$  is an input parameter of the algorithm. Given the  $p$ NN lists of all such  $A$ -points to their  $p$ -th neighbors, let  $\epsilon$  be the smallest of these distances. We can use this distance as a bound and run Lines 10-12 of SECP

to finalize ECP pairs for some of the  $A$ -points. For this purpose, we directly use the  $p$ NN lists, instead of re-computing the nearest points by running  $\epsilon$ -INNECP. At each step, after all  $A$ -points have been processed, some of them will have found their ECP pair. For the remaining ones, we continue the INN search until their NN set contains exactly  $p$  neighbors. For these points we repeat the whole process at the next step.

- **1INN ECP search.** In the initial state of the 1INN ECP algorithm, for each unassigned query  $a$  we maintain a CPM heap  $H$  for it and use it to find the nearest CPM element of  $a$  (this could be a rectangle, a cell, or an object). The nearest elements of all unassigned  $a \in A$  are stored in a candidate queue (CQ) which is a priority queue organizing them in ascending order of the distance. At each step of the algorithm, we pop the top element from CQ. If this is a rectangle or a cell, we proceed to find the next nearest CPM element of the corresponding query object and push it into CQ. If the popped element from CQ is an unassigned object, it must be the ECP result of the corresponding query (by definition). If it is an assigned one, we ignore it and get the next nearest neighbor of the query object, which is pushed back to CQ. This process is continued until all ECP results are computed.

We evaluate the performance of the static ECP algorithms with synthetic datasets (to study their scalability with respect to various parameters and due to lack of real data for ECP problems). In each dataset, the coordinates of points are random values uniformly generated in a  $[0, 10000] \times [0, 10000]$  space. By default, the total number of queries and objects is 100K and there are as many objects as queries (i.e.,  $|A| = |B|$  and  $|A| + |B| = 100K$ ). By default the CPM grid used was  $128 \times 128$  and the value of  $p$  used by the  $p$ INN ECP search algorithm is 8 (we found out by experimentation that this method performs best for  $p = 8$ ). The  $k$  parameter of the ECP join is set to  $k = \min\{|A|, |B|\}$  (i.e., we seek for the maximum possible assignment).

Figure 6 shows the performances of the three static ECP algorithms for CPM grid sizes  $|G| \times |G|$  ranging from  $32 \times 32$  to  $256 \times 256$ . Although the grid sizes with the best CPU time performance are between  $64 \times 64$  and  $96 \times 96$ ,  $|G| = 128$  presents a good trade-off between the CPU time and the memory usage by all three algorithms. Furthermore, as we will show later, the  $128 \times 128$  grid outperforms other sizes when larger amounts of data are searched. Note that our SECP algorithm outperforms the other two methods, while having only slightly higher memory requirements than 1INN ECP. The reason behind the good performance of SECP is that (i) unlike  $p$ INN ECP, it searches only up to the necessary nearest neighbors for each query and (ii) it avoids using and updating the huge CQ heap of 1INN ECP.

Figure 7 compares the three algorithms for various grid sizes and database sizes  $|O| = |A| + |B|$ . The results are consistent with the previous experiment. SECP performs the best in terms of CPU while the costs of  $p$ INN and 1INN are more sensitive to the database size. Note that when the number of objects increases finer grids become more efficient; this is expected since the space becomes denser and using a finer partitioning pays off. Note that more memory is required for smaller grid sizes, since more individual objects (instead of cells and rectangles) enter the search heaps of the queries. Again, SECP has slightly higher memory requirements than 1INN. Finally, Figure 8 shows the performance as a function of different data size ratios ( $|A|/|B|$ ) and database

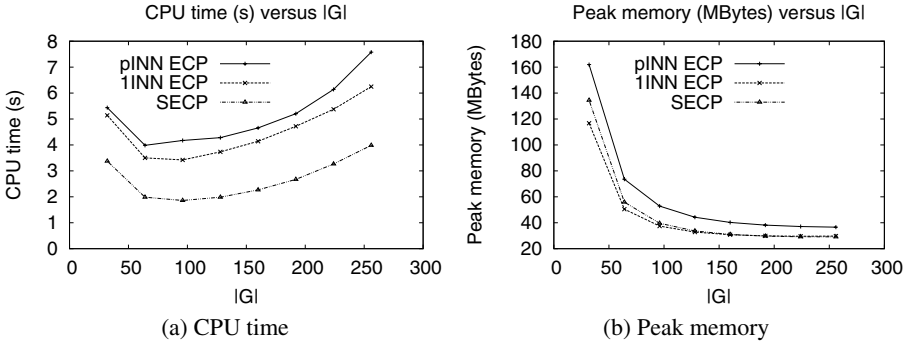


Fig. 6. Effect of  $|G|$

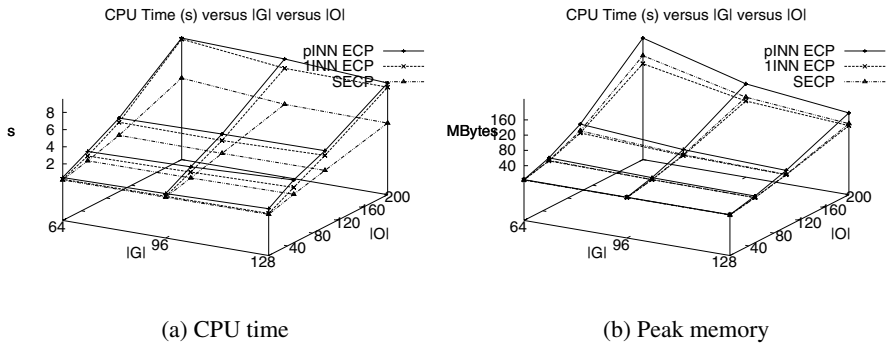


Fig. 7. Combined effect of  $|G|$  and  $|O|$

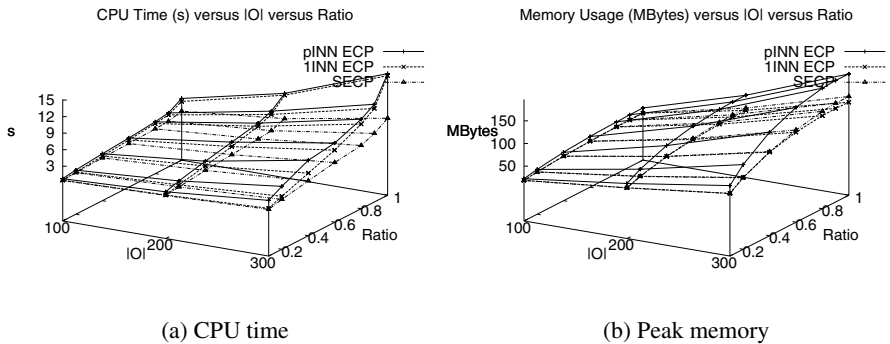


Fig. 8. Combined effect of  $|O|$  and  $|A|/|B|$  ratio

sizes ( $|O| = |A| + |B|$ ). SECP has the best performance and its relative difference to other methods increases with the database size and  $|A|/|B|$  ratio. We do not need to consider ratios larger than 1, since the ECP computation is symmetric (the smallest dataset is taken as the *query* dataset  $A$ ).



## 5.2 Maintenance of ECP Results

We developed a data generator that simulates a real-life car-parking assignment problem and monitoring problem, based on the specifications of Section 4. The generator starts with a set of parking slots and a set of cars which are uniformly distributed in a  $[0, 10000] \times [0, 10000]$  space. A parking-request probability  $P_{req}$ , an unparking probability  $P_{unpark}$ , and a velocity  $V$  are assigned to each car. Initially, all cars are moving to a random direction and they request for parking with probability  $P_{req}$  at each timestamp. If a car  $c$  issues a parking request to the system ( $E_r$ ) it moves to the parking request state and the system attempts to assign a slot to it. Once a slot  $s$  is assigned to  $c$ ,  $c$  moves towards  $s$  according to its velocity and when it reaches  $s$  it parks, issuing a  $E_p$  event. After  $c$  has parked, at each subsequent timestamp it has  $P_{unpark}$  probability to issue a  $E_m$  event. A car that unparks sets its slot free and starts moving to a direction 90 degrees different than its direction when moving towards its parking slot. At each timestamp, the system processes all incoming events according to Section 4.

Table 4 shows the parameters of the generator, their range of values and their default value in bold font. In each experiment, only one parameter varies while the others are fixed to their default values. We measured the average CPU cost and memory requirements of the CECP algorithm for each timestamp, after letting the system to run for 1000 timestamps.

In the first experiment, we verify the effectiveness of the optimizations of Sections 4.1 and 4.2 in CECP. These optimizations aim at reducing the re-assignment cost for cars in  $C_a$  using  $S_f$  (i.e., Lines 1–7 of Algorithm 3). Figure 9 shows the re-assignment cost of CECP without, with one (CECP+O1 or CECP+O2), and with both (CECP+O1+O2) optimizations, for different time instants with different sizes of  $C_a$  and  $S_f$ . Optimization 2 (Section 4.2) incurs larger improvement compared to optimization 1 (Section 4.1) since it avoids additional accesses. The combination of both methods result in the best performance for CECP at all settings. In the remaining experiments we use both optimizations in the first phase of CECP.

Figure 10a shows the average performance per timestamp of both CECP phases for different values of  $P_{req}$ . The first phase (i.e., the handling of  $S_f$  and  $C_a$ ) uses both optimizations of Sections 4.1 and 4.2. The second phase (Line 8 of Algorithm 3) is performed by the SECP algorithm. For small values of  $P_{req}$  the distances between assigned cars and their slots tend to be large, a fact that increases the cost of CECP’s first phase (as many re-assignments are performed). Larger  $P_{req}$  reduces the cost of the first phase due to the decrease of the average distance between assigned pairs. On the other hand,

**Table 4.** Stream generation parameters

Parameter	Values
Number of cars, $ C $	<b>600K</b>
Number of slots, $ S $	<b>150K</b>
Parking request probability, $P_{req}\%$	0.5%, 1%, <b>2%</b> , 4%, 8%
Unparking probability, $P_{unpark}\%$	0.5%, 1%, <b>2%</b> , 4%, 8%
Average velocity of cars, $V$	1.67, 3.33, <b>5.27</b> , 6.67, 13.33

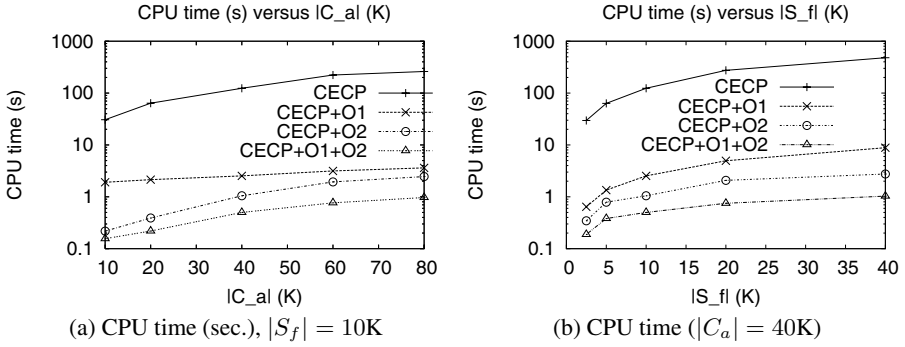


Fig. 9. Effect of  $|C|$  and  $|S_f|$

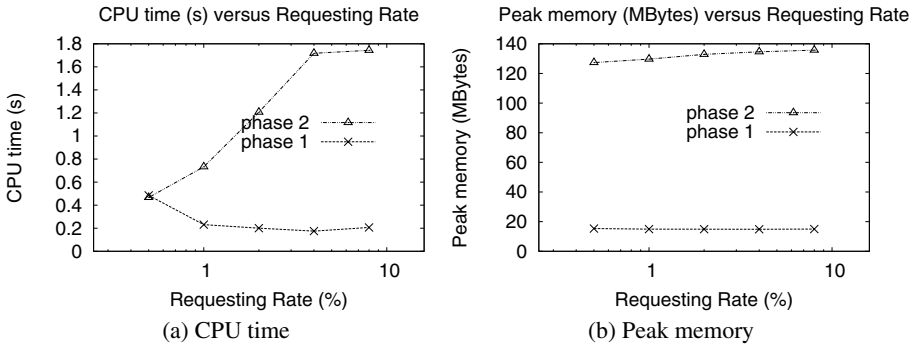


Fig. 10. Effect of requesting rate

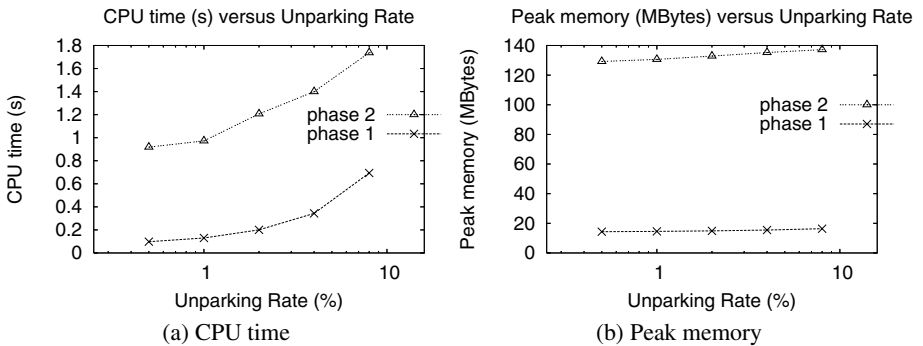
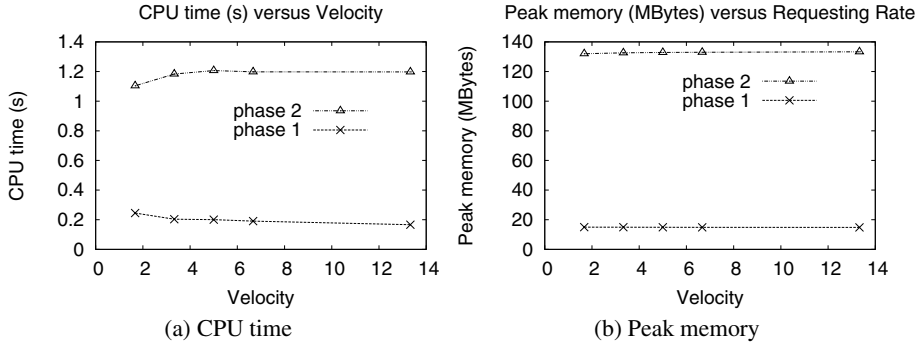


Fig. 11. Effect of leaving rate

as  $P_{req}$  increases  $|C_r|$  becomes larger and the second phase of CECP becomes more expensive. Figure 10a shows that the memory requirements of both phases of CECP are slightly affected by  $P_{req}$ , with the same trend as the CPU time difference.



**Fig. 12.** Effect of different velocities

Figure 11 shows the effect of  $P_{unpark}$  on the performance of the algorithm, after fixing  $P_{req}$  and  $V$  to their default values. There is a slight increase on the CPU time and memory requirements for both phases as  $P_{unpark}$  increases (due to the increase of  $|S_f|$ ). Finally, Figure 12 shows that our problem is not sensitive to the objects velocity ( $P_{req}$  and  $P_{unpark}$  are fixed to their default values).

## 6 Conclusion

In this paper we identified the exclusive closest pairs (ECP) problem, which is a spatial assignment problem. A motivating application of it is the matching of cars and parking slots. We proposed an efficient main-memory algorithm for solving the static version of the problem. In addition, we defined the problem of continuous monitoring ECP pairs in a dynamic environment where assignment requests and de-assignment notifications arrive from a stream. We presented a thorough experimental evaluation that demonstrates the efficiency of the proposed solutions on synthetically generated data that simulate a real-life dynamic car/parking assignment problem. In the future, we will consider other types of one-to-one assignments (e.g., finding and maintaining an assignment that minimizes an aggregate distance).

## References

1. Anderberg, M.R.: Cluster Analysis for Applications. Academic Press, Inc, London (1973)
2. Brinkhoff, T., Kriegel, H.-P., Seeger, B.: Efficient processing of spatial joins using r-trees. In: SIGMOD Conference, pp. 237–246 (1993)
3. Cardinal, J., Eppstein, D.: Lazy algorithms for dynamic closest pair with arbitrary distance measures. In: Algorithm Engineering and Experiments Workshop (ALENEX) (2004)
4. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Closest pair queries in spatial databases. In: SIGMOD Conference, pp. 189–200 (2000)
5. Eppstein, D.: Fast hierarchical clustering and other applications of dynamic closest pairs. ACM Journal of Experimental Algorithms 5(1) (2000)

6. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. *Amer. Math.* 69, 9–14 (1962)
7. Gusfield, D., Irving, R.W.: *The Stable Marriage Problem, Structure and Algorithms*. MIT Press, Cambridge (1989)
8. Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: *SIGMOD* (1984)
9. Hjaltason, G.R., Samet, H.: Incremental distance join algorithms for spatial databases. In: *SIGMOD Conference*, pp. 237–248 (1998)
10. Hjaltason, G.R., Samet, H.: Distance browsing in spatial databases. *ACM Trans. Database Syst.* 24(2), 265–318 (1999)
11. Hu, H., Xu, J., Lee, D.L.: A generic framework for monitoring continuous spatial queries over moving objects. In: *SIGMOD Conference* (2005)
12. Iwerks, G.S., Samet, H., Smith, K.P.: Maintenance of k-nn and spatial join queries on continuously moving points. *ACM Trans. Database Syst.* 31(2), 485–536 (2006)
13. Koudas, N., Ooi, B.C., Tan, K.-L., Zhang, R.: Approximate nn queries on streams with guaranteed error/performance bounds. In: *VLDB*, pp. 804–815 (2004)
14. Koudas, N., Sevcik, K.C.: High dimensional similarity joins: Algorithms and performance evaluation. In: *ICDE* (1998)
15. Lee, M.-L., Hsu, W., Jensen, C.S., Cui, B., Teo, K.L.: Supporting frequent updates in r-trees: A bottom-up approach. In: *VLDB*, pp. 608–619 (2003)
16. Mokbel, M.F., Xiong, X., Aref, W.G.: Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In: *SIGMOD Conference*, pp. 623–634 (2004)
17. Mokbel, M.F., Xiong, X., Hammad, M.A., Aref, W.G.: Continuous query processing of spatio-temporal data streams in place. In: *STDBM*, pp. 57–64 (2004)
18. Mouratidis, K., Papadias, D., Hadjieleftheriou, M.: Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In: *SIGMOD Conference*, pp. 634–645 (2005)
19. Nering, E.D., Tucker, A.W.: *Linear Programs & Related Problems: A Volume in the Computer Science and Scientific Computing Series*. Academic Press, Inc. London (1992)
20. Xiong, X., Aref, W.G.: R-trees with update memos. In: *ICDE* (2006)
21. Xiong, X., Mokbel, M.F., Aref, W.G.: Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In: *ICDE*, pp. 643–654 (2005)
22. Yang, C., Lin, K.-I.: An index structure for improving nearest closest pairs and related join queries in spatial databases. In: *IDEAS*, pp. 140–149 (2002)
23. Yu, X., Pu, K.Q., Koudas, N.: Monitoring k-nearest neighbor queries over moving objects. In: *ICDE*, pp. 631–642 (2005)
24. Zhang, J., Mamoulis, N., Papadias, D., Tao, Y.: All-nearest-neighbors queries in spatial databases. In: *SSDBM*, pp. 297–306 (2004)

# Continuous Evaluation of Fastest Path Queries on Road Networks

Chia-Chen Lee<sup>1</sup>, Yi-Hung Wu<sup>2</sup>, and Arbee L.P. Chen<sup>3,\*</sup>

<sup>1</sup> Department of Computer Science, National Tsing Hua University, Taiwan, R.O.C.  
pasha1105@gmail.com

<sup>2</sup> Department of Information & Computer Engineering, Chung Yuan Christian University,  
Taiwan, R.O.C.

yhwu@ice.cycu.edu.tw

<sup>3</sup> Department of Computer Science, National Chengchi University, Taiwan, R.O.C.  
alpchen@cs.nccu.edu.tw

**Abstract.** The one-shot shortest path query has been studied for decades. However, in the applications on road networks, users are actually interested in the path with the minimum travel time (the *fastest path*), which varies as time goes. This motivates us to study the continuous evaluation of fastest path queries in order to capture the dynamics of road networks. Repeatedly evaluating a large number of fastest path queries at every moment is infeasible due to its computationally expensive cost. We propose a novel approach that employs the concept of the affecting area and the tolerance parameter to avoid the reevaluation while the travel time of the current answer is close enough to that of the fastest path. Furthermore, a grid-based index is designed to achieve the efficient processing of multiple queries. Experiments on real datasets show significant reduction on the total amount of reevaluation and therefore the cost for reevaluating a query.

**Keywords:** Fastest Path, Road Network, Continuous Query Processing.

## 1 Introduction

When driving on the road, the driver often wants to know the best path from the current position to the target place. Most of the existing navigation systems find the path with the shortest distance in response to the query. However, the traffic conditions often vary as time goes and the user actually wants to know the path with the minimum travel time even if it may not have the shortest distance. Such need incurs a demand for the *fastest path query* on road networks.

The advances in GPS and sensor networks enable us to get an object's position and the traffic conditions, and to perform the fastest path query in real time. We represent the road network as an undirected and weighted graph, called the *travel time network*, as described in [11]. In this graph, edges represent road segments, vertices denote the intersections and end points of the road segments, and the weight attached to each

---

\* Corresponding author.

edge stands for the travel time for the associated road segment. The fastest path query posted on the travel time network contains an *origin point* and a *destination point* in an arbitrary location of an edge. The query answer should be the path between the two points with the minimal total weight of the edges in the path in the entire graph. Evaluating a one-shot fastest path query is equivalent to evaluating the single source shortest path problem addressed in [2], and can be solved by the conventional algorithms such as Dijkstra's algorithm [4] or A\* search [13][14].

Given a large graph, the execution time of both the above methods can be huge. Two kinds of approaches were developed for performance improvement. One relies on the pre-computation of the shortest paths, and the other uses the skills of space transformation. The former approaches [1][5][8][9][10] pre-compute and tabulate the shortest paths between all pairs of vertices in the entire or a part of the graph. The shortest path can then be found by a table lookup. The latter approaches [7][17][18] transform the graph into another representation in which a shortest path between two points can be found in constant time. However, both approaches described above are not suitable for the graph we consider in this paper, where the weights (travel time) may change as time goes. Under this setting, the fastest path initially computed may become invalid at some instant. That is, whenever the travel time of a road segment changes, the "pre-computation" approach has to re-compute the fastest paths in the table previously built, and the "space transformation" approach needs to rerun the encoding method for all nodes in order to fit the current network. Therefore, several studies are made to investigate the processing of fastest path queries in a dynamic environment.

Fu and Rilett [6] solve the dynamic and stochastic shortest path problem by modeling the travel time of each road segment as a continuous stochastic process to estimate the travel time and find the expected fastest path. Kanoulas et al. [12] introduce the Time-Interval All Fastest Path (*allFP*) query. The *speed pattern*, which specifies the predicted speeds in different time periods for each road segment, is given. The fastest paths corresponding to different time periods can then be derived. Both of these works adopt the predicted travel time to find the expected fastest path, which may not be the true fastest path.

Frigioni et al. [5] take into account the changes on edge weights, and repeatedly find the shortest paths between the origin and the destination. Its first search is the same as A\*, but the subsequent searches can be much faster because it reuses those parts in the previous search tree that do not change. Unfortunately, it is inadequate to preserve the information of all previous search trees while a large number of queries are simultaneously running. Moreover, this method can be applied only when the edge with a changed weight is close to the origin or the destination.

To our knowledge, there is no satisfactory solution to the problem of the continuous fastest path query processing. An efficient method to continuously keep track of the fastest paths for multiple queries on the travel time network is required. An intuitive way to process the continuous fastest path query is to periodically re-compute the fastest path. However, this approach will repeatedly rerun a query even when its fastest path does not change. Moreover, the processing time of a fastest path query increases with the number of nodes in the network. If a huge number of fastest path queries are simultaneously executed in a large network, the computational cost may become too much to afford. Therefore, in this paper we propose a novel

approach, named the *ellipse bounding method (EBM)*, to eliminate the unnecessary reevaluations of fastest path queries and to reduce the search space for finding the fastest path.

Given the maximal speed  $V$  of a moving object and a fastest path query  $Q = (p1, p2)$  and its current answer with travel time  $T$ , we can compute the maximal distance  $D=V \times T$  this moving object can travel from  $p1$  to  $p2$  within time  $T$ . We want to derive an area for  $Q$  according to  $D$  such that any path containing a point that falls out of this area cannot be a faster path than the current answer. As Figure 1(a) shows, for any point  $p$ ,  $d1$  and  $d2$  denote the *Euclidean distances* from  $p1$  and  $p2$  to  $p$  respectively. The *network distance* of a path is defined as the total weight of the edges in this path. The network distance from  $p1$  to  $p2$  through  $p$  must be larger than or equal to  $d1$  plus  $d2$ . This is because the Euclidean distance between two points is a lower bound of their network distance. We have the following property that if a path contains a point  $p$  such that  $d1$  plus  $d2$  is larger than  $D$  then it cannot be a faster path than the current answer. Mathematically, the points with an equal sum of the distances to two fixed points form the curve of an ellipse. We therefore define the region within this ellipse as the *affecting area* (the gray area in Figure 1(b)) of the given query.

According to the affecting area, we can determine whether the current answer is invalid. A query should be reevaluated only if the travel time of the current answer increases or in its affecting area the travel time of an edge not in the current answer decreases. Moreover, once we decide to re-compute the fastest path for a query, the search space is also limited to the corresponding affecting area.

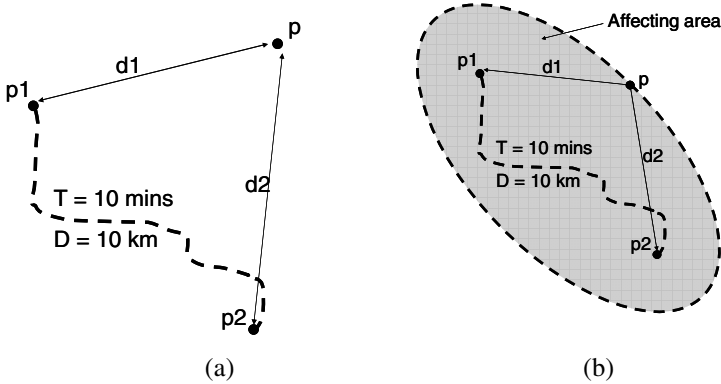


Fig. 1. Concept of the affecting area

To process more than one continuous fastest path query simultaneously, we design a grid-based index structure to record all the affecting areas. While an edge’s weight changes, we find all the queries whose affecting areas cover this edge to re-compute the corresponding fastest paths. The affecting area changes when the object moves toward its destination or the travel time of the current path varies. The maintenance of the grid-based index by dynamically updating the affected areas will result in considerable overheads. We design a mechanism to efficiently perform the updates with the accuracy of the query answers guaranteed.

The rest of the paper is organized as follows. The definitions and storage of the travel time network are described in Section 2. In Section 3, the main idea of EBM and the methods for query processing are presented. Section 4 describes how to maintain index in a dynamic environment. Section 5 experimentally evaluates the proposed method. Finally, a summary of our work and some future works are provided in Section 6.

## 2 Travel Time Network

In this section, the problem definition and the storage structure for the travel time network are introduced.

### 2.1 Problem Definition

In this paper, a road network together with its current traffic is modeled as the travel time network [11], which can be defined as follows.

**Definition 1.** *Given a road network, the travel time network is a directed weighted graph  $G = \{V, E, W\}$ , where the vertices in  $V$  are the end points or the intersections of the road segments, each edge  $e$ , denoted by  $\langle n_i, n_j \rangle$ , in  $E$  is the road segments with start point  $n_i$  and end point  $n_j$ , and each weight in  $W$  is the travel time of the corresponding road segment.*

In the road network, we use a point  $p$  represented by  $\langle n_i, n_j, d \rangle$  to indicate a location in  $\langle n_i, n_j \rangle$  where  $d$  is the travel time from  $n_i$  to  $p$ .

**Definition 2.** *A sub-edge of an edge  $e = \langle n_i, n_j \rangle$  in the travel time network is denoted by  $\langle n_i, p \rangle$  or  $\langle p, n_j \rangle$  where  $p$  is a point in  $e$ . The weight of a sub-edge  $\langle n_i, p \rangle$*

*( $\langle p, n_j \rangle$ ) is equal to  $\frac{|\overline{n_i p}|}{|\overline{n_i n_j}|} \cdot w_e$  ( $\frac{|\overline{p n_j}|}{|\overline{n_i n_j}|} \cdot w_e$ ) where  $|\overline{n_a n_b}|$  is the length of  $\langle n_a, n_b \rangle$  and  $w_e$  is the weight of  $e$ .*

In the following, we define a path, the path weight, and the fastest path, respectively.

**Definition 3.** *A path  $P$  from  $p_1$  to  $p_2$  is formed by two sub-edges, and several conjunctive edges. We denote  $P$  as  $\{\langle p_1, n_1 \rangle, \langle n_1, n_2 \rangle, \dots, \langle n_{k-1}, n_k \rangle, \langle n_k, p_2 \rangle\}$ . The path weight is the sum of the weights associated with the edges and sub-edges on  $P$ . More than one path may exist from a specific point to another, and the one with the minimum path weight is termed the fastest path.*

The travel time of a road segment will vary as the traffic varies, and so does the fastest path for a travel, the movement from the origin to the destination. However, if the difference between the travel times of the new fastest path and the old one is below a threshold, the new fastest path does not need to be computed. We propose the concepts of the tolerance parameter and the continuous fastest path query in Definitions 4 and 5 respectively.

**Definition 4.** *A tolerance parameter is a user-defined time period, denoted by  $\Delta T$ . If the difference between the travel times of the new fastest path and the old one is lower*



than  $\Delta T$ , we say that the old fastest path is tolerable and there is no need to re-compute the new fastest path; otherwise, we say that the old fastest path expires.

As a result, the problem we consider in this paper can be formulated as follows.

**Definition 5.** According to a tolerance parameter, the continuous fastest path query processing keeps monitoring and returning the new fastest path for the user when needed to reach the destination.

### 2.2 Basic Storage

There are two basic storage components, the *adjacency list* and the *edge table*. The *adjacency list* preserves the connectivity among vertices. Each vertex in the graph is stored as a node in the adjacency list and associated with three pieces of information, including the vertex’s identifier, its coordinates, and a pointer to a list of all the adjacent vertices. Referring to the gray table in Figure 2(b), each node  $u$  corresponds to a vertex in Figure 2(a). In addition to the identifier and the coordinates,  $u$  is also followed by a list of pairs in the form of  $(v, w)$ , indicating an adjacent vertex  $v$  and the edge weight  $w$  of  $\langle u, v \rangle$ . For example, the pair (2, 18) in the linked list of the first node  $n_1$  represents an edge from  $n_1$  to  $n_2$  with weight 18.

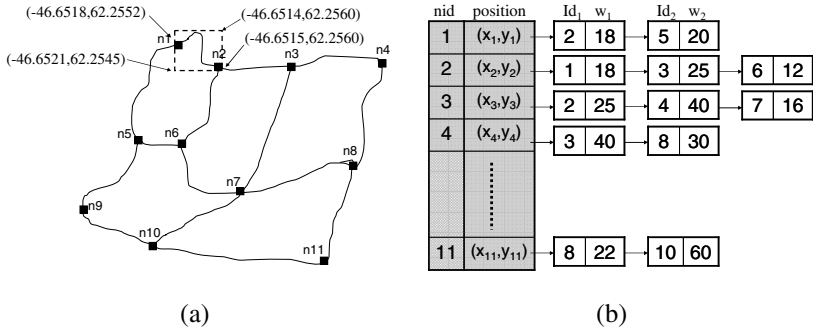


Fig. 2. (a) a road network (b) the adjacency list (c) the edge table

In order to retrieve the spatial information of edges, the *minimum bounding rectangle* (MBR) is adopted to build the *edge table*. Since the MBRs of  $\langle n_i, n_j \rangle$  and  $\langle n_j, n_i \rangle$  are identical, we therefore only store one copy of them by keeping the indices in the order of  $n_i, n_j$  where  $i < j$ . For instance, the rectangle in dotted lines in Figure 2(a) is the MBR of edge  $\langle n_1, n_2 \rangle$  and  $\langle n_2, n_1 \rangle$ . For each edge, in addition to the coordinates of end points, two more pieces of information, i.e., the left-bottom and right-top coordinates of the corresponding MBR, are stored. Figure 2(c) shows the coordinates of  $\langle n_1, n_2 \rangle$  and  $\langle n_2, n_1 \rangle$ .

### 3 Ellipse Bounding Method

Time is conceptually divided into discrete slots in our work. In this way, edge weights and query answers are updated only at the beginning of each time slot. Some notations we will use in the rest of this paper are summarized in Table 1.

In this section, we first illustrate the concept of the affecting area, which is used to reduce unnecessary re-computations and to prune the search space while the queries are reevaluated. The property is then applied to the method for continuous query processing.

**Table 1.** Summary of notations

Notation	Description
$t_j$	The $j^{\text{th}}$ time slot
$P_{i,j}$	The path returned to the $i^{\text{th}}$ query at the beginning of $t_j$
$P_{i,j}^*$	The fastest path of the $i^{\text{th}}$ query at the beginning of $t_j$
$T_j(P_{i,j-1})$	The travel time of $P_{i,j-1}$ at the beginning of $t_j$
$T_{i,j}^*$	The travel time of $P_{i,j}^*$
$\Delta T_i$	The tolerance parameter of the $i^{\text{th}}$ query
$d_N(p_1, p_2)$	The network distance between $p_1$ and $p_2$
$d_E(p_1, p_2)$	The Euclidean distance between $p_1$ and $p_2$

#### 3.1 Affecting Area

The affecting area is mainly developed from the *ellipse bounding property* as described below.

**Property 1.** Given two points  $p_1$  and  $p_2$ , a path  $P$  between them with travel time  $T$ , and the highest speed of the object movement  $V$ , any path faster than  $P$  must entirely fall within an ellipse  $E$ , in which the foci are  $p_1$  and  $p_2$  and the length of the major axis is  $T \times V$ .

**Proof:** Assume that a path  $P'$  is faster than  $P$  and does not entirely fall in  $E$ . Let  $q$  be a point on  $P'$  but not in  $E$ . By definition,  $d_N(p_1, p_2)$  via  $P' = d_N(p_1, q) + d_N(q, p_2) \geq d_E(p_1, q) + d_E(q, p_2)$ . Moreover, the maximum length of  $P$  can be computed as  $T \times V$ , which equals the length of the major axis, i.e.,  $d_E(p_1, x) + d_E(x, p_2)$ , for any  $x$  on  $E$ . Since  $q$  is located outside  $E$ ,  $d_E(p_1, q) + d_E(q, p_2) > d_E(p_1, x) + d_E(x, p_2) = T \times V$ . Therefore,  $d_N(p_1, p_2)$  via  $P' > T \times V$  and the minimum travel time of  $P' > (T \times V) / V = T$ , which contradicts the assumption. ■

**Example 1.** In Figure 3,  $P$  is a path between  $p_1$  and  $p_2$  with travel time 5 minutes. The highest speed  $V$  is 2 km/min.  $E$  is the ellipse in which the foci are  $p_1$  and  $p_2$  and the length of the major axis is 10km.  $P'$  is a path that is not entirely covered by  $E$ . We know that the length of  $P'$  is larger than  $d_1 + d_2$ , and  $d_1 + d_2$  is larger than 10km. Therefore, even if a user move from  $p_1$  to  $p_2$  along  $P'$  at the highest speed 2 km/min, the travel time will be larger than 5 minutes. It means that any path not entirely in  $E$  cannot be faster than  $P$ .

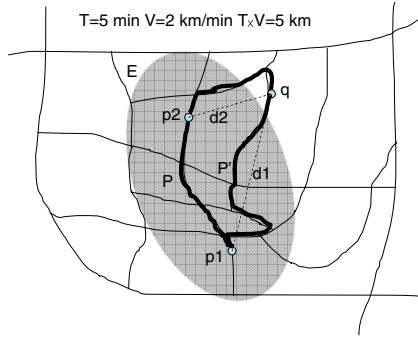


Fig. 3. Illustration of the ellipse bounding property

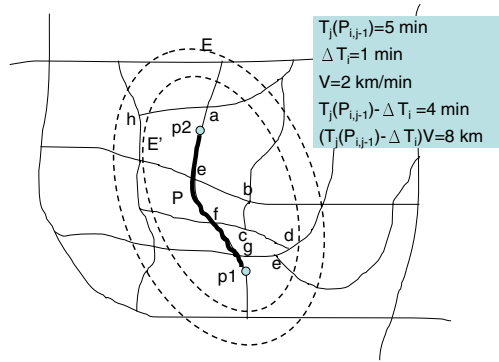


Fig. 4. The affecting area defined by the tolerance parameter

For simplicity, we use  $E(p_1, p_2, D)$  to represent an ellipse with foci are  $p_1$  and  $p_2$ , and the length of its major axis is  $D$ . For each query, we only care if the current answer  $P$  is *tolerable*, i.e., its travel time  $T$  is lower than or equal to the travel time of the fastest path  $T^*$  plus the tolerance parameter  $\Delta T$ . In other words,  $P$  is tolerable if  $T - T^* \leq \Delta T$ , i.e.  $T^* \geq T - \Delta T$ . According to Property 1, any path with travel time lower than  $T - \Delta T$  must entirely fall within the ellipse  $E(o, d, (T - \Delta T) \times V)$ , where  $o$  and  $d$  denote the origin and destination of the query respectively and  $V$  is the highest speed. We define the *affecting area* of each query  $q_i$  at  $t_j$  as the ellipse  $E(o_{i,j}, d_i, (T_j(P_{i,j-1}) - \Delta T_i) \times V)$ , where  $o_{i,j}$  and  $d_i$  are the origin of  $q_i$  at  $t_j$  and its destination, respectively. If there is an edge with its weight decreased, we have to check whether there is a path passing this edge and its travel time less than  $T_j(P_{i,j-1}) - \Delta T_i$ . Since this path must entirely fall within the affecting area of the query, the check should be done only if the edge with weight decreased is entirely covered in the affecting area.

**Example 2.** Figure 4 shows how the ellipse bounding property still works even when the tolerance parameter is used. Assume that the travel time  $T_j(P_{i,j-1})$  of the current answer  $P_{i,j-1}$  is 5 minutes and the tolerance parameter  $\Delta T_i$  is 1 minute. If the travel time of the fastest path is more than 4 minutes,  $P$  is tolerable and we do not have to search the fastest path for the user. That is, we find and return the fastest path only

when its travel time is less than 4 minutes. With the tolerance parameter, we derive the affecting area  $E'$  as described above and thus the length of its major axis is 8 km. By property 1, only if the weight of an edge in the set  $\{<e,b>, <f,c>, <b,c>, <c,d>, <d,e>, <e,g>\}$  decreases,  $P$  may expire and the query needs to be reevaluated.

Notice that  $E$  is the ellipse in which the length of its major axis is  $T_j(P_{i,j-1}) \times V$  and  $<a,h>$  is covered by  $E$ . If the weight of  $<a,h>$  decreases, any path passing  $<a,h>$  will be faster. However, by property 1, its travel time will never be lower than  $T_j(P_{i,j-1}) - \Delta T_i$ . Therefore, in this case  $P_{i,j-1}$  is always tolerable and the query needs not be reevaluated. In this way, unnecessary computation is avoided while the quality of the returned answers, specified by the tolerance parameter, is guaranteed.

### 3.2 Query Processing

In this subsection, we first focus on single query processing and then describe how to process multiple queries. In the case of single query processing, only query  $i$  exists in the system. Initially, the fastest path from the given origin to the destination is computed by the Dijkstra's algorithm. At the beginning of each subsequent time slot  $k$ , two tasks, *update* and *check*, are performed. The first task is to update the current origin of the query, the travel time network, and the affecting area. We have to update the affecting area since both the origin and the travel time network, according to which the affecting area is computed, vary. The second task examines the query answer to decide whether any edges with weights updated have impact on its accuracy, i.e., it is tolerable or expires. If the query answer is tolerable, it is not necessary to reevaluate the query and thus the considerable cost can be saved. The two conditions for making this decision are described below.

Suppose that the query answer at the beginning of  $t_{j-1}$  is  $P_{i,j-1}$  and some edge weights are updated in  $t_{j-1}$ . We compute the travel time  $T_j(P_{i,j-1})$  at the beginning of  $t_j$  according to the current origin and the travel time network. To see whether the answer  $P_{i,j-1}$  is still tolerable, i.e.,  $T_{i,j}^* \geq T_j(P_{i,j-1}) - \Delta T_i$ , a natural but costly way is to directly reevaluate the query for getting  $T_{i,j}^*$ . For efficiency, before the query reevaluation, we apply two necessary conditions that must be satisfied for the above inequality to fail. For clarity, we denote the set of edges entirely falling in the affecting area of query  $i$  at the beginning of  $t_j$  as  $E_{ij}$ .

1. Any edge weight on  $P_{i,j-1}$  increases: Since the increased weight makes the travel time higher than what we expected previously, the query must be reevaluated no matter whether any other edge is updated.
2. At least one edge  $e$  in  $E_{ij}$  but not on  $P_{i,j-1}$  has its weight decreased: Since there exist some paths other than  $P_{i,j-1}$  in the affecting area with decreased travel time, the answer  $P_{i,j-1}$  may expire and the query reevaluation is needed.

The above conditions come from the property that the query answer  $P_{i,j-1}$  expires only if the weight of an edge on it increases or the weight of an edge in  $E_{ij}$  but not on  $P_{i,j-1}$  decreases. If neither of them holds, by property 1 the answer  $P_{i,j-1}$  is tolerable at  $t_j$  and therefore it is not necessary to reevaluate the query. We perform the two tasks for every time slot until the destination is reached.

While there are more than one query posed in the system, the initial answers are computed similarly as soon as the queries are posed and the update task for each

query is performed at the beginning of every time slot. If any edge has its weight updated, the conditions described above are checked to identify which queries must be reevaluated. The query that satisfies the first condition, implying that the travel time of its current answer increases, is called an *increased query*. The query that satisfies the second condition, implying that its affecting area entirely contains an edge with a decreased weight, is called an *affected query*. Both kinds of queries are then put into a *Pool of Queries to be Reevaluated* (abbreviated as PQR). Finally, each query in this pool is reevaluated.

### 4 Grid-Based Index for Efficient Query Reevaluation

In every time slot, EBM consists of two main tasks, update and check, as the two dotted rectangles shown in Figure 5. For the first task, in addition to the update of the travel time network and the origins of queries, the MBRs of queries are also either

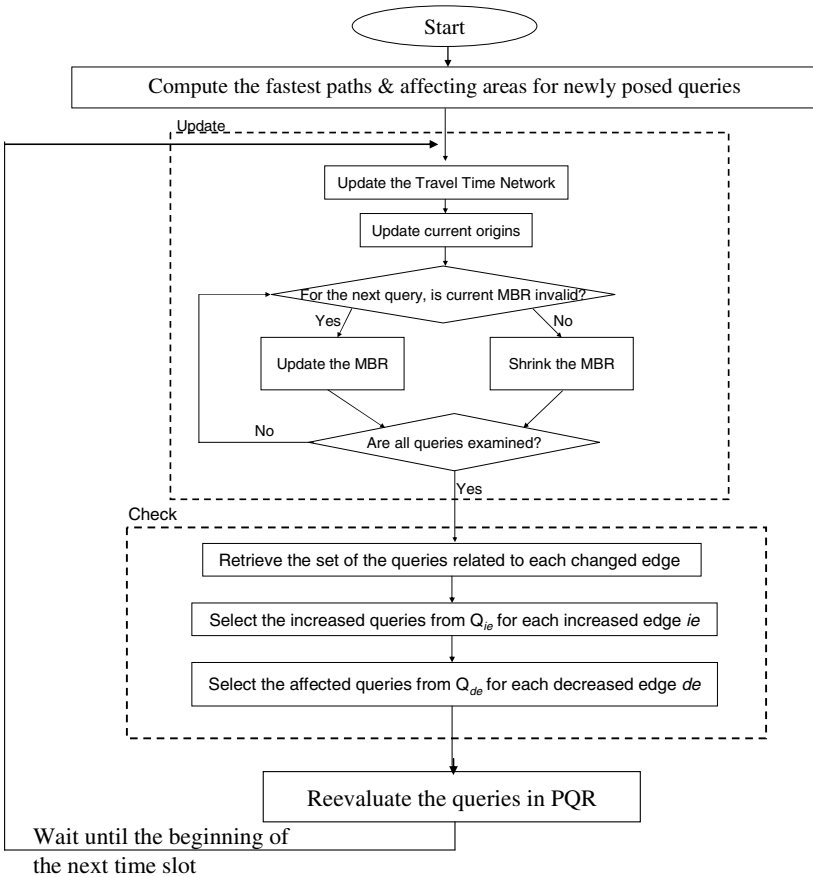


Fig. 5. The flow chart of EBM

updated or shrunk. For the second task, each updated edge is used to select either kind of queries for reevaluation. In this section, for convenience of presentation, we first introduce the construction of our grid index and then show how it is used in the second task. After that, we present the manipulation of the MBRs for queries in the first task.

To select the queries for PQR, the edges with their weights updated are related to the queries by checking their spatial relationships with the current answers (for increased queries) and the affecting areas (for affected queries). Finding the queries related to an edge based on their spatial relationships is called a *spatial join*. There are two kinds of spatial joins, corresponding to the two kinds of queries in PQR, respectively. One returns the queries whose current answers contain the given edge (for increased queries), while the other returns the queries whose affecting areas entirely contain the given edge (for affected queries). Since the spatial joins are time-consuming, an efficient method to perform them is required. We design a grid-based indexing method for this purpose and introduce it in the following.

#### 4.1 Index Construction and Utilization

Conceptually, we decompose the entire 2D-space of the travel time network into fixed-sized cells, each with width  $\alpha$  and height  $\beta$ . The affecting area of each query is represented by the MBR (minimum bounding rectangle) that exactly covers the corresponding ellipse. Specifically, the query identifier is stored in each of the cells covered by or intersects with the MBR corresponding to its affecting area. To find such cells for a query, we adopt formulae derived from a geometric property to compute the bottom-left and top-right coordinates of the MBR for the corresponding ellipse. The related definitions and the geometric property are given in the following.

**Definition 6.** Given two points  $p_1$  and  $p_2$  on the 2D-space, we define their minimum distance constrained by a straight line  $L$ , denoted as  $MD(p_1, p_2, L)$ , as the minimum of the sum of Euclidean distances from  $p_1$  to a point  $q$  on  $L$  and from  $q$  to  $p_2$ , i.e.,  $\min\{d_E(p_1, q) + d_E(p_2, q)\}$  for any  $q$  on  $L$ . In our work, the straight line  $L$  can only be either vertical ( $x=a$ ) or horizontal ( $y=a$ ).

**Property 2.** Let  $p_1$  and  $p_2$  be two points with coordinates  $(x_1, y_1)$  and  $(x_2, y_2)$ . We have:

$$(a) \quad MD(p_1, p_2, x=a) = \sqrt{(X_+ - 2a)^2 + Y_-^2}$$

$$(b) \quad MD(p_1, p_2, y=a) = \sqrt{X_-^2 + (Y_+ - 2a)^2}$$

where  $X_+ = x_1 + x_2$ ,  $X_- = |x_1 - x_2|$ ,  $Y_+ = y_1 + y_2$ , and  $Y_- = |y_1 - y_2|$ .

**Proof:** (a) Consider the points  $p_1(x_1, y_1)$ ,  $p_2(x_2, y_2)$  and the vertical line  $L$  in Figure 6. By the plane geometry, we have the mirror point  $p_1'(2a - x_1, y_1)$  of  $p_1$  across  $L$ . Moreover, for any point  $q$  on  $L$  triangles  $\triangle p_1 q n$  and  $\triangle p_1' q n$  are congruent and  $d_E(p_1, q)$  equals  $d_E(p_1', q)$ . Thus,  $d_E(p_1, q) + d_E(p_2, q)$  can be replaced by  $d_E(p_1', q) + d_E(p_2, q)$ . Let  $m$  be the intersection of  $L$  and the line segment  $\overline{p_1' p_2}$ . By the triangular inequality, we have  $d_E(p_1', m) + d_E(m, p_2) = d_E(p_1', p_2) \leq d_E(p_1', q) + d_E(p_2, q)$  for any point  $q$  on  $L$ .  $MD(p_1, p_2, L) = d_E(p_1', p_2) = \sqrt{(x_2 - (2a - x_1))^2 + (y_2 - y_1)^2} = \sqrt{(X_+ - 2a)^2 + Y_-^2}$ , where  $X_+ = x_1 + x_2$  and  $Y_- = |y_1 - y_2|$ . Similarly, (b) can also be proved by replacing  $L$  with a horizontal line.  $\blacksquare$

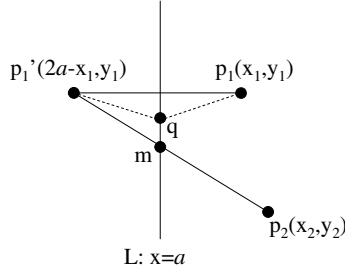


Fig. 6. The MD function

**Definition 7.** For an ellipse  $E(o,d,D)$  where  $D$  is the length of its major axis, the vertical line tangent to the left (right) of  $E$  is denoted as  $S_{left}$  ( $S_{right}$ ). Similarly, the horizontal line tangent to the top (bottom) of  $E$  is denoted as  $S_{top}$  ( $S_{bottom}$ ). By property 2, we can compute  $MD(o,d,S_{left})$ ,  $MD(o,d,S_{right})$ ,  $MD(o,d,S_{top})$ , and  $MD(o,d,S_{bottom})$ , which are abbreviated as  $MD_{left}$ ,  $MD_{right}$ ,  $MD_{top}$ , and  $MD_{bottom}$  respectively.

**Property 3.** Given an ellipse  $E(o,d,D)$  whose foci are  $(x_o, y_o)$  and  $(x_d, y_d)$ , the bottom-left and top-right coordinates of the corresponding MBR are:

$$\left(\sqrt{X_+ - (D^2 - Y_-^2)}/2, \sqrt{Y_+ - (D^2 - X_-^2)}/2\right), \left(\sqrt{X_+ + (D^2 - Y_-^2)}/2, \sqrt{Y_+ + (D^2 - X_-^2)}/2\right)$$

where  $X_+ = x_o + x_d$ ,  $X_- = |x_o - x_d|$ ,  $Y_+ = y_o + y_d$ , and  $Y_- = |y_o - y_d|$ .

**Proof:** According to the plane geometry, the four sides of the corresponding MBR are  $S_{left}$ ,  $S_{right}$ ,  $S_{top}$ , and  $S_{bottom}$  as defined above. The two coordinates can be derived from the property that each side of the MBR is tangent to  $E$ . Without loss of generality, we use  $S_{left}$ :  $x=a$  to show the derivation. Let  $m$  be the intersection of  $E$  and  $S_{left}$ . Since the other points on  $S_{left}$  are all outside  $E$ ,  $m$  is the only point getting  $MD(o,d,S_{left})$ . By property 2 and the definition of ellipse,  $MD_{left} = \sqrt{(X_+ - 2a)^2 + Y_-^2} = D$ , where  $X_+ = x_o + x_d$  and  $Y_- = |y_o - y_d|$ . Since  $S_{left}$  is on the left of  $E$ ,  $2a < X_+$ . Solving this equation will get  $a = X_+ - \sqrt{D^2 - Y_-^2}/2$ , which is the  $x$ -value of the bottom-left coordinate of the MBR. Similarly, the other three values of the two coordinates can also be derived from the remaining sides, respectively. ■

By using the above formulae, when a query is posed, all the cells covered by its MBR are located and then the query identifier is stored into these cells. As a result, each cell is associated with a set of query identifiers. An example grid recording  $q_1$  and  $q_2$  is shown in Figure 7. When an edge is updated, we use the grid to select all the queries relevant to this edge in two steps. At first, the MBR corresponding to this edge is retrieved from the edge table and all the cells covered by it are collected. We then compute the intersection of all the sets of query identifiers in these cells. If a query is not in the intersection, neither its MBR nor its affecting area will entirely cover the edge. Let the set of query identifiers in the intersection be noted as  $Q$ . The second step further examines  $Q$  and proceeds in two cases, depending on how the edge is updated.

**Case 1.** If the edge weight is increased, the increased queries in  $Q$  are selected. For each query in  $Q$ , the edges in its current answer are examined. A query is an increased query and will be selected into  $PQR$  if its current answer contains the increased edge.

**Case 2.** If the edge weight is decreased, the affected queries in  $Q$  are selected. Since to select all queries whose affecting area entirely covers a given edge is costly, we select a subset of the affected queries without destroying the guarantee of the answer precision. Notice that, in this stage, the queries selected are fewer than all affected queries, and this can save more unnecessary reevaluations. A rule is applied to prune the queries that need not be reevaluated, as expressed by Property 4 below.

**Property 4.** Given an decreased edge  $e = \langle n_1, n_2 \rangle$  at the beginning of  $t^k$ , we define  $SQ_e$  as the set of queries in which every query  $q$  satisfies the following inequality:

$$\min\{d_E(p_1, n_1) + \text{len}_e + d_E(n_2, p_2), d_E(p_1, n_2) + \text{len}_e + d_E(n_1, p_2)\} \leq D,$$

where  $p_1$  and  $p_2$  are the origin and destination of  $q$ ,  $D$  is the major axis length of  $q$ 's affecting area, i.e.  $D = (T_k(P_{qk-1}) - \Delta T_q) \times V$ , and  $\text{len}_e$  is the length (network distance) of  $e$ . Moreover, let  $AQ$  be the set of all affected queries covering  $e$ . Then, the following two conditions always hold:

- 1)  $SQ_e$  is a subset of  $AQ$ .
- 2) For each query not in  $SQ_e$ , no path from  $p_1$  to  $p_2$  containing  $e$  can have travel time lower than or equal to  $T_k(P_{qk-1}) - \Delta T_q$ .

**Proof:**

1) We prove this condition by showing that any query  $q$  not in  $AQ$  will not belong to  $SQ_e$ . Let  $E(p_1, p_2, D)$  denote the affecting area of such  $q$ . Since  $E$ , the affecting area of  $q$ , does not entirely cover  $e$ , at least one point in  $e$  must be outside  $E$ . We assume the point  $n_x$  on  $e$  is outside  $E$  and thus  $\text{len}_e = d_N(n_1, n_x) + d_N(n_x, n_2)$  and  $d_E(p_1, n_x) + d_E(n_x, p_2) > D$ . The left-hand side of the above inequality considers the following two sums of distances:

(a)  $d_E(p_1, n_1) + \text{len}_e + d_E(n_2, p_2)$ : According to the triangle inequality, we obtain that  $d_E(p_1, n_1) + d_E(n_1, n_x) \geq d_E(p_1, n_x)$  and  $d_E(n_x, n_2) + d_E(n_2, p_2) \geq d_E(n_x, p_2)$ . Therefore, we have that  $d_E(p_1, n_1) + d_N(n_1, n_x) + d_N(n_x, n_2) + d_E(n_2, p_2) \geq d_E(p_1, n_1) + d_E(n_1, n_x) + d_E(n_x, n_2) + d_E(n_2, p_2) \geq d_E(p_1, n_x) + d_E(n_x, p_2) > D$ .

(b)  $d_E(p_1, n_2) + \text{len}_e + d_E(n_1, p_2)$ : Similarly, because  $d_E(p_1, n_2) + d_E(n_2, n_x) \geq d_E(p_1, n_x)$  and  $d_E(n_x, n_1) + d_E(n_1, p_2) \geq d_E(n_x, p_2)$ , we have that  $d_E(p_1, n_2) + d_N(n_2, n_x) + d_N(n_x, n_1) + d_E(n_1, p_2) \geq d_E(p_1, n_2) + d_E(n_2, n_x) + d_E(n_x, n_1) + d_E(n_1, p_2) \geq d_E(p_1, n_x) + d_E(n_x, p_2) > D$ .

From the above, we conclude that  $q$  is not in  $SQ$  and thus  $SQ$  is a subset of  $AQ$ .

2) Let  $E(p_1, p_2, D)$  denote the affecting area of  $q$ , which is not in  $SQ$ . The network distance of the path  $P$  containing  $e$  is equal to  $\min\{d_N(p_1, n_1) + \text{len}_e + d_N(n_2, p_2), d_N(p_1, n_2) + \text{len}_e + d_N(n_1, p_2)\}$ , which is larger than or equal to  $\min\{d_E(p_1, n_1) + \text{len}_e + d_E(n_2, p_2), d_E(p_1, n_2) + \text{len}_e + d_E(n_1, p_2)\}$ . Because  $q$  is not in  $SQ$ , the minimum length of such  $P > D$  and thus the minimum travel time of such  $P > D/V = T_k(P_{qk-1}) - \Delta T_q$ . ■

Suppose that  $q$  is a query not in any  $SQ_e$  where  $e$  is the edge with decreased weight. By Property 4, we have that any path containing a decreased edge from the origin to the destination of  $q$  will have travel time larger than or equal to  $T_k(P_{qk-1}) - \Delta T_q$ . Moreover, all the paths which contains no decreased edge for  $q$  also can not have travel time smaller than  $T_k(P_{qk-1}) - \Delta T_q$ . That is the current answer of  $q$  is tolerable and do not need to be reevaluated. Thus, we only select the queries in  $SQ_e$  for each decreased edge  $e$  into  $PQR$ .



Consider the grid in Figure 7 as an example. If the edge  $\langle n_4, n_{20} \rangle$  has its weight decreased, we apply Case 2 to find the affected queries. Since this edge covers the four cells (5,5), (5,6), (6,5), and (6,6), only  $q_1$  is left in the intersection after the first step. In the second step,  $q_1$  is not selected as  $SQ_{\langle n_4, n_{20} \rangle}$  and is not put into the PQR when examining  $\langle n_4, n_{20} \rangle$  because the inequality in property 4 does not hold.

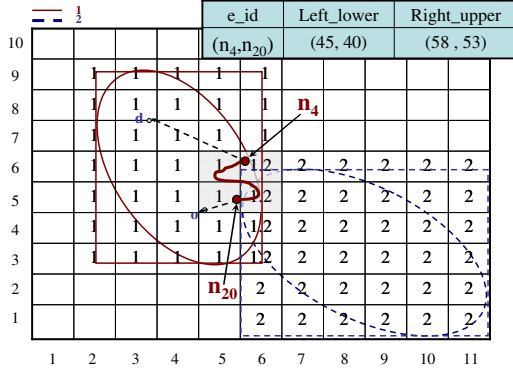


Fig. 7. Grid-based index

### 4.2 Index Maintenance

In the dynamic environment, the origin of a query and the travel time of a path often vary as time goes, and so do the affecting area of a query and its MBR. The grid index will incur considerable costs due to the large amount of updates for these varying MBRs, especially for the re-computations of current MBRs for all queries. Therefore, we further design a method for efficient index update, in which the MBRs are not continuously updated but the accuracy of query answers is guaranteed. Recall that there is no need to reevaluate a query if its answer previously computed is tolerable. Similarly, it is not necessary to update the MBR if the previously computed answer is guaranteed to be tolerable at this moment.

For illustration, consider the two affecting areas in Figure 8,  $a_1$  and  $a_2$ , for the same query at two time slots  $t_1$  and  $t_2$ , respectively. Suppose that at  $t_1$  the origin is  $o_1$  and the destination is fixed at  $d$ . The corresponding MBR, which can be computed from Property 3, is denoted as  $r_1$ . After a while, at  $t_2$  the origin moves to  $o_2$  and the affecting area becomes  $a_2$ . If the query is reevaluated, the corresponding MBR will be updated as  $r_2$ . If we do not update the MBR, any path passing the region  $a_2-r_1$  will not be considered as the query answer even though it is indeed the fastest path inside  $a_2$ . We name the path (from  $o_2$  to  $d$ ) passing the region  $a_2-r_1$  the *potentially missed path*.

According to the tolerance parameter, whenever the origin of a query is updated, our method quickly estimates a lower bound on the travel time of all the potentially missed paths, denoted as  $LB$ . Then, the MBR is not updated if the lower bound is higher than the travel time of the previously computed answer minus the tolerance parameter, i.e.,  $LB > T_{t_2}(P_{t_1}) - \Delta T$ . The lower bound  $LB$  is derived as below. In Figure 8, we find that a potentially missed path must visit a point under  $S_3$  and therefore its length must exceed  $MD(o_2, d, S_{bottom})$ . In fact, while the object movement is unpredictable, the potentially missed paths can be located in any region outside the

four sides of the MBR, including  $S_{left}$ ,  $S_{top}$ , and  $S_{right}$ . As a result, the length of a potentially missed path must be larger than  $\min\{MD(o_2, d, S_{left}), MD(o_2, d, S_{top}), MD(o_2, d, S_{right}), MD(o_2, d, S_{bottom})\}$ , which is denoted as  $MD_{min}$ . It follows that the travel time of a potentially missed path is lower bounded by  $MD_{min}/V$ , where  $V$  is the highest speed of object movements. We set  $LB$  as  $MD_{min}/V$  for the reason that no potentially missed path can make the answer expire if all the potentially missed paths have travel times higher than  $T_{t_2}(P_{t_1}) - \Delta T$ . Let  $MAXD$  denote  $V*(T_{t_2}(P_{t_1}) - \Delta T)$ , i.e., the maximal distance an object can move in  $T_{t_2}(P_{t_1}) - \Delta T$ . Based on the current affecting area, the criteria for deciding whether the MBR of a query should be updated are stated in the following.

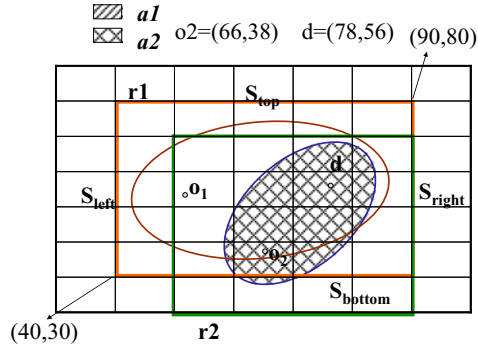


Fig. 8. An update of the affecting area

**Lemma 1.** Let the bottom-left and the top-right coordinates of the MBR recorded in the last time slot be  $(a_1, b_1)$  and  $(a_2, b_2)$ . Given the current origin  $o_2=(x_{o_2}, y_{o_2})$  and the destination  $d=(x_d, y_d)$ , if the following four conditions are all met, we do not need to update the MBR with the accuracy of the answer guaranteed.

$$(i) a_1 \leq \frac{X_+ - \sqrt{MAXD^2 - Y_-^2}}{2} \quad (ii) b_1 \leq \frac{Y_+ - \sqrt{MAXD^2 - X_-^2}}{2}$$

$$(iii) a_2 \geq \frac{X_+ + \sqrt{MAXD^2 - Y_-^2}}{2} \quad (iv) b_2 \geq \frac{Y_+ + \sqrt{MAXD^2 - X_-^2}}{2}$$

where  $X_+ = x_{o_2} + x_d$ ,  $X_- = |x_{o_2} - x_d|$ ,  $Y_+ = y_{o_2} + y_d$ , and  $Y_- = |y_{o_2} - y_d|$ .

**Proof:** As described, we do not update the MBR if  $LB = MD_{min}/V > T_{t_2}(P_{t_1}) - \Delta T$ . It implies that for every side  $S = S_{left}, S_{top}, S_{right}, S_{bottom}$ ,  $MD(o_2, d, S) \geq MD_{min} > V*(T_{t_2}(P_{t_1}) - \Delta T) = MAXD$ . Solving the inequality with one side will get one of the four conditions. Without loss of generality, we use  $S_{bottom}$ :  $y=b_1$  to show the derivation. From property 2(b), we have that  $MD(o_2, d, S_{bottom}) = \sqrt{X_-^2 + (Y_+ - 2*b_1)^2} \geq MAXD$ . In this way, the condition (ii) can be obtained. Similarly, the other three conditions can also be derived from  $a_1$ ,  $a_2$ , and  $b_2$ , respectively. ■

**Example 2.** According to the coordinates in Figure 8, we have  $X_+=144$ ,  $Y_+=96$ ,  $X_-=12$ , and  $Y_- = 18$ . If  $MAXD$  is currently equal to 35, we examine the criteria in Lemma 1 as follows and conclude that no update is needed:

$$(i) \frac{144 - \sqrt{35^2 - 18^2}}{2} = 55.99 \geq a1 = 40 \quad (ii) \frac{96 - \sqrt{35^2 - 12^2}}{2} = 30.56 \geq b1 = 30$$

$$(iii) \frac{144 + \sqrt{35^2 - 18^2}}{2} = 87 \leq a2 = 90 \quad (iv) \frac{96 + \sqrt{35^2 - 12^2}}{2} = 64.44 \leq b2 = 80$$

If it is decided not to update the affecting area and the corresponding MBR, we further shrink it by removing the query identifier from the cells that have been outside the current affecting area. For example, let the ellipse E in Figure 9 be the current affecting area, where D is the length of its major axis and m is the mean of its origin o and destination d. Since E is exactly contained by circle C with m as its center and D/2 as the radius, the cells covered by E must also be covered by C. Obviously, the cells that are outside C must also be outside E. Therefore, the cells marked in gray must be outside the current affect area and the query identifier corresponding to E stored in them can then be removed.

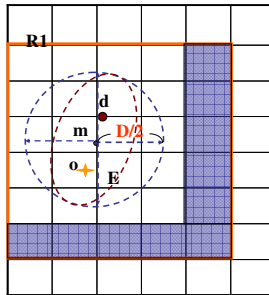


Fig. 9. Update by shrinking the MBR

Finally, we summarize the procedure of EBM. The initial answer of a query is computed as soon as it is posed. At the beginning of every time slot, update is first performed, followed by the check and reevaluation of queries in PQR. Notice that in the check stage, we apply Lemma 1 to decide whether to update the MBRs. For the query whose MBR is not updated, we shrink its MBR by deleting the query id from parts of its cells to further reduce the cost of index maintenance.

## 5 Experiment

### 5.1 Experiment Settings

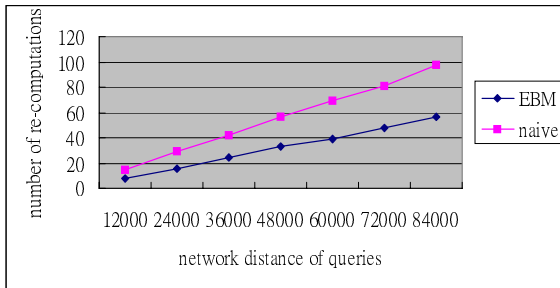
The road network of Nova Scotia, Canada, extracted from the GML file in 2005 Road Network File project [19], is used in our experiments. There are 77,084 edges and 58,379 vertices in the road network. The method based on the cellular automata [3] is adopted to simulate the traffic flow on road segments at the beginning of each time slot. Initially, five million objects are generated with velocities and positions in the network randomly assigned. Subsequently, at the beginning of every time slot, each object is assigned an event such as acceleration and deceleration by considering its velocity and the number of objects in front of it. The travel time of each edge is computed per minute. For each edge, its current travel time will be reported if the

difference on the travel time of the current answer and last one is larger than 60 seconds. More details of the traffic simulation can be referenced in [3].

In our implementation, the width and the height of a cell in the grid are both set to 1 km and each time period is equal to two minutes. Remember that EBM aims on saving unnecessary re-computations to accelerate the processing of fastest path queries. We design experiments in which our EBM method is compared with a naïve method, which re-computes the fastest path at the beginning of every time slot, by respectively observing the number of re-computations for single query processing and the processing time for multiple queries in a time slot. In addition, EBM restricts the search space within the affecting area so that the number of nodes processed is reduced. Since a large road network must be stored in the secondary storage, the number of nodes processed is related to the number of disk I/O, which directly influences the processing time of a fastest path query. Therefore, we also compare EBM with the Dijkstra's algorithm on the number of nodes processed for the one-shot query. All the experiments are performed on Intel Pentium 4 CPU 2.8 GHz with 1GB RAM.

## 5.2 Experiment Results

In the first experiment, we generate queries with network distances that range from 12 km to 84 km in multiples of 12 km. For each network distance, one thousand queries are generated. We generate different sets of queries according to the network distance because the numbers of re-computations correspond to the naïve method of the queries with the same network distance are similar. The averages on the numbers of re-computations for two methods are shown in Figure 10. It can be seen that our method saves 43% re-computations on average.



**Fig. 10.** Comparison on the numbers of re-computations for single query processing

Secondly, we consider the processing time needed for multiple queries in a time slot. In this experiment, we generate query sets by considering the number of nodes to be processed for evaluating a query by the Dijkstra's algorithm (*NumOfNodes*), and aims on equalizing the processing time of the queries in the same query set. Five sets of one thousand queries, the *NumOfNodes* ranges from 2,000 to 10,000 in multiples of 2,000, are generated. As Figure 11 shows, our method outperforms the naïve method since unnecessary re-computations are avoided and the search space is reduced in EBM. As it can be seen from Figure 11, EBM saves 34% processing time on average.

Finally, the numbers of nodes processed for the one-shot query by our method and Dijkstra's algorithm are compared in Figure 12. For this experiment, we generate five

sets of queries by the same manner as the second experiment. The vertical axis of Figure 12 indicates the ratio of nodes saved, compared to the Dijkstra’s algorithm, while the ellipse bounding property is applied to reduce the number of re-computations. The result shows that only 24% of the nodes processed by Dijkstra’s have to be processed by EBM.

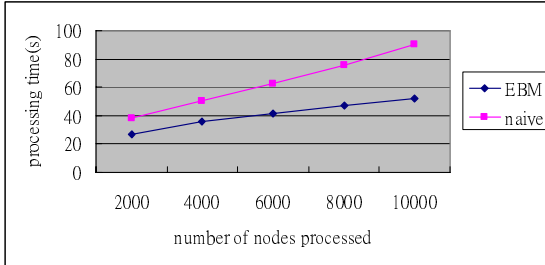


Fig. 11. Comparison on the processing time for multiple queries in a time slot

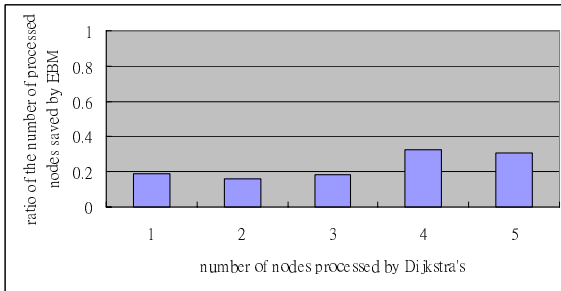


Fig. 12. Comparisons on the number of nodes processed for the one-shot query

## 6 Conclusion and Future Works

The problem of continuously processing the fastest path queries on road networks is addressed in this paper. Differing from the previous works, our work considers the environment in which the traffic varies with time and cannot be predicted. Our method, EBM, reduces both the number of re-computations and the search space for reevaluating a query and substantially speeds up the query processing in such a dynamic environment. We design a grid-based index structure to reflect the affecting areas of multiple queries currently running in the system. Moreover, we propose novel techniques for efficient index maintenance with user-specified precision guaranteed. We perform experiments using the data from a real road network and simulate the traffic based on the cellular automata. Experiment results confirm that our method saves on average 43% re-computations during the evaluation of a single query. While continuously monitoring multiple queries, at each time instant, on average 34% processing time for all the running queries can be reduced.

In the future, how to divide the queries into groups in a way that the scalability of our approach can be improved by processing the queries in the same groups in a batch will be studied. In addition, the other kinds of continuous queries, e.g., kNN queries and range queries, are also important to various location-aware applications on road networks. It will be interesting and challenging to adapt the techniques proposed in this paper to these kinds of queries.

## Reference

- [1] Agrawal, R., Jagadish, H.: Materialization and Incremental Update of Path Information. In: Proc. Fifth Int'l Conf. Data Eng. pp. 374-383 (1989)
- [2] Cormen, T., Lieserson, C., Rivest, R.: Introduction to algorithms (1990)
- [3] Esser, J., Schrechenberg, M.: Microscopic Simulation of Urban Traffic Based on Cellular Automata. *International Journal of Modern Physics* 8, 1025–1036 (1997)
- [4] Dijkstra, E.: A Note on Two Problems in Connection with Graphs. *Numerische Mathematik* 1, 269–271 (1959)
- [5] Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Lifelong planning A\*. *Artificial Intelligence* 155, 93–146 (2004)
- [6] Fu, L., Rilett, L.: Expected shortest paths in dynamic and stochastic traffic networks. *Transportation Research, Methodological* 32, 499–516 (1998)
- [7] Gupta, S., Kopparty, S., Ravishankar, C.: Roads, Codes, and Spatiotemporal Queries. In: Proc. of the 19th ACM Symp. on Principles of Database Systems (PODS), pp. 13–18. ACM Press, New York (2004)
- [8] Huang, Y., Jing, N., Rundensteiner, E.: A Semi-Materialized View Approach for Route Maintenance in Intelligent Vehicle Highway Systems. In: Proc. Second ACM Workshop Geographic Information Systems, pp. 144–151. ACM Press, New York (1994)
- [9] Huang, Y., Jing, N., Rundensteiner, E.: Hierarchical Path Views: A Model Based on Fragmentation and Transportation Road Types. In: Proc. Third ACM Workshop Geographic Information Systems, pp. 93–100. ACM Press, New York (1995)
- [10] Jing, N., Huang, Y., Rundensteiner, E.: Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation. *IEEE Transactions on Knowledge and Data Engineering* 10, 409–432 (1998)
- [11] Ku, W., Zimmermann, R., Wang, H., Wan, C.: Adaptive Nearest Neighbor Queries in Travel Time Networks. In: Proc. of the 13th annual ACM international workshop on Geographic information systems, ACM Press, New York (2005)
- [12] Kanoulas, E., Du, Y., Xia, T., Zhang, D.: Finding Fastest Paths on A Road Network with Speed Patterns. In: 22nd International Conference on Data Engineering (2006)
- [13] Nilsson, N.: Problem-Solving Methods. *Artificial Intelligence* (1971)
- [14] Pallottino, S., Scutella, M.: Shortest Path Algorithms in Transportation Models: Classical and Innovative Aspects. *Equilibrium and Advanced Transportation Modeling* (1998)
- [15] Russell, S., Norvig, P.: *Artificial Intelligence: A Modern Approach* (2003)
- [16] Shekhar, S., Fetterer, A., Goyal, B.: Materialization Trade-Offs in Hierarchical Shortest Path Algorithms. In: Proc. Of 4th Advances in Spatial and Temporal Databases (1997)
- [17] Shahabl, C., Kolahdouzan, M., Sharifzadeh, M.: A Road Network Embedding Technique for K-nearest Neighbor Search in Moving Object Databases. *GeoInformatica* 7, 255–273 (2003)
- [18] Sankaranarayanan, J., Alborzi, H., Samet, H.: Efficient Query Processing on Spatial Networks. In: Proc. of the 13th annual ACM international workshop on Geographic information systems, ACM Press, New York (2005)
- [19] [http://geodepot.statcan.ca/Diss/2006Dissemination/Data/FRR\\_RNF\\_e.cfm](http://geodepot.statcan.ca/Diss/2006Dissemination/Data/FRR_RNF_e.cfm)

# Continuous Medoid Queries over Moving Objects

Stavros Papadopoulos<sup>1</sup>, Dimitris Sacharidis<sup>2</sup>, and Kyriakos Mouratidis<sup>3</sup>

<sup>1</sup> Department of Computer Science and Engineering  
Hong Kong University of Science and Technology  
Clear Water Bay, Hong Kong  
stavros@cse.ust.hk

<sup>2</sup> School of Electrical and Computer Engineering  
National Technical University of Athens Greece, 15780  
dsachar@dblabb.ntua.gr

<sup>3</sup> School of Information Systems, Singapore Management University  
Singapore, 178902  
kyriakos@smu.edu.sg

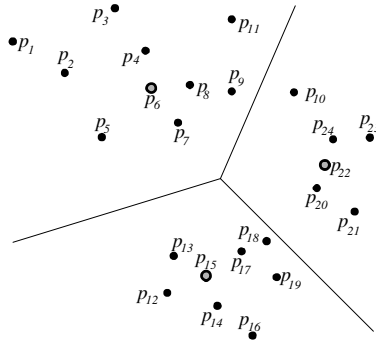
**Abstract.** In the  $k$ -medoid problem, given a dataset  $P$ , we are asked to choose  $k$  points in  $P$  as the *medoids*. The optimal medoid set minimizes the average Euclidean distance between the points in  $P$  and their closest medoid. Finding the optimal  $k$  medoids is NP hard, and existing algorithms aim at approximate answers, i.e., they compute medoids that achieve a small, yet not minimal, average distance. Similarly in this paper, we also aim at approximate solutions. We consider, however, the continuous version of the problem, where the points in  $P$  move and our task is to maintain the medoid set on-the-fly (trying to keep the average distance small). To the best of our knowledge, this work constitutes the first attempt on continuous medoid queries. First, we consider *centralized* monitoring, where the points issue location updates whenever they move. A server processes the stream of generated updates and constantly reports the current medoid set. Next, we address *distributed* monitoring, where we assume that the data points have some computational capabilities, and they take over part of the monitoring task. In particular, the server installs adaptive filters (i.e., permissible spatial ranges, called *safe regions*) to the points, which report their location only when they move outside their filters. The distributed techniques reduce the frequency of location updates (and, thus, the network overhead and the server load), at the cost of a slightly higher average distance, compared to the centralized methods. Both our centralized and distributed methods do not make any assumption about the data moving patterns (e.g., velocity vectors, trajectories, etc) and can be applied to an arbitrary number of medoids  $k$ . We demonstrate the efficiency and efficacy of our techniques through extensive experiments.

**Keywords:** Medoid Queries, Continuous Query Processing, Moving Object Databases.

## 1 Introduction

Given a dataset  $P$  and a user-specified parameter  $k$ , a  $k$ -medoid query returns a subset of  $P$  consisting of  $k$  points. These points are called the *medoids* and are selected so

that the average distance between the points in  $P$  and their closest medoid is minimized. The  $k$ -medoid problem arises in many fields and application domains, including resource allocation, data mining, spatial decision making, etc. Consider the example in Figure 1, where  $P = \{p_1, \dots, p_{24}\}$  is the set of residential blocks in a city, and fire stations are to be opened at three of them. To achieve the shortest average response time to emergency calls, we should minimize the average distance between residential blocks and their closest station. In this case, the best blocks to open fire stations at are the  $k = 3$  medoids of  $P$ . In our example, the medoids are blocks  $p_6$ ,  $p_{15}$  and  $p_{22}$ , shown in grey. The lines in the figure signify the assignment of the residential blocks to their responsible (i.e., closest) fire station. Due to this implicit assignment,  $k$ -medoids have also been used in different contexts for partitioning clustering.



**Fig. 1.** A 3-medoid example

Computing an optimal medoid set is NP hard [GJ79], and only approximate answers are possible even for relatively small input datasets. To this end, existing methods range from theoretical approximation schemes (e.g., [ARR98]), to hill-climbing approaches for moderate size datasets (e.g., [KR90, NH94]), to heuristic-based algorithms for disk-resident data (e.g., [EKX95a, EKX95b, MPP]). All previous methods assume a static  $P$ , i.e., they compute the  $k$  medoids once and then terminate. In this paper, we address a dynamic version of the problem, where the points in  $P$  send frequent location updates and the medoid set needs to be continuously maintained. In accordance with most real-world scenarios, the points in  $P$  move arbitrarily, with unknown motion patterns. We term the problem *continuous medoid monitoring*.

As a medoid monitoring example, consider a number of users accessing a location based service through their mobile devices, e.g., cellular phones or PDAs. To reduce the communication cost (and, thus, energy consumption), a number  $k$  of supernodes are selected among the mobile devices; the supernodes collect, aggregate and forward to the location server messages received from their vicinity. Due to signal attenuation for long distances, the devices should be close to some supernode. In other words, the supernode selection essentially reduces to a  $k$ -medoid computation over the set of devices. Additionally, the mobile nature of the system requires on-the-fly medoid



maintenance. All the devices (supernodes or not) move frequently and arbitrarily, necessitating supernode re-assignment in order to retain the quality of service.

We consider two system models, corresponding to different mobile environments. First, we address *centralized* medoid monitoring. In this setting, the data objects<sup>1</sup> in  $P$  send updates to a central server whenever they move. The server processes the location updates and computes/reports the new medoid set. We propose two incremental monitoring algorithms that aim at minimizing the processing time for medoid maintenance. In the centralized model, the objects issue frequent location updates. This raises the additional concern about the communication cost. In particular, in many mobile computing applications, the objects have scarce power resources and we wish to preserve battery life by limiting the number of messages transmitted to the server. This motivates our second, *distributed* processing model. In this context, the server assigns *safe regions* to the data objects, which issue location updates only if they move outside their region. We design effective safe region computation strategies and incorporate them to our medoid monitoring framework. We demonstrate that the distributed methods drastically reduce the object communication overhead, while sacrificing minimal medoid quality (i.e., they result in marginally higher average distance compared to their centralized counterparts).

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 describes our two centralized methods, while Section 4 presents their distributed versions. Section 5 experimentally evaluates the performance of our algorithms. Finally, Section 6 concludes the paper.

## 2 Related Work

In this section, we survey previous work on medoid queries (in Section 2.1), focusing on solutions targeted at large datasets. We also review spatial query monitoring techniques (in Section 2.2), since we assume a similar system architecture and use related geometric techniques and indexes.

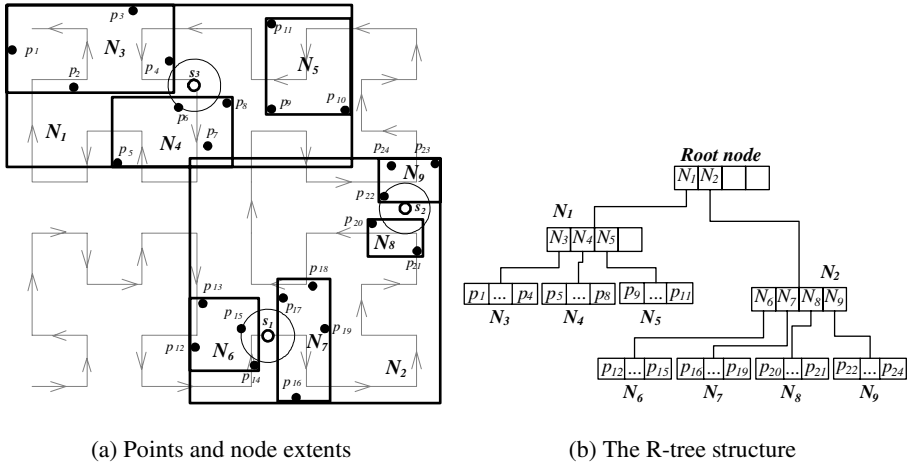
### 2.1 Medoid Queries

Finding the  $k$ -medoids is a classic problem in Computational Geometry, where it is usually referred to as the  $k$ -medians problem. Since it is NP hard, several approximation schemes have been proposed for its solution (e.g., [ARR98]). These schemes are of theoretical nature, aiming at graceful asymptotic bounds. More practical solutions include hill-climbing algorithms, such as PAM and CLARA [KR90]. Starting with a randomly chosen  $k$ -medoid set, these methods consider swapping one medoid with another, randomly chosen data point. If the swap leads to a lower average distance, then the resulting medoid set becomes the new candidate answer. This procedure is repeated for a fixed number of possible swaps. It terminates when no considered swap achieves a lower distance than the current medoid set, and returns the latter as the solution. To achieve better scalability than PAM and CLARA, Ng and Han [NH94] propose CLARANS. It builds upon CLARA, examining

---

<sup>1</sup> Henceforth, the terms point and object are used interchangeably.

however a smaller set of possible swaps, and, thus, speeding up the execution (i.e., converging faster to a local minimum). CLARANS is still slow for large problem instances (being restricted to inputs of just a few thousand objects), and it is impractical for disk-resident data. Motivated by this fact, Ester et al. [EKX95a, EKX95b] design FOR. In FOR, dataset  $P$  is indexed with an R-tree [G84, BKSS90], and a sample is formed by drawing one data point from each leaf of the R-tree. FOR executes CLARANS on this sample and returns the computed medoids. Focused also on disk-resident data, Mouratidis et al. [MPP] propose TPAQ, a method that solves  $k$ -medoid and related problems. TPAQ assumes that  $P$  is indexed with an R-tree and exploits its grouping properties to avoid reading the entire dataset, while achieving a low average distance. To exemplify, consider dataset  $P = \{p_1, \dots, p_{24}\}$  in Figure 2a and its R-tree in Figure 2b.



(a) Points and node extents

(b) The R-tree structure

**Fig. 2.** Example R-tree and TPAQ execution for 3-medoid computation

Assume that TPAQ is posed with a 3-medoid query. It descends the R-tree from the root down to the topmost level that contains more than (or equal to)  $k$  entries. This level is called the *partitioning level*, and let  $E$  denote the set of its entries. In Figure 2, the partitioning level is the second one, and its entries are  $E = \{N_3, \dots, N_9\}$ . The entries in  $E$  are sorted according to their center's Hilbert value, and the resulting sorted list is divided into  $k$  groups  $S_i$  of equal cardinality (i.e.,  $|E|/k$  entries each). The sorted list in our example (as given by the Hilbert curve shown in Figure 2a) is  $N_6, N_7, N_8, N_9, N_5, N_4, N_3$ , and the 3 groups are  $S_1 = \{N_6, N_7\}$ ,  $S_2 = \{N_8, N_9\}$ , and  $S_3 = \{N_5, N_4, N_3\}$ . For each  $S_i$ , TPAQ computes the geometric centroid<sup>2</sup>  $s_i$  and performs a nearest neighbor (NN) search at  $s_i$  among the underlying data points (i.e., among the points corresponding to the sub-trees of entries in  $S_i$ ). In Figure 2a, the centroids of the three groups are points  $s_1$ ,  $s_2$ , and  $s_3$  (appearing hollow). The three medoids returned are

<sup>2</sup> The geometric centroid of group  $S_i$  is point  $s_i$  with coordinates  $s_{i,x}$  and  $s_{i,y}$  equal to the average  $x$ - and  $y$ - coordinates, respectively, of the entry centers in  $S_i$ .

their NNs, i.e., points  $p_{15}$ ,  $p_{22}$ , and  $p_6$ . TPAQ is shown to achieve lower distance than FOR and exhibits better scalability.

The existing  $k$ -medoid algorithms are unsuitable for our continuous monitoring setting. All aforementioned methods are designed for static datasets and snapshot queries (i.e., they compute the medoids once and then terminate); their extension to incremental medoid maintenance (in the presence of updates) is non-trivial, if possible at all. On the other hand, the naïve approach of re-computing from scratch the medoids (with some existing algorithm) in each update processing cycle is prohibitively expensive in a highly dynamic scenario, failing to reuse previous results. Additional problems of existing methods are: (i) the hill-climbing approaches (PAM, CLARA, CLARANS, etc.) are very slow for moderate or large input sizes, while (ii) TPAQ and FOR are designed for disk-resident data, with primary objective the minimization of the I/O cost; disk accesses are not an issue in our main memory setting, where CPU time (and communication cost, in the distributed case) is the only concern. On the other hand, an important finding of previous work to our problem is the efficiency and, more so, the efficacy of TPAQ, which motivates us to use a similar Hilbert-based (or, in general, space filling curve-based) approach for our purposes.

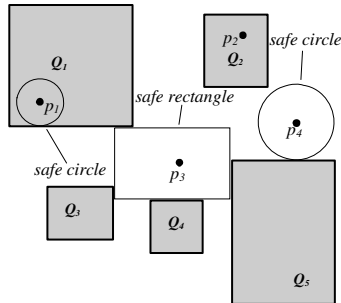
Regarding medoid-related problems in dynamic settings, Guha et al. [GMM+03] solve the  $k$ -medoid problem in a streaming environment. In the assumed model, the points of the input dataset  $P$  stream into the system. The main memory is not enough to store entire  $P$ , so the streamed data points are processed once and then discarded as new ones arrive. When the entire input set is seen, the system reports its  $k$ -medoids. [GMM+03] proposes an one-pass  $k$ -medoid algorithm that solves the above problem, using a small amount of space. Even though this is a dynamic method, it does not apply to our setting; in our case, (i) the memory does fit the *entire* dataset, but the points therein receive *location updates* in an on-line fashion, and (ii) the system needs to continuously report the  $k$ -medoid set *at any time*.

A problem related to  $k$ -medoids is *min-dist optimal-location* (MDOL) computation. The input consists of a set of data points  $P$ , a set of existing facilities (i.e., a set of existing medoids) and a user-specified spatial region  $R$ , wherein a new facility should open. The output of an MDOL query is the location in  $R$  where the new facility should be built in order to minimize the overall average distance between the data points and their closest facility. Zhang et al. [ZDXT06] propose an exact method for this problem. The main differences from the  $k$ -medoid problem is that (i) MDOL assumes that a set of facilities already exists, (ii) it computes a single point (as opposed to  $k$ ), and (iii) the returned point does not necessarily belong to  $P$ , but it can be anywhere inside region  $R$ .

The  $k$ -medoid problem is related to clustering; essentially, given the medoids, the input dataset can be partitioned into  $k$  clusters by assigning each point to its closest medoid. The other direction, however, does not work; although there are numerous clustering methods for large input sets (e.g., DBSCAN [EKSX96], BIRCH [ZRL96], CURE [GRS98] and OPTICS [ABKS99]), their objective is to create clusters such that the points in any cluster are more similar to each other than to points in other clusters. In addition to addressing a problem of different nature, most clustering algorithms are computationally intensive and unsuitable for the highly dynamic environments we tackle in this work.

## 2.2 Continuous Spatial Queries

The first spatial monitoring techniques were targeted at range queries, where the data objects send location updates to a central server, and the latter continuously reports the objects that fall in each monitored range. *Q-index* [PXX+02] processes static range queries. It indexes the ranges using an R-tree and probes moving objects against the index in order to determine the affected queries and update their results. SINA [MXA04] monitors (potentially moving) range queries using a three-step spatial join between moving objects and ranges. *Mobieyes* [GL04] and MQM [CHC04] follow a distributed processing approach, where the objects utilize their computational capabilities and suppress some location updates. In particular, all of *Q-index*, *Mobieyes* and MQM utilize the concept of *safe regions*, according to which each object  $p$  is assigned a circular or rectangular region, such that  $p$  needs to issue an update only if it exits this area (because, otherwise, it does not influence the result of any query). Figure 3 shows a range monitoring example, where the current result of query  $Q_1$  is object  $p_1$ , of  $Q_2$  is object  $p_2$ , while no object qualifies queries  $Q_3$ ,  $Q_4$ ,  $Q_5$ . The safe regions for  $p_1$  and  $p_4$  are circular, while for  $p_2$  and  $p_3$  they are rectangular, as shown in the figure (the safe rectangle for  $p_2$  coincides with the boundary of  $Q_2$ ). Note that even if the objects move, unless they fall outside their assigned safe regions, no query result can change.



**Fig. 3.** The safe regions concept

In addition to range queries, several methods have been recently proposed for  $k$  Nearest Neighbor ( $k$ -NN) monitoring. Koudas et al. [KOTZ04] present a system for approximate  $k$ -NN queries over streams of multidimensional points. Yu et al. [YPK05], Xiong et al. [XMA05] and Mouratidis et al. [MHP05] describe algorithms for exact  $k$ -NN queries; all three methods index the data with a regular grid and maintain the  $k$ -NN results by considering only object movements that may influence some query. The aforementioned techniques aim at low processing time. There exist, however, methods designed for network cost minimization [MPBT05, HXL05] by exploitation of the objects' computational resources; their rationale is similar to that of the *safe regions* explained in Figure 3.

### 3 Centralized Medoid Monitoring

In this section we present our centralized methods. We assume that dataset  $P$  consists of  $|P|$  two-dimensional points. Although our methods are applicable to higher dimensions, in accordance with most real-world mobile environments, we focus on two dimensions. Furthermore, for ease of presentation, we consider a unit dataspace, i.e., all data fall in  $[0,1]^2$ . Every point  $p$  in  $P$  is a tuple of the form  $\langle p.id, p.x, p.y \rangle$ , where  $p.id$  is a unique identifier and  $(p.x, p.y)$  are  $p$ 's coordinates. Whenever  $p$  moves, it issues an update to the monitoring server; the update has the form  $\langle p.id, p.x_{old}, p.y_{old}, p.x_{new}, p.y_{new} \rangle^3$ , implying that  $p$  moves from  $(p.x_{old}, p.y_{old})$  to  $(p.x_{new}, p.y_{new})$ . The objects move frequently and arbitrarily.

We present two centralized medoid monitoring algorithms, based on a common intuition exemplified in Figure 4. Dataset  $P$  contains two clusters  $C_1$  and  $C_2$ . Suppose that a 2-medoid query returns one medoid in  $C_1$  and another in  $C_2$ . Now consider that we wish to compute three medoids. Observe that, although  $C_1$  has a smaller *diameter* than  $C_2$ , it contains more points. Due to the larger cardinality of  $C_1$ , the distances of its points from its medoid affect the global average distance to a greater extent than that of the points in  $C_2$ . Therefore, placing the third medoid in  $C_1$  leads to a larger distance reduction than placing it in  $C_2$ . Intuitively, more medoids must be assigned to denser areas of the dataspace.

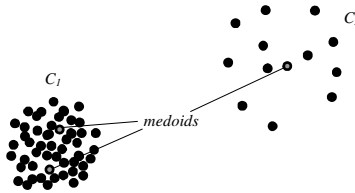


Fig. 4. The three medoids of a dataset consisting of two clusters

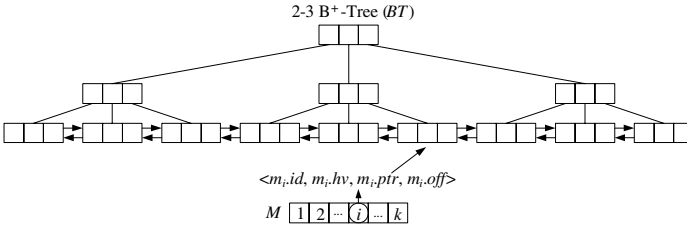
Motivated by this observation, our algorithms (i) partition the points in  $P$  into  $k$  groups of (roughly) equal cardinality and, then, (ii) select the most centrally located object from each group as the corresponding medoid. To quickly perform step (i) we project the points on a one-dimensional space using a space filling curve. We employ the Hilbert curve since it is shown to best preserve locality compared to alternatives [MJFS01]. Next, we partition the Hilbert-sorted list of points into  $k$  groups of equal cardinality (i.e.,  $|P|/k$ ). Due to the locality preservation of the Hilbert curve, the resulting groups can be regarded as well-defined partitions of  $P$  in the two-dimensional space. Finally, we extract a medoid from each group; the medoid is the point in the group with the median Hilbert value, as it is expected to be the most centrally located. The above rationale underlies both modules of our algorithms, namely, the initial medoid computation and their maintenance.

<sup>3</sup> If the update is an insertion (deletion),  $p.x_{old}, p.y_{old}$  ( $p.x_{new}, p.y_{new}$ ) are set to a negative value.

### 3.1 The HBM Algorithm

Our first method is *Hilbert-based Monitoring (HBM)*. It indexes the data objects with an in-memory 2-3 B<sup>+</sup>-Tree [C79] (i.e., a B<sup>+</sup>-Tree where each internal node has two or three children), using their Hilbert values as search keys. We denote this tree by *BT*. At the leaf level, except for the standard *right sibling* pointers, *BT* is modified to also accommodate *left sibling* pointers. In other words, the leaves are organized as a doubly connected linked list. When the continuous medoid query is installed at the server for the first time and *BT* is built, every entry *E* in an internal node *N* temporarily stores aggregate information about the number of points *E.a* contained in its subtree. *E.a* facilitates the initial medoid computation and is discarded afterwards.

In particular, according to our general approach, the *i*-th medoid of *P* is the  $[(i-0.5) \cdot |P|/k]$ -th object in the linear order imposed by the Hilbert values. HBM locates the *k* medoids by performing *k* traversals in *BT*, at a total cost of  $O(k \cdot \log |P|)$ . Before each traversal *i*, an auxiliary variable *V* is initialized to zero. The traversal starts from root  $N_R$  and it checks whether  $V + E_1.a$  is larger than or equal to  $(i-0.5) \cdot |P|/k$ , where  $E_1$  is  $N_R$ 's first entry. If that is the case, the medoid is located in  $E_1$ 's subtree and, therefore, the traversal continues by visiting  $E_1$ 's child. Otherwise,  $E_1.a$  is added to *V* and the algorithm continues similarly by checking  $V + E_2.a$  against  $(i-0.5) \cdot |P|/k$  ( $E_2$  is  $N_R$ 's second entry). *V* always keeps the number of points preceding (in the Hilbert order) the point with the smallest search key that is reachable by the traversal. Finally, the algorithm reaches the leaf node containing the *i*-th medoid. For every computed medoid *m*, an array *M* of size *k* stores a tuple of the form  $\langle m.id, m.hv, m.ptr, m.off \rangle$ , where *m.id* is the identifier of the point selected as *m*, *m.hv* is *m*'s Hilbert value, *m.ptr* points to the leaf node of *BT* that accommodates *m*, and *m.off* is an integer (initialized to zero) used by the maintenance module and whose functionality is explained later. The temporary *E.a* values are discarded after the end of the initial computation step. Figure 5 summarizes the data structures in HBM.



**Fig. 5.** The data structures of the HBM method

The server periodically receives updates from the objects in batches. HBM accordingly updates *BT*, after computing the necessary Hilbert values of the inserted, deleted or moving points. Note that the movement of an object involves its deletion from the index followed by its subsequent re-insertion with the new Hilbert value. Whenever a split or merge operation moves a medoid *m* to a different leaf node, the corresponding *m.ptr* must also be altered in *M*. While updates are reflected in *BT*, HBM stores some book-keeping information, to be used for result maintenance

according to its medoid selection strategy. In particular, after processing the insertion/deletion of a point  $p$ , HBM performs a binary search in array  $M$  to locate the leftmost medoid  $m_u$  with Hilbert value greater than (or equal to)  $p.hv$ . In case  $p$  initiated an insertion (deletion), the algorithm increases (decreases)  $m_u.off$  by one. Particular care must be taken when a medoid  $m$  is deleted. In this case, HBM substitutes it with its predecessor in the Hilbert order and decreases  $m.off$  by one.

After processing all updates, HBM computes the new medoids as follows. The  $i$ -th medoid  $m_i$  was formerly data point  $p_{old}$  at position  $(i-0.5) \cdot |P|/k$ . After the updates,  $p_{old}$  moves to position  $(i-0.5) \cdot |P|/k + \sum_{j=1}^{i-1} m_j.off$ . The actual medoid must be located at position  $(i-0.5) \cdot |P'|/k$ , where  $P'$  is the updated version of dataset  $P$  (which may have different cardinality if new objects were inserted or existing ones deleted). Therefore, the new medoid  $m_i$  can be found  $OFF_i = (i-0.5) \cdot |P'|/k - (i-0.5) \cdot |P|/k - \sum_{j=1}^{i-1} m_j.off$  positions to the right or left of  $p_{old}$  in the linear order, depending on whether  $OFF_i$  is positive or negative, respectively. For every medoid  $m_i$  in  $M$ , HBM first visits the leaf node pointed by  $m_i.ptr$  to find its old corresponding point  $p_{old}$ . Then, using the left/right sibling pointers of  $BT$ , it locates the new medoid and properly updates  $m_i$ 's entry in  $M$ . The pseudocode of the maintenance procedure is given in Figure 6.

---

Function **updateMedoids** (*array M, Tree T*)

1. Initialize  $V$  to 0
  2. For  $i=1$  to  $k$
  3.     Locate medoid  $m_i$  in leaf  $M[i].ptr$  of  $T$
  4.      $OFF_i = (i-0.5) \cdot |P'|/k - (i-0.5) \cdot |P|/k - V$
  5.     If  $OFF_i = 0$ , continue
  6.     Else if  $OFF_i > 0$ , find point  $p$  located  $|OFF_i|$  positions to the right of  $m_i$
  7.     Else if  $OFF_i < 0$ , find point  $p$  located  $|OFF_i|$  positions to the left of  $m_i$
  8.      $V += M[i].off$ ;
  9.     Assign  $p.id$ ,  $p.hv$ , the pointer of the leaf of  $T$  that accommodates  $p$  and 0 to  $M[i].id$ ,  $M[i].hv$ ,  $M[i].ptr$  and  $M[i].off$ , respectively
- 

**Fig. 6.** The maintenance module of HBM

Figure 7 illustrates the initial computation and maintenance of  $k = 2$  medoids in a set of points, which at timestamp  $T_1$  has cardinality 14. For ease of demonstration, we omit the  $BT$  operations and focus on the leaf level of the tree, which constitutes a doubly connected linked list of points sorted on their Hilbert values. At timestamp  $T_1$ , the set is subdivided into two subsets of seven points each. The medians of the subsets ( $p_4$  and  $p_{12}$ ) are selected as the medoids ( $m_1$  and  $m_2$ , respectively). At timestamp  $T_2$ , four updates occur;  $p_1$  and  $p_{13}$  are deleted, and  $p_3$  and  $p_5$  move to new positions. Due to  $p_1$ 's deletion,  $m_1.off$  is decreased by one. On the contrary, the deletion of  $p_{13}$  does not affect any  $off$  value because there is no medoid with higher (or equal) Hilbert value. Regarding  $p_3$  and  $p_5$ , recall that a point movement is handled as a deletion followed by an insertion. Upon  $p_3$ 's deletion, the algorithm decreases  $m_1.off$ . Subsequently, the point is re-inserted in a position between  $m_1$  and  $m_2$  and, therefore,  $m_2.off$  is increased by one. Finally,  $p_5$ 's movement causes  $m_2.off$  to decrease (due to its deletion) and immediately increase (due to its re-insertion) by one, because both its old and new Hilbert values are between  $m_1.hv$  and  $m_2.hv$ . Let  $old\_pos_i$  be the position

(in the Hilbert order) of the point that was selected as medoid  $m_i$  at timestamp  $T_1$ . Also let  $curr\_pos_i$  be the position of the new point to become  $m_i$  at timestamp  $T_2$ . For  $m_1$ ,  $old\_pos_1 = 4$ ,  $curr\_pos_1 = 3$ , and  $OFF_1 = 1$ . Similarly for  $m_2$ ,  $old\_pos_2 = 11$ ,  $curr\_pos_2 = 9$ , and  $OFF_2 = -1$ . The algorithm locates the new medoids  $p_3$  and  $p_{11}$ , by moving one position to the right and one to the left from old medoids  $p_4$  and  $p_{12}$ , respectively.

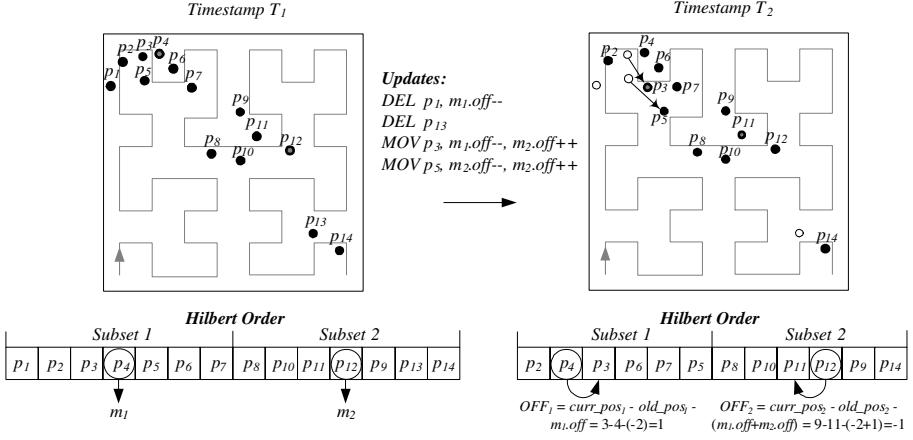


Fig. 7. A medoid monitoring example in HBM

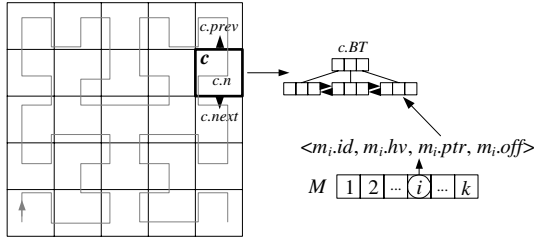
### 3.2 The GBM Algorithm

The *Grid-based Monitoring (GBM)* algorithm utilizes a  $C \times C$  regular grid for indexing  $P$ . Let  $\delta$  be the side-length of each cell. A point  $p$  in  $P$  with coordinates  $(p.x, p.y)$  can be located in constant time in cell  $c_{i,j}$  (i.e., the cell in column  $i$  and row  $j$ , starting from the low-left corner of the grid), where  $i = \lfloor p.x/\delta \rfloor$  and  $j = \lfloor p.y/\delta \rfloor$ . GBM imposes a linear order on the cells by sorting them according to the Hilbert values of their centers. Every cell  $c$  is associated with a tuple  $\langle c.n, c.prev, c.next, c.BT \rangle$ , where  $c.n$  is the cardinality of the set of points contained in  $c$ ,  $c.prev$  and  $c.next$  are the cells preceding and succeeding  $c$  in the Hilbert order respectively, and  $c.BT$  is a *BT* that indexes the points in  $c$  (using their Hilbert values as search keys). Similarly to HBM, the internal nodes in the *BTs* temporarily incorporate aggregate information, which is discarded after the initial computation of the medoids.

The grouping strategy of GBM is similar to HBM, the difference being in the linear order of the points, which now takes into account firstly the order of the cells. Specifically, the points are considered sorted according to the following rules; (i) a point  $p_1$  in cell  $c_1$  precedes point  $p_2$  in cell  $c_2$ , if  $c_1$  precedes  $c_2$  in their Hilbert order, and (ii) the order of the points in the same cell is determined by their Hilbert values. Following similar reasoning as in HBM, the  $i$ -th medoid  $m_i$  is the  $\lfloor (i-0.5) \cdot |P|/k \rfloor$ -th object in the above order. GBM starts by initializing an auxiliary variable  $V$  to zero and scans the linked list of the (sorted) cells. To locate medoid  $m_i$ , in every visited cell  $c_i$ , it checks whether  $V+c_i.n$  is larger than or equal to  $(i-0.5) \cdot |P|/k$ . If that is the case, it



traverses  $c_i.BT$  in order to find the  $[V+c_i.n-(i-0.5)\cdot|P|/k]$ -th object in the cell, which is then selected as medoid  $m_i$ . Otherwise, it adds  $c_i.n$  to  $V$  and continues to the next cell.  $V$  keeps the number of points encountered by the scan so far. Note that GBM locates all medoids in a single linear scan of the cells, i.e., after finding medoid  $m_i$ , it does not restart the scan for finding  $m_{i+1}$ ; instead, it continues from the cell that contains  $m_i$ . Finally, it maintains an array  $M$  with functionality identical to that used by HBM. Figure 8 depicts the data structures of GBM.



**Fig. 8.** The data structures of the GBM method

For every received update, GBM first determines in constant time the cell  $c$  where the insertion/deletion takes place, and properly updates  $c.BT$ . Subsequently, it scans  $M$  and updates the *off* value of the leftmost medoid with Hilbert value larger than or equal to that of the object that initiated the update, in a similar fashion to HBM. After processing all the updates, the maintenance module of GBM identifies the points to be selected as the new medoids as follows. It scans  $M$  and for every  $m_i$ , it computes  $OFF_i$  in a fashion similar to Section 3.1. Suppose that  $m_i$  lies in cell  $c$ . Then, starting from the leaf of  $c.BT$  that accommodates  $m_i$  and is pointed by  $m_i.ptr$ , it searches for the point that will be selected as the new  $m_i$ . This point lies  $OFF_i$  positions to the left or right of old  $m_i$ , depending on whether  $OFF_i$  is negative or positive, respectively. If the search reaches the leftmost or rightmost (in the Hilbert order) point of cell  $c$ , it continues to the cell pointed by  $c.prev$  or  $c.next$ , respectively. Note that the algorithm may skip entire cells (i.e., it may not traverse their  $BT$ s at all), since it can always determine whether  $m_i$  is located in a visited cell by comparing the cell's cardinality against  $OFF_i$ . After finding a new medoid, GBM updates the respective entry in  $M$  accordingly.

In Figure 9 we exemplify the initial medoid computation and monitoring in a scenario where  $k = 2$  and  $P$  contains points  $p_1$  to  $p_{14}$ . Consider cells  $c_{2,2}$  and  $c_{1,2}$  at timestamp  $T_1$ . The Hilbert curve first passes through  $c_{2,2}$  and, thus,  $p_{11}$  precedes  $p_1$  in the GBM order, although it succeeds it in the global Hilbert order (i.e.,  $p_{11}.hv > p_1.hv$ , where  $p_{11}.hv$  and  $p_1.hv$  are the Hilbert values of  $p_{11}$  and  $p_1$ , respectively). At timestamp  $T_1$ , the medoids are  $m_1 = p_4$  and  $m_2 = p_{14}$ , since they are at positions  $0.5 \cdot |P|/k = 4$  and  $1.5 \cdot |P|/k = 11$ , respectively, in the linear order. At timestamp  $T_2$ , objects  $p_7$ ,  $p_6$  and  $p_{11}$  issue updates, as shown in the figure. Their movement leads to  $OFF_1 = 1$  and  $OFF_2 = 1$ , and updates the medoids to  $m_1 = p_7$  and  $m_2 = p_{11}$ .

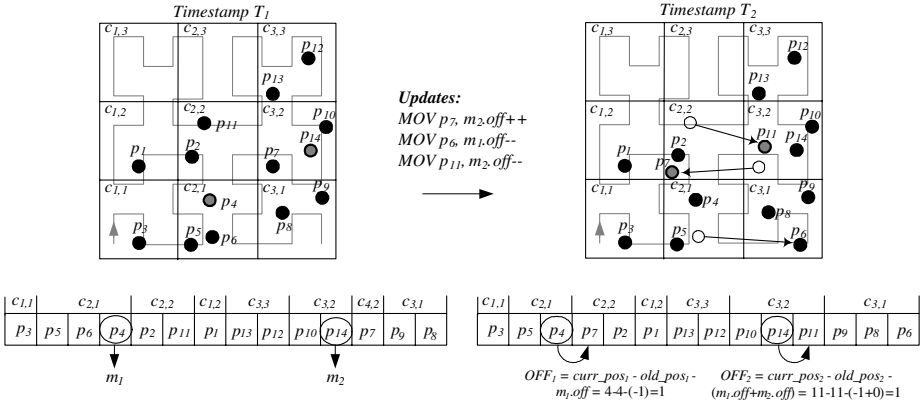


Fig. 9. A medoid monitoring example in GBM

Compared to HBM, index update and medoid maintenance in GBM are expected to be faster. HBM keeps a common  $BT$  over all  $|P|$  points, which leads to an  $O(\log|P|)$  cost for every point insertion or deletion. On the other hand, letting  $c$  be the cell of the inserted/deleted point,  $c.BT$  contains  $c.n$  objects (where  $c.n \ll |P|$ ), requiring  $O(\log c.n)$  time per update. Furthermore, maintaining the medoids is also more efficient in GBM, because for large  $OFF_i$  values, entire cell contents may be skipped when sliding in the linear point order towards the new medoid position. Another major advantage of GBM over HBM, is the fact that its data index is compatible with existing methods for other spatial query types; most range and nearest neighbor monitoring algorithms use a regular grid index<sup>4</sup>. This allows GBM to be used in conjunction with other methods, in a system that answers general spatial queries over moving objects, utilizing a single data index.

A final remark concerns the average distance, which is in general different but similar for GBM and HBM, since their medoid selection rationale is alike. In particular, if the grid granularity in HBM is selected so that  $C$  is a power of two (recall that the grid has  $C \times C$  cells), their medoids are identical. The reason is that the Hilbert values themselves are computed by definition based on a transparent space partitioning with a grid, whose granularity on each axis is always a power of two (this power is called the *order* of the Hilbert curve). If  $C$  is also a power of two, the cells of the object grid contain continuous, non-overlapping intervals of the curve. In other words, if cell  $c_1$  precedes  $c_2$  on the curve, then *any* point  $p_1$  in  $c_1$  precedes *every*  $p_2$  in  $c_2$ . In turn, this fact implies that the linear point orders of GBM and HBM are identical and, thus, the medoids are the same.

## 4 Distributed Medoid Monitoring

The main idea in the distributed version of our methods is to allow objects to move within assigned *safe regions*, without having to transmit updates to the server. Since

<sup>4</sup> All methods covered in Section 2.2 use a regular grid, except for [MPBT05] and [HXL05], where processing time minimization is not the main objective.

our general medoid selection strategy relies on a linear point order, the safe regions are defined with respect to the neighboring objects (in the order). Particularly, let *leeway*  $\lambda$  be an integer system parameter. The safe region of the  $i$ -th object in the order  $p_i$  is a Hilbert interval  $SR_i^\lambda = [p_i.sr_L, p_i.sr_R]$ . The left boundary  $p_i.sr_L$  is the mean of the Hilbert values of  $p_i$  and its  $\lambda$ -th left neighbor  $p_{i-\lambda}$  (i.e.,  $p_i.sr_L = \lfloor (p_i.hv + p_{i-\lambda}.hv)/2 \rfloor$ ). The right boundary  $p_i.sr_R$  is set similarly with respect to the  $\lambda$ -th right neighbor (i.e.,  $p_i.sr_R = \lceil (p_i.hv + p_{i+\lambda}.hv)/2 \rceil$ ). Object  $p_i$  may change location without issuing an update, as long as  $p_i.hv \in SR_i^\lambda$ . When  $p_i$  does move outside  $SR_i^\lambda$ , it sends its new location to the server. The latter updates its index and the medoid set accordingly<sup>5</sup>, and assigns a new safe region to  $p_i$ . Note that the new  $SR_i^\lambda$  is defined based on the latest point positions reported. Particularly for GBM, the linear point order takes into account the grid cells ordering. Thus, the safe regions are defined within each cell individually (i.e., in the Hilbert order of the objects therein). Whenever an object exits its cell, it sends an update regardless of whether it violates its safe region.

Figure 10 demonstrates the safe region function in the case of HBM (the case of GBM is similar, subject to the aforementioned modifications), showing the position of the points on the Hilbert curve. At timestamp  $T_1$ , the safe region  $SR_3^{\lambda=1}$  ( $SR_3^{\lambda=2}$ ) of  $p_3$  is defined according to  $p_2$  and  $p_4$  ( $p_1$  and  $p_5$ ) for  $\lambda = 1$  ( $\lambda = 2$ ). Similarly,  $SR_4^{\lambda=1}$  is determined by  $p_3$  and  $p_5$ . Assuming that  $\lambda = 1$ , at timestamp  $T_2$ , points  $p_3$  and  $p_4$  move. However, only  $p_3$  issues an update, because  $p_4$  remains within its safe region. The solid points in the figure correspond to the positions known by the server, the hollow point is  $p_3$ 's old Hilbert value, while the grey is  $p_4$ 's actual one. Object  $p_3$  is assigned a new region, based on the Hilbert values of  $p_2$  and  $p_4$ . Note that the server is not aware of the new location of  $p_4$  and, thus, uses the last reported one (as of  $T_1$ ).

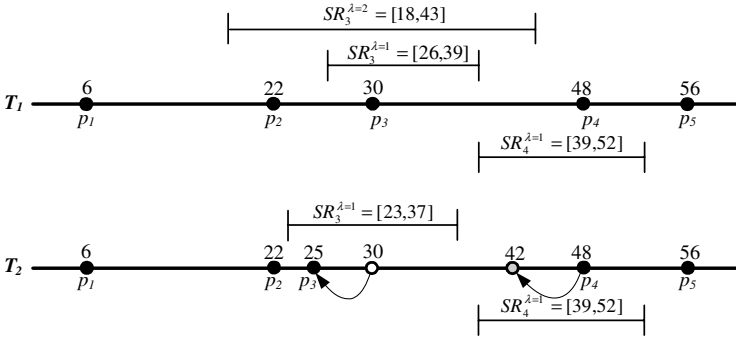


Fig. 10. Safe regions and update handling

## 5 Experimental Evaluation

In this section we evaluate the performance of our methods, in terms of processing time (at the server), number of object updates (i.e., communication cost for the

<sup>5</sup> Medoid maintenance at the server side is identical to the centralized case.

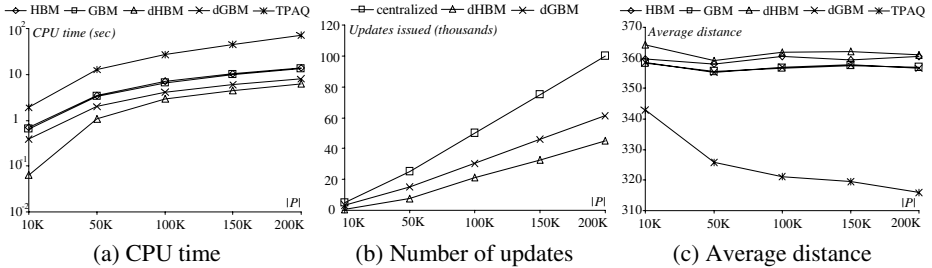
objects) and achieved average distance. We generate datasets of cardinality  $|P|$  ranging between 10K and 200K objects as follows. For each tested  $|P|$ , we randomly select the initial position and the destination of each object among the points of a real spatial dataset (North America, available at [www.maproom.psu.edu/dcw](http://www.maproom.psu.edu/dcw)). The object follows a linear trajectory between the two points. Upon reaching the endpoint, a new random destination is selected and the process is repeated. At every timestamp, a percentage  $a$  of the objects move towards their endpoint (while the remaining ones remain static), covering a distance  $v$ . We refer to  $a$  and  $v$  as the object *agility* and *velocity*, respectively. The velocity is expressed as a percentage of the dataspace extent on the  $x$  axis (we have a  $[0,10^4] \times [0,10^4]$  dataspace). The simulation length is 100 timestamps for each setting, and the reported measurements are the average observed values over all timestamps. We process continuous  $k$ -medoid queries for  $k$  between 2 and 512. We evaluate our four methods HBM, GBM, dHBM, and dGBM (where the latter two are the distributed versions of HBM and GBM). Also, we use as a competitor the TPAQ method with a main memory R-tree, since none of the other existing algorithms works for the large cardinalities tested, even for snapshot queries. To adapt TPAQ to medoid monitoring, we rerun it for the timestamps where (i) some of the medoids move, or (ii) the object updates affect the extents of the R-tree entries at the partitioning level. In each experiment we vary one parameter, while setting the remaining to their default values. The parameter ranges and defaults are shown in Table 1. For GBM and dGBM we fine-tuned the grid granularity (with respect to the average distance) for the default settings and use the best one (100×100) in all our experiments. We use a machine with a 3.2 GHz Pentium IV CPU and 1 GB RAM.

**Table 1.** Parameter ranges and default values

Parameter	Default	Range
Dataset cardinality $ P $	100K	10, 50, 100, 150, 200 (K)
No. of medoids $k$	32	2, 8, 32, 128, 512
Agility $a$	50%	10, 30, 50, 70, 100 (%)
Velocity $v$	0.5%	0.1, 0.3, 0.5, 0.7, 1 (%)
Leeway $\lambda$	300	100, 200, 300, 400, 500

In Figure 11, we measure the effect of object cardinality  $|P|$ , varying it from 10K to 200K objects and setting the other parameters to their defaults. Figure 11a shows the CPU cost (in logarithmic scale) for medoid maintenance per timestamp, i.e., the time to update the object index and the medoids. We observe that the centralized methods have similar cost (with GBM being slightly faster). The distributed algorithms have shorter running time, because they process fewer updates; dHBM (dGBM) takes less than 45% (60%) of the time of its centralized counterpart. dHBM is faster than dGBM, because the latter’s safe regions are practically smaller, as they are bounded by the grid cell boundaries (leading to more reported updates and, thus, higher processing cost). Compared to our methods, TPAQ is slower by an order of magnitude, mainly due to the excessive update cost of its R-tree index. An important remark about Figure 11a (and all remaining CPU time charts) is that we focus on

pure maintenance cost, i.e., we exclude the initial  $k$ -medoid computation. For the sake of completeness, the first-time medoid extraction for the default setting takes 12.9, 12.4 and 54.4 sec for HBM, GBM and TPAQ, respectively (the times for dHBM and dGBM are identical to HBM and GBM).



**Fig. 11.** Performance versus dataset cardinality  $|P|$

Figure 11b shows the number of updates sent to/processed by the server in the same experimental setup. All centralized methods (i.e., HBM, GBM, TPAQ) have the same communication cost, with the objects reporting their positions whenever they move. On the other hand, the safe regions of dHBM and dGBM save around 55% and 40% of these updates, respectively. dGBM avoids less updates than dHBM, due to the necessary updates required when the objects move to another cell, as explained in the context of Figure 11a. Figure 11c illustrates the achieved distance for the various cardinalities, expressed in distance units in our  $[0, 10^4] \times [0, 10^4]$  dataspace. We observe that the distributed methods compute only slightly worse medoid sets, verifying their efficacy. Note that both versions of GBM are better than those of HBM. The reason is that HBM is solely based on the one-dimensional Hilbert mapping, while GBM preserves a stronger connection to the original (two-dimensional) space, due to its spatial grid index. For a similar reason, TPAQ achieves 4 to 11% smaller distance than our methods, exploiting the graceful grouping properties of its R-tree. However, this benefit comes at a prohibitive update cost, leading to an excessive processing time (see Figure 11a). Another remark for TPAQ is that it improves with  $|P|$ ; for a denser space, the nearest neighbor queries (in its final step) retrieve medoids that lie closer to the “ideal” geometric centroids of the  $k$  groups, leading to a lower distance.

In Figure 12 we use the default settings and vary  $k$  between 2 and 512. Figure 12a shows the CPU time. Again dGBM is the fastest, for the reasons explained above. We observe that the processing cost is almost constant for each method and unaffected by  $k$ . The reason is that, in all methods (and especially in TPAQ), the monitoring cost is dominated by the number of processed updates (mainly due to index maintenance), which is irrelevant to  $k$ . Furthermore, in our algorithms, for larger  $k$ , there are more medoids to maintain, but the offsets (to slide in the linear point order) are smaller. On the other hand, the average distance drops with  $k$  for all methods, and our techniques’ difference from TPAQ decreases.

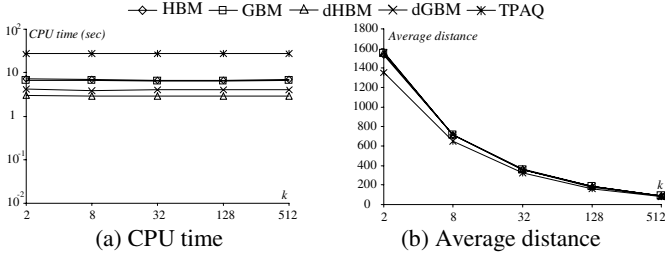


Fig. 12. Performance versus number of medoids  $k$

In Figure 13 we examine the effect of object agility  $a$ , with 10% up to 100% of the data points moving at each timestamp. The CPU cost (Figure 13a) increases with  $a$  due to the larger number of updates processed. Figure 13b shows the number of issued updates, which, as expected, is linear to  $a$ . In terms of average distance (Figure 13c), there is not much fluctuation; the small differences are due to the randomness of the dataset generation.

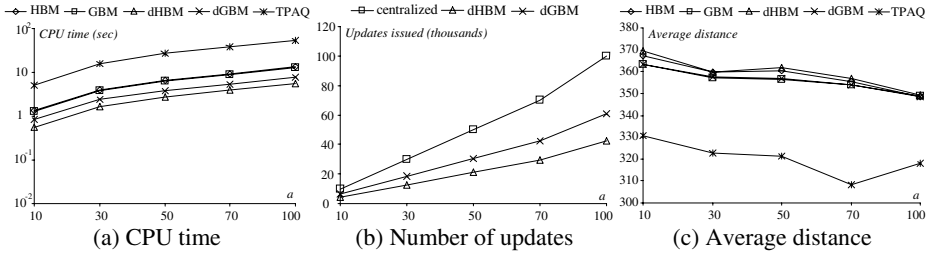


Fig. 13. Performance versus object agility  $a$

In Figure 14 we vary the object velocity  $v$  from 0.1 to 1% of the dataspace extent on the  $x$  dimension. Figure 14a shows the CPU time. The centralized methods are unaffected by  $v$ . On the other hand, the cost of the decentralized increases as more objects move outside their safe regions for larger  $v$ , sending more updates to the server for processing. This is also evident in Figure 14b. Interestingly, for  $v = 0.1\%$ , dGBM incurs less object updates than dHBM (because its cells are large with respect to  $v$ , without practically limiting the safe regions), while for  $v = 1\%$  their number is almost as high as for the centralized methods. The average distance (Figure 14c) is similar for all values of  $v$ .

Figure 15 investigates the effect of the leeway  $\lambda$ , varying it from 100 to 500. The performance of the centralized methods is identical, because they do not use safe regions. As shown in Figure 15b, for  $\lambda = 500$ , dHBM achieves 65% reduction of the location updates. For dGBM, however, there is a marginal decrease, because the safe regions are restricted by the grid cells, rather than by  $\lambda$ . The number of updates has a direct impact on the CPU time and, thus, the trends in Figure 15a are similar as in Figure 15b. In terms of average distance,  $\lambda$  affects only dHBM, whose performance

deteriorates for larger  $\lambda$ . This trend verifies the tradeoff between update cost and medoid quality. On the other hand, dGBM is not affected because the server processes a similar set of updates.

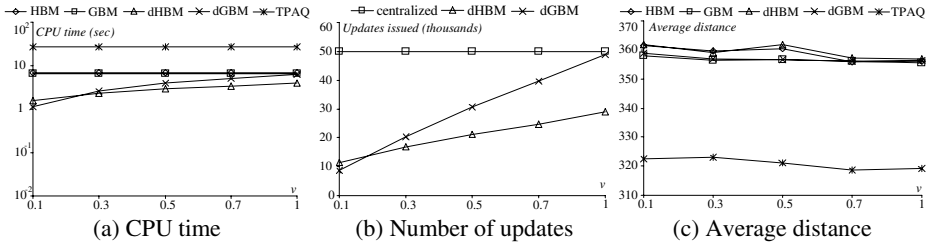


Fig. 14. Performance versus object velocity  $v$

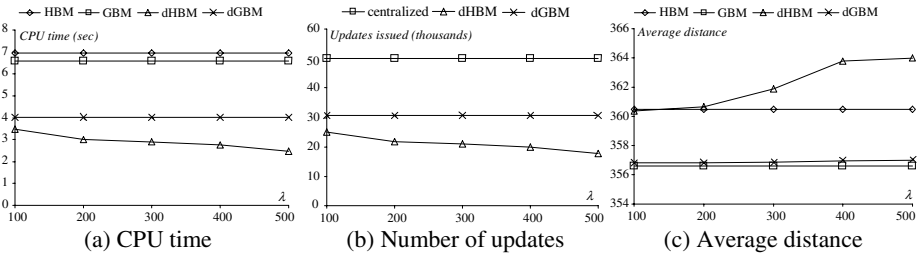


Fig. 15. Performance versus leeway  $\lambda$

## 6 Conclusion

In this paper we address the problem of  $k$ -medoid monitoring. To the best of our knowledge this is the first work on this topic. We consider a central server that continuously receives the locations of frequently moving objects and incrementally maintains their medoid set. Without making any assumption about the data moving patterns, our methods achieve low running times while keeping the medoid quality high. Furthermore, we consider distributed environments, where the data objects have limited power resources and attempt to preserve them by reducing the number of updates they transmit to the server. In this context, the server assigns safe regions to the objects, which report their position only when they exit their region. We evaluate our methods through extensive experiments and investigate tradeoffs between communication cost and medoid quality.

## Acknowledgements

This work was supported by grant HKUST 6184/06E from Hong Kong RGC, and by an award from the Lee Foundation.

## References

- [ARR98] Arora, S., Raghavan, P., Rao, S.: Polynomial Time Approximation Schemes for Euclidean  $k$ -Medians and Related Problems. In: STOC (1998)
- [ABKS99] Ankerst, M., Breunig, M., Kriegel, H.P., Sander, J.: OPTICS: Ordering Points To Identify the Clustering Structure. In: SIGMOD (1999)
- [BKSS90] Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The  $R^*$ -tree: An Efficient and Robust Access Method for Points and Rectangles. In: SIGMOD (1990)
- [C79] Comer, D.: The Ubiquitous B-Tree. ACM Computing Surveys 11(2), 121–137 (1979)
- [CHC04] Cai, Y., Hua, K., Cao, G.: Processing Range-Monitoring Queries on Heterogeneous Mobile Objects. In: MDM (2004)
- [EKX95a] Ester, M., Kriegel, H.P., Xu, X.: A Database Interface for Clustering in Large Spatial Databases. In: KDD (1995)
- [EKX95b] Ester, M., Kriegel, H.P., Xu, X.: Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification. In: SSD (1995)
- [EKXS96] Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A Density Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In: KDD (1996)
- [G84] R-Trees, A.: dynamic index structure for spatial searching. In: SIGMOD (1984)
- [GJ79] Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
- [GL04] Gedik, B., Liu, L.: MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, Springer, Heidelberg (2004)
- [GMM+03] Guha, S., Meyerson, A., Mishra, N., Motwani, R., O’Callaghan, L.: Clustering Data Streams: Theory and Practice. IEEE TKDE 15(3), 515–528 (2003)
- [GRS98] Guha, S., Rastogi, R., Shim, K.: CURE: An Efficient Clustering Algorithm for Large Databases. In: SIGMOD (1998)
- [HXL05] Hu, H., Xu, J., Lee, D.: A generic framework for monitoring continuous spatial queries over moving objects. In: SIGMOD (2005)
- [KOTZ04] Koudas, N., Ooi, B., Tan, K., Zhang, R.: Approximate NN queries on Streams with Guaranteed Error/performance Bounds. In: VLDB (2004)
- [KR90] Kaufman, L., Rousseeuw, P.: Finding Groups in Data. Wiley-Interscience, Chichester (1990)
- [MHP05] Mouratidis, K., Hadjieleftheriou, M., Papadias, D.: Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring. In: SIGMOD (2005)
- [MJFS01] Moon, B., Jagadish, H.V., Faloutsos, C., Saltz, J.H.: Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. IEEE TKDE 13(1), 124–141 (2001)
- [MPBT05] Mouratidis, K., Papadias, D., Bakiras, S., Tao, Y.: A Threshold-based Algorithm for Continuous Monitoring of  $k$  Nearest Neighbors. IEEE TKDE 17(11), 1451–1464 (2005)
- [MPP] Mouratidis, K., Papadias, D., Papadimitriou, S.: Tree-based Partition Querying: A Methodology for Computing Medoids in Large Spatial Datasets. In: VLDBJ (to appear)
- [MXA04] Mokbel, M., Xiong, X., Aref, W.: SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In: SIGMOD (2004)



- [NH94] Ng, R., Han, J.: Efficient and Effective Clustering Methods for Spatial Data Mining. In: VLDB (1994)
- [P XK+02] Prabhakar, S., Xia, Y., Kalashnikov, D., Aref, W., Hambrusch, S.: Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Transactions on Computers* 51(10), 1124–1140 (2002)
- [XMA05] Xiong, X., Mokbel, M., Aref, W.: SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. In: ICDE (2005)
- [Y PK05] Yu, X., Pu, K., Koudas, N.: Monitoring K-Nearest Neighbor Queries Over Moving Objects. In: ICDE (2005)
- [ZDXT06] Zhang, D., Du, Y., Xia, T., Tao, Y.: Progressive Computation of the Min-Dist Optimal-Location Query. In: VLDB (2006)
- [ZRL96] Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An Efficient Data Clustering Method for Very Large Databases. In: SIGMOD (1996)

# Efficient Index Support for View-Dependent Queries on CFD Data

Christoph Brochhaus and Thomas Seidl

Data Management and Exploration Group  
RWTH Aachen University, Germany  
{brochhaus, seidl}@informatik.rwth-aachen.de

**Abstract.** Recent years have revealed a growing importance of *Virtual Reality* (VR) visualization techniques which offer comfortable means to enable users to interactively explore 3D data sets. Particularly in the field of *computational fluid dynamics* (CFD), the rapidly increasing size of data sets with complex geometric and supplementary scalar information requires new out-of-core solutions for fast isosurface extraction and other CFD post-processing tasks. Whereas spatial access methods overcome the limitations of main memory size and support fast data selection, their VR support needs to be improved. Firstly, interactive users strongly depend on quick first views of the regions in their view direction and, secondly, they require quick relevant views even when they change their view point or view direction.

We develop novel view-dependent extensions for access methods which support static and dynamic scenarios. Our new human vision-oriented distance function defines an adjusted order of appearance for data objects in the visualization space and, thus, supports quick first views. By a novel incremental concept of view-dependent result streaming which interactively follows dynamic changes of users' viewpoints and view directions, we provide a high degree of interactivity and mobility in VR environments. Our integration into the new *index based graphics data server* "IndeGS" proves the efficiency of our techniques in the context of post-processing CFD data with dynamically interacting users.

## 1 Introduction

In recent years, numerical simulations in the area of fluid dynamics offer a very high level of accuracy and reproducibility of fluid behavior and replace tedious and expensive physical experiments which often strongly depend on environmental conditions. Both in industrial development and in research, simulations are acknowledged methods in application domains including physics, automotive engineering and analysis of aerodynamic forces. These methods, simulating the interaction of gases or fluids with complex surfaces, are generally described as *computational fluid dynamics* (CFD).

During a post-processing step certain features of the CFD data sets are extracted by request of interactive users which are often experts in the application domain. The results are commonly visualized in virtual reality environments, e.g. the *HoloBench*,

a stereo projection table composed of two right-angled projection surfaces, or two- to six-sided rear projection systems called *CAVE* (cf. figure 1), or even small mobile devices like PDAs. They offer a high degree of interactivity by letting users immerse into the visualized objects. Common post-processing tasks include isosurface extraction (“display regions with a temperature of exactly  $125^{\circ}\text{C}$ ”) amongst others.



a) CAVE

b) HoloBench

**Fig. 1.** Examples of VR hardware

With increasing CPU powers, computers are able to produce larger and more accurate simulation data sets, often up to many gigabytes in size. Efficient post-processing is one of the major requirements that VR frameworks have to satisfy, reducing expensive idle times until a result is presented and ready for visual inspection, thus not delaying the user’s flow of work. A significant increase in efficiency can be achieved by streaming parts of the solution, thus enabling the user to catch a first impression of the overall result set and react with change of view or post-processing parameters. The first impression can be improved by dynamically incorporating the user’s position and view direction into query processing: fractions of the solution set that are close to and in front of the users regarding view direction contribute more to the first impression than others. In this paper, we introduce a new distance function for a view-dependent ranking of the results which is aligned to the special characteristics of human perception. We present streaming techniques incorporating dynamically changing view parameters (users’ view points and directions), thus offering a qualitative first impression to moving users.

State-of-the-art frameworks mainly use standard PCs for post-processing, which allow for lower costs and a high degree of scalability. They work with the CFD data stored in main memory and quickly extract interesting features by completely scanning the data set. We present external memory techniques for efficiently indexing and accessing CFD data sets through post-processing, thus breaking main memory limitations and allowing for data sets of almost arbitrary size, as well as supporting fast query processing. We propose the **Index based Graphics data Server** “IndeGS”, using secondary storage methods to offer dynamic view-dependent access methods with the above mentioned benefits of offering quick first impressions, enabling the user to change view and post-processing parameters “on the fly” with immediate response from the system, thus increasing efficiency and speed of knowledge extraction. This server can be integrated into almost any network and a multitude of available visualization frameworks.

The paper is organized as follows: In section 2, we present related work from the field of spatial indexing and query processing. We introduce our new human vision oriented distance function in section 3. Efficiently querying CFD data by a static user is presented in section 4, with dynamic query adaptation aspects shown in section 5. Section 6 presents efficient secondary memory indexing methods for CFD. Section 7 presents the results of experiments on real world data sets.

## 2 Related Work

A great variety of secondary storage index methods for handling multidimensional data exists. Many of these methods hierarchically and spatially divide the data space into regions. One main characteristic and advantage of the structures is the property that objects with local proximity in the data space are stored with topological proximity in the index. Index structures like the R-Tree [1] and its variants (e.g.  $R^+$ -Tree [2],  $R^*$ -Tree [3]) use hierarchically nested minimum bounding rectangles (MBRs) for the inner directory nodes to manage the data points stored in the leaf nodes.

In [4], efficient methods for performing isosurface extraction on external memory interval trees is presented. These techniques do not support streaming and view-dependent ordering of the result data, and combinations of post-processing queries are not endorsed.

Nearest neighbor algorithms such as the Depth-first Traversal algorithm of [5] and the Priority Search algorithm described in [6] use these index structures to prune whole branches of the tree once the feature space region they represent can be guaranteed to not include the nearest neighbor. We focus on the second approach and modify it in order to achieve a complete and exact ranking of all result cells depending on the query point, which represents the user's standpoint in our application.

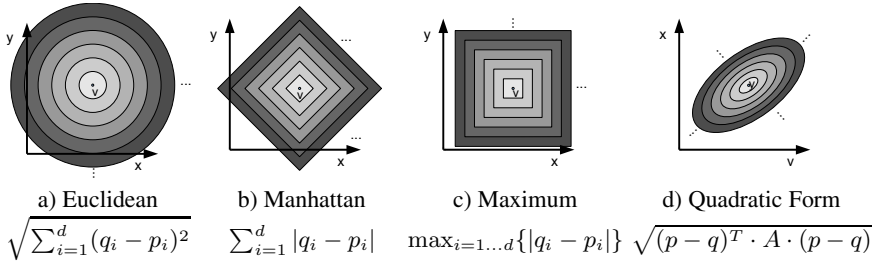
To be able to exchange the distance function in the context of these index structures and access methods, new pruning measures have to be introduced. These determine the minimum distance (*MINDIST*) of directory node elements to the query point and define the priority by which these node have to be visited. In the context of quadratic form distance functions, [7] introduces a *MINDIST* calculation by applying a gradient technique to find the location on the MBR which has least distance to the query point.

In the field of dynamic indexing and query processing, several methods for indexing moving objects have been proposed [8,9]. The mentioned papers introduce R-Tree based techniques for indexing and querying moving objects in dynamic data sets. Instead of indexing moving objects, the processing of continuous queries is discussed in [10]. Query results are updated over time via invalidating and joining previous results sets. In [11], nearest neighbor sets for query points at sampled positions in the data space are calculated by utilizing results of neighboring query points to reduce the overall number of expensive nearest neighbor queries. Due to the immense size of CFD data sets and the high complexity of post-processing queries, approaches using precalculation of queries and maintaining precalculated results render inefficient.

## 3 Human Vision Oriented Distance Functions

In order to decrease costly operation times of complex VR hardware and at the same time increase efficiency of expert users performing CFD post-processing, it is of utmost importance to reduce query times for isosurface extraction and to allow the user to get a quick first impression of the query result. Our presented access methods show the following benefits: (1) results close to the viewer are presented rapidly, (2) results in direct line of sight are ranked with higher priorities, (3) changing view position and view direction are reflected by dynamic query adaptation. Queries  $q$  are specified as follows:

$$q := (\text{view point } v, \text{ view direction } a, \text{ box } b, \text{ scalar range } sc_1, \dots, \text{ scalar range } sc_n).$$



**Fig. 2.** Traditional distance functions: isolines around a viewpoint  $v$

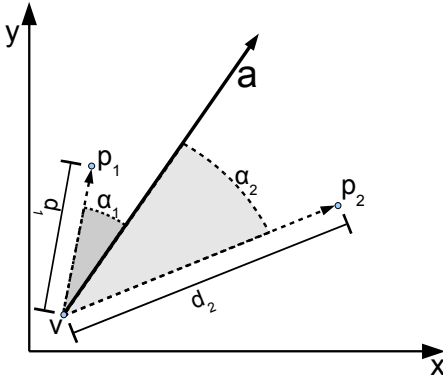
Here,  $v$  is the exact position of the viewer in or around the visualized data set, and by  $a$  we denote the view direction specified as a three-dimensional vector. Box  $b$  specifies the geometrical range of the query. Usually, this box covers the complete range of the three dimensions, as no results shall be omitted from the result set, except in the case of “geometrical selection” post-processing. Ranges of the scalar values  $sc_i$  are specified to meet the requirements of each query. In the case of isosurface extraction, the corresponding scalar range  $sc_i$  has to be set to the designated value. CFD data is usually provided by simulation software as a collection of numerous cells with geometrical and supplemental scalar information. Cells are considered *active*, if all their values intersect with the corresponding ranges specified in  $q$ .

### 3.1 Traditional Distance Functions

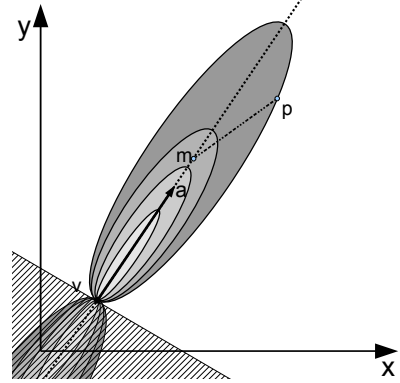
Standard ranking methods based on k-nearest neighbor queries like presented in [5] and [6] perform a recursive walk through the underlying spatial data structure by calculating distance approximations for MBRs in directory nodes based on distances like Euclidean, Manhattan or Maximum distance etc. Using these distance functions ranks the results in a spherical or octahedron-like manner, respectively, around the query view point  $v$  (cf. figure 2a-c). One side effect is, that results not in the user’s line of sight are visualized with the same priority as results directly in the line of sight. To improve the speed of result extraction we assign higher priorities to objects directly in or close to the line of sight and therefore letting the output “grow” faster in direction of the user’s view. This can be achieved by using quadratic form distances, which stretch the corresponding ellipsoids representing isoline/isosurfaces in the direction of the user’s view (cf. figure 2d). One obvious shortcoming with quadratic forms is the assignment of higher priorities to objects along the viewing axis independent of their orientation from the users standpoint (i.e. in front or in the back of the user).

### 3.2 View Direction Oriented Heuristics

In a first approach, we developed a distance function based on polar coordinates. As illustrated in figure 3,  $\alpha$  denotes the angle/deviation between the user’s line of sight  $a$  and the connecting line between the viewpoint  $v$  and object  $p_i$ . By  $d_i$  we denote the Euclidean distance between  $p_i$  and  $v$ . In the following, we present some straightforward heuristics and discuss their suitability for our purposes:



**Fig. 3.** Heuristics based on polar coordinates



**Fig. 4.** Human vision oriented  $hv$ -distance

1.  $d(v, p_i) := d_i + w \cdot \alpha$ : Finding an appropriate weight  $w$  is problematic. This weight strongly depends on the distribution of distances between  $v$  and objects  $p_i$  in the data set. If the closest active cell is relative far from  $v$ , the angle  $\alpha$  is probably dominated by all possible Euclidean distances  $d_i$ . On the other hand, if the user is standing “inside” the data set and the maximum distance to the farthest active cell is quite low,  $\alpha$ ’s influence can become inappropriately high.
2.  $d(v, p_i) := d_i \cdot \alpha$ : Objects directly on the line of sight show unfavorable behavior, as they all share the distance value of 0, independent of their Euclidean distance. Furthermore, objects far from  $v$ , but close to the axis of view  $a$ , are assigned lower priorities as objects with a lower Euclidean distance  $d_i$ , but the same distance from  $a$ . The decreasing  $\alpha$  overrules the increasing  $d_i$ .
3.  $d(v, p_i) := d_i \cdot e^\alpha$ : Objects directly on the line of sight are now assigned their Euclidean distances  $d_i$ , as  $e^0 = e^0 = 1$ . The effect of the overruling of  $d_i$  is compensated by applying the exponential function on  $\alpha$ . Still problematic is the separation of objects in front of the user from objects in the back.
4. lexicographical order  $(\alpha, d_i)$ : Ranking active cells by their lexicographical order  $(\alpha, d_i)$  leads to the effect that objects with small  $\alpha$  are reported first in the order of increasing Euclidean distances. Objects very close to the user, but with a high  $\alpha$  are presented unwantedly late. This inhibits a quick impression of the result objects in the direct vicinity of the user as objects with a large distance are ranked before objects with a very low distance and a marginally larger corresponding  $\alpha$ . A clear separation of objects in front of the user from the objects in the back is possible, as objects with  $\alpha \geq 90^\circ$  are only displayed after all objects with  $\alpha < 90^\circ$  have been visualized. This heuristic does not work well without an appropriate discretization of the range of  $\alpha$ .

### 3.3 Human Vision Oriented Distance Function

To combine the advantages of both, traditional distance functions and the heuristics presented above, we define a distance function which joins positive attributes of both

types and perfectly reflects the characteristics of human vision. We use the following symbols to explain our new distance function:

$v$	represents the view point/location of user
$A$	similarity matrix to be used in quadratic form calculations
$p$	object/point, for which distance has to be calculated
$m$	center of ellipsoid which traverses $v$ and $p$ and is located on axis of sight
$a$	user's line of sight (vector representation), normalized to a length of 1

Our distance, referred to as  $hv$ -distance (**h**uman **v**ision oriented distance,  $dist_{hv}$ ) caters for an accelerated growth of the result set in direction of the line of sight and at the same time a delayed growth in the area of human peripheral vision. The corresponding isolines are illustrated in figure 4; the distance of  $p$  is defined by the radius of the ellipsoid which crosses  $p$  and the user's standpoint  $v$  with center point  $m$ , which necessarily lies on the line of sight  $a$  starting at  $v$ . To avoid that the ellipsoid is covering space in the back of the user, we let the ellipsoid's center move (with increasing radius) along the vector  $a$ . Elements behind the user (located in the hatched region defined by the hyperplane running through  $v$  and perpendicular to  $a$ ) are treated separately, as we describe in the following paragraphs. The ellipsoids of both the standard quadratic forms and our distance function are stretched in direction of  $a$ . The ratio of growth in direction of the line of sight compared to growth to any orthogonal direction is defined by a parameter.

The formula  $d_A(q, p) := \sqrt{(p - q)^T \cdot A \cdot (p - q)}$  defines standard quadratic form distances, with  $A$  being a similarity matrix specifying cross-similarity weights for the dimensions. For our purpose to derive an appropriate matrix  $A$ , we start with the diagonalization  $A = R^T \cdot D \cdot R$  with a diagonal matrix  $D$  and orthonormal transformation  $R$ . We set the first column of  $R$  to the user's viewing direction  $a$ . The remaining columns  $r_2, \dots, r_n$  of  $R$  are vectors which are orthonormal to  $a$  and to each other. These can be calculated via the Gram-Schmidt process or the numerically more stable QR decomposition. The diagonal matrix  $D$  contains eigenvalues of the transformation matrix  $R$ . To acquire a "stretching" of the ellipsoid along the view axis, we increase the top left-most value in  $D$  to a value  $d_1^2 \gg 1$ . The effect of this is that vector  $a$  is mapped to itself, but its length is increased by factor  $d_1^2$ . All in all, matrix  $R$  is a basis transformation depending on the view direction and  $D$  caters for the stretching of the quadratic form ellipsoid in the desired direction. The mathematical notation is as follows:

$$A := R^T \cdot D \cdot R \text{ with } D = \begin{pmatrix} d_1^2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } R = \begin{pmatrix} a_1 & r_{2,1} & r_{3,1} \\ a_2 & r_{2,2} & r_{3,2} \\ a_3 & r_{2,3} & r_{3,3} \end{pmatrix}$$

Our  $hv$ -distance function differs from standard quadratic form distances on one important property: the centers of the ellipsoids are not equal to the query point, they instead move along the line of sight, depending on the object for which the distance is calculated. Figure 4 shows several ellipsoids and additionally the ellipsoid relevant for the distance calculation of  $dist_{hv}(v, p)$  with center  $m$ . The radius of the ellipsoid passing through  $v$  and  $p$  defines  $dist_{hv}$ :

$$dist_{hv}(v, p) := d(v, m)$$

The following equations represent our design concepts that the distance between  $v$  and  $m$  (2) is equal to the distance between  $p$  and  $m$  (5), i.e. the ellipsoid around  $m$  is passing through  $v$  and  $p$ . This distance is the radius of the corresponding ellipsoid. Furthermore,  $m$  is on the user's axis of sight, located  $\lambda$  times vector  $a$  from origin  $v$  (3).

$$d(p, m) = d(v, m) \quad (1)$$

Let us first prove that  $d(v, m)$  is proportional to the ellipsoids' stretching factor  $d_1$ , i.e.  $d(v, m) = \lambda \cdot d_1$ :

$$d(v, m) = \sqrt{(v - m)^T \cdot A \cdot (v - m)} \quad (2)$$

$$m = v + \lambda \cdot a \quad (3)$$

After insertion of (3) in (2):

$$\begin{aligned} d(v, m) &= \sqrt{(v - (v + \lambda \cdot a))^T \cdot A \cdot (v - (v + \lambda \cdot a))} \\ &\Rightarrow d(v, m) = \sqrt{\lambda \cdot a^T \cdot A \cdot \lambda \cdot a} \end{aligned}$$

With  $a^T \cdot A = d_1^2 \cdot a^T$ , because eigenvector  $a$  is mapped to itself with  $d_1^2$  times its length, and  $a^T \cdot a = 1$ , because  $a$  is normed,  $d(v, m)$  simplifies to:

$$d(v, m) = \lambda \cdot \sqrt{d_1^2} = \lambda \cdot d_1 \quad (4)$$

□

The distance  $d(p, m)$  is defined as:

$$d(p, m) = \sqrt{(p - m)^T \cdot A \cdot (p - m)} \quad (5)$$

Inserting (3) in (5):

$$\begin{aligned} d(p, m) &= \sqrt{(p - (v + \lambda \cdot a))^T \cdot A \cdot (p - (v + \lambda \cdot a))} = \\ &\sqrt{(p - v)^T \cdot A \cdot (p - v) - 2 \cdot \lambda \cdot a^T \cdot A \cdot (p - v) + \lambda^2 \cdot \underbrace{a^T \cdot A \cdot a}_{=d_1^2}} \end{aligned}$$

With (4):

$$\Leftrightarrow \lambda = \frac{(p - v)^T \cdot A \cdot (p - v)}{2 \cdot d_1^2 \cdot a^T \cdot (p - v)}$$

The overall formula for the  $hv$ -distance is defined as follows:

$$dist_{hv}(v, p) := d(v, m) = \lambda \cdot d_1 = \frac{(p - v)^T \cdot A \cdot (p - v)}{2 \cdot d_1 \cdot a^T \cdot (p - v)} \quad (6)$$

Let us note a useful side effect: objects in the back of the user all have negative distances. The ellipsoids in front of the user are mirrored along the separating hyperplane introduced above (cf. figure 4). Distances of objects on these ellipsoids and their mirrored counterparts only differ in their sign. Whereas this property hinders the  $hv$ -distance to be a metric, it can be utilized to start a ranking of the objects in the back of the user, once all objects in front have been completely visualized during post-processing.



## 4 Static Query Scenarios

In state-of-the-art VR frameworks, users start post-processing queries and results are visualized, once the query is completely processed. Thus, the efficiency of post-processing strongly depends on the speed of query execution, and with larger and larger data sets, query execution times increase. A continuous presentation of partial results based on streaming techniques enables the user to catch a first impression of the result set even before query completion. The quality of the impression is improved by presenting partial results which are close to the viewer and in the direction of view earlier than objects which contribute less to the impression. The quality of the first impression at any stage of query processing can be assessed by the fraction of the user’s vision field which is covered by the partial result set. We present view-dependent streaming techniques, which achieve a high coverage at early stages of post-processing. In VR frameworks, these result streams are usually received by the so called *visualizer*, a software component which caters for the presentation of the results to the user.

We base our external memory index structure on wide-spread tree-like data structures using minimum bounding rectangles (MBRs), as described in detail in section 6. After starting the query execution at the root node, child nodes (MBRs or data objects/cells) are assigned priorities based on a specific distance between their geometrical location and the user’s standpoint  $v$ . Objects qualified for the result set regarding their scalar values are then ranked according to their distance in e.g. a priority queue, handling the object with the lowest distance first: in the case of directory nodes, their child nodes are assigned priorities and ranked in the queue; for leaf nodes containing CFD cell information, these cells are inserted in the queue, and cells at the top position are streamed to the visualizer. The distance between directory node and the query point is determined with help of a *MINDIST* approximation defining a lower bound for all objects contained in the subtree of the respective node. For static users, wide spread access methods based on Euclidean or Manhattan distance like presented in 5 and 6 can be utilized to enable view point oriented post-processing. The use of quadratic form distance functions incorporates the user’s view axis into query processing. We choose a matrix for the distance calculation which stretches ellipsoids along the axis of sight  $a$ . One effect is shown in figure 2d: objects along the axis but in the back of the user are assigned the same priorities as equi-distant objects in front of the user.

To additionally consider the viewing direction, we use our human vision oriented distance function  $dist_{hv}$  from above. The corresponding *MINDIST* calculation is performed by a modification of the gradient procedure presented in 7. Starting on an appropriate corner point of the MBR (here: nearest corner point to  $v$  according to Euclidean distance), we iteratively perform a walk on the surface of the MBR defined by the gradient of the quadratic form. After each iteration, we adjust the gradient, as the center points of the intermediate ellipsoids move along the line of sight  $a$ . These iterations are repeated until an ellipsoid is found, which is tangent to the MBR. *MINDIST* is then set to the radius of this ellipsoid. The speed of data extraction can be increased by heuristics to find the approximation  $MINDIST_{approx}$  at the the cost of losing the correctness of the ranking process:

- *Limit number of iterations:* The maximum number of iterations performed during gradient descent is limited to a fixed value.

- *Nearest (Euclidean) corner point*: Calculate the Euclidean distance for each corner point of the MBR point and define  $MINDIST_{approx}$  to be the  $hv$ -distance of this corner point to the query point. This heuristic is equal to the previous one with a limit of zero iterations.
- *Center point*: Determine the center point for the MBR and select the corresponding  $hv$ -distance to be the  $MINDIST_{approx}$  distance.

In the first heuristic, the accuracy of  $MINDIST_{approx}$  compared to the correct  $MINDIST$  depends on the number of iterations performed. Experiments on our sample data demonstrate that a reduction of the number of iterations to 3 only introduced a  $MINDIST_{approx}$  error of 0.035%, but reduced the overall number of iterations to one third during processing of a complete query. The second heuristic is supposed to work well on data sets with relative constant CFD cell sizes. A data set with a large variety in cell sizes perform worse because of the changing quality of  $MINDIST$  approximations. Same applies for the third heuristic, which is calculated with the least complexity of all three heuristics, but performs worse for large MBRs in the upper levels of the index structure with an overall average error of 30%.

The heuristics are not defining lower bounds in all cases, as it holds for the  $MINDIST$  function, so it can possibly happen that a directory node at the first queue position contains data objects that have a lower distance to the query point than objects that were streamed to the visualizer at an earlier stage. Thus, the optimal ranking order is not guaranteed when using heuristics, nevertheless it is ensured that no result cell is omitted. A detailed examination of these heuristics' accuracy can be found in the experimental section (cf. figure [10](#)).

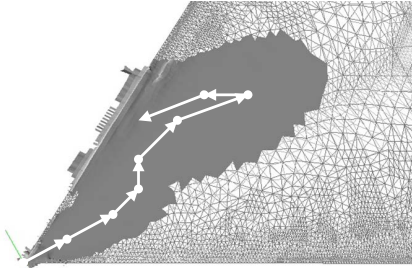
For communication via standard network protocols, data is sent in packages for which the size needs to be tuned appropriately. Answer packets containing numerous cell specifications as well as point data are sent and their content is visualized after reception by the visualizer. One effect is that only cells which are displaced crossing packet boundaries, can be perceived as displaced. The choice of packet sizes is vital and has the following influences: If the packet size is chosen very small, too much communication overhead is generated and streaming of results is slowed down. A very large packet size leads to a discontinuous streaming and jerky result visualization. Our evaluations have shown, that packet sizes between 100 and 1,000 cells performed best regarding query times and visualization smoothness.

## 5 Dynamic Query Adaptation

In this section we present novel approaches to dynamic query processing, supporting interactive users who perform post-processing on CFD data in VR environments and freely move in VR environments and change their view direction.

With our methods, the result stream is simultaneously adapted, with the effect that the result set is steadily growing in the proximity and aligned to the view direction of the user, thus always enabling the user to obtain a quick impression of the result data set from dynamically chosen view positions. Results prior to query update are incorporated during further processing. Figure [5](#) shows an exemplary isosurface extraction from a bird's eye view with a moving user (depicted by the arrows). It can be clearly

observed that the result set is growing in front of the user, whenever a new query is triggered (here: one arrow represents one view modification). This screenshot shows query execution up to 9,000 out of  $\approx 150,000$  result cells.



**Fig. 5.** Dynamic query example

function itself has to be adapted to the changed viewing axis. For quadratic forms, the similarity matrix  $A$  needs recalculation depending on view direction  $a$ , before applying the new quadratic form distance function on the queue elements. When using our human vision oriented  $h\nu$ -distance, the similarity matrix  $A$  is recalculated similarly to the quadratic form distances.

## 5.1 Queue Rearrangement Strategies

The most time-consuming part of queue rearrangements after a view parameter change is to calculate new priority values for all queue elements and then generate a new queue according to these values. We developed heuristics to speed up the expensive regeneration of the queue at the cost of losing the correctness of the result ranking order, but nonetheless a completeness of the result set is guaranteed. The goal is to keep the ranking disorder at a tolerable level and to reduce the effect of queue rearrangement calculations on the overall query processing time.

- Heuristic 1: *Stream leaf nodes*: The queue is emptied and leaf node objects containing cell data are streamed to the visualizer without recalculating their priorities. Only priorities of inner nodes are recalculated and stored in the queue.
- Heuristic 2: *Insert only directory nodes in queue*: The average queue size is reduced by only holding directory nodes in the queue. When visiting a leaf node, all stored cells are streamed to the visualizer. This approach is justified by the fact that cells referenced by leaf nodes at the top of the queue are most likely to appear next in the result stream due to their locality in the index.
- Heuristic 3: *Stream subtree*: To keep the average queue size low, a fixed level  $l$  is defined prior to query invocation. When a node in the tree on level  $l$  is reached during traversal, the referenced subtree is completely streamed to the visualizer. The degree of randomization is depending on the definition of  $l$ . The closer  $l$  is to leaf node level, the higher is the correctness of ranking.

Depending on the distance function used for ranking (cf. section 3), different parameters become relevant for queue reorganization. For the traditional distances like Euclidean or Manhattan distance (view direction independent), only the changes of the view point  $v$  induce a change on the priorities of elements in the queue. In the case of reorganization, the elements in queue are assigned new priorities and the queue is rebuilt. The use of distance functions that take the view direction into account require an additional step when view parameter updates take place. Besides the resorting of elements stored in the queue, the distance

- Heuristic 4: *No recalculation*: No distances of elements in the queue are recalculated. Query processing is continued, ranking visited nodes or cells according to the updated distance function. This drastically reduces the calculation costs due to a “lazy” handling of query updates. Consequently, elements in the queue prior to query update are possibly displaced in the result stream.

Due to the packet-wise communication with the visualizer, the same observation as described in the context of *MINDIST*-approximations (cf. section 4) holds here: Only cells, which are displaced crossing packet boundaries, can be perceived as displaced. For an experimental evaluation of the queue heuristics’ quality, we refer to section 7.

## 5.2 Query Update Frequency

In VR environments, the users’ standpoints and viewing directions are observed constantly by optical or electromagnetic tracking systems at relative high frequencies (e.g. 60 Hertz). Therefore even minor movements yield changes in query parameters with the need for rearrangements of the result queue. Effects of the queue rearrangement calculations on the overall query response time are examined in the experimental section (cf. section 7). We followed three approaches: (1) *time dependent* updates triggered by regular clock cycles, (2) *result cell amount dependent* updates triggered by result portions or (3) *user dependent* updates triggered by user movement passing a predefined threshold, or arbitrary combinations of these. With the time dependent approach, the query is adjusted after a predefined time interval  $t$  has passed, integrating previously calculated results. The second approach is very similar: instead of restarting query processing after time interval  $t$ , queries are adjusted after the streaming of  $n$  result cells. The third approach measures the degree of change  $\Delta$  of view parameters between two query adaptation triggers. If this change is below a predefined threshold, query processing will not be adjusted. This avoids queue rearrangements after only minor changes in view parameters. We propose a  $\Delta$  which reflects the average relative change  $\Delta_i$  of each geometrical dimension  $i$  normed by the geometrical extent  $range_i$  of the data set regarding dimension  $i$  as well as the change of view direction ( $\Delta_\alpha$ ):

$$\Delta := \frac{1}{4} \cdot \left( \frac{\Delta_\alpha}{360^\circ} + \sum_{i=1}^3 \frac{\Delta_{p_i}}{range_i} \right)$$

We integrated the above mentioned query update mechanisms into “IndeGS”. The subjectivity of perception of query result presentation using different settings of parameters  $range_i$  and  $\Delta_\alpha$  renders an extensive experimental evaluation almost impossible. We concentrated our experiments on the *result cell amount dependent* method (cf. section 7).

## 6 Index Support for View-Oriented CFD Post-Processing

The immense data sets which are created during CFD simulations require efficient indexing to enable post-processing with acceptable response times. Many secondary storage index methods exist for appropriately indexing spatial data sets and offer a

multitude of different access methods. So far, no customized index structures in the field of CFD post-processing exist to reflect the characteristics of simulation data sets and existing structures prove inept for these purposes. Over many years, index structures based on the R-Tree [1] like the  $R^*$ -Tree [3] or X-Tree [12] have proven to be appropriate for indexing spatial data. Certain characteristics of CFD data sets render these structures inefficient and ask for a different storage scheme. So far, CFD data has been stored in interval trees, without offering appropriate view-dependent access methods. We have decided to integrate a rectangular index structure as a basis for our query processor. Candidates for such an index structure are R-Trees or Octrees. To simplify matters, we decided to use the R-Tree as the basis for our new index structure for integrating view-dependent access methods. During index creation, we use the bulk-loading method STR (*sort tile recursive*) [13] which appropriately arranges the data before creating the index in a bottom-up manner. We concentrate on the use of CFD simulation data in the format proposed by the open-source library VTK [14]. CFD data consists of cells (tetrahedra, hexahedra, pyramids etc.) which are defined by a topology descriptor and their corner points, which carry information about their geometrical position and additional scalar information (temperature, pressure, density etc.).

A naive approach is to use a standard R-Tree implementation which indexes the elements of their geometrical as well as scalar dimensions. Three severe problems emerge: (1) The use of a high number of dimensions leads to the phenomenon called “Curse of Dimensionality”, which describes the rapidly degrading performance with increasing dimensionality. The data sets in the experimental section include 5 and 8 scalar fields, and together with the geometrical dimensions  $x, y, z$ , this leads to an overall dimensionality of 8 and 11, respectively, which cannot be handled by R-Trees efficiently. With the advancements in the field of CFD simulations, higher dimensionalities through introduction of additional scalar values are easily imaginable. Even data structures like the X-Tree [12], which were developed to handle higher dimensionalities, come to their limits. (2) The leaf nodes need not only store MBRs of cells and the cells themselves, but additionally the exact cell information consisting of all cell-constructing data points and their scalars. This consequently decreases the payload of leaf nodes, thus increasing the height of the R-Tree and query response times. (3) Another effect is the high redundancy when storing data points, as each data point can be part of an arbitrary number of cells and is therefore stored redundantly in the leaf nodes containing the related cells. In the data sets used in our experiments (cf. section 7), each data point is on average part of eight (engine data set) or twelve cells (delta wing data set), and therefore on average stored eight and twelve times, respectively, providing for an explosion of the index structure in size.

**Optimizing the Dimensionality of the Index Structure:** By carefully examining the data sets prior to indexing and taking a reasonable choice of dimensions to be subsumed to form several lower dimensional indexes, the effects of the “Curse of Dimensionality” can be bypassed. If certain scalar values appear more frequent in combined queries, storing these scalars together in one index avoids querying on multiple indexes and stream result joining. As studies on query behavior of post-processing experts are not part of this paper and for reasons of simplicity, we arbitrarily distributed scalar values to indexes with adequate dimensionalities.

**Reducing Index Size and Swapping Out Data Points:** The second and third problem (low payload in leaf nodes, high redundancy of data points) are solved by swapping out cells to a subordinate index. Instead of storing exact cell information consisting of all cell-forming data points with geometrical and scalar information, only point identifiers are stored in leaf nodes. The data points themselves are stored only once in a secondary index, despite their number of occurrences in cells (which can be up to 140 times for the delta wing data set). Thereby redundancy and the size which is occupied by each cell in the leaf nodes is reduced at the same time, significantly reducing the overall index size.

To avoid unnecessary block reads in the secondary index, data points, which are neighboring each other in the data space, are ideally stored in the same or a neighboring blocks on hard disk. We developed heuristics to achieve the local proximity of point data in the secondary index with regards to the proximity of cells in the R-Tree:

1. *“Appearance in index”* heuristics: A traversal of the index’ leaf nodes from left to right induces an order on cells and indirectly on the data points. As points are most likely to appear several times in different cells, we have the choice of choosing their 1<sup>st</sup>, 2<sup>nd</sup>, . . . ,  $n^{\text{th}}$  appearance for defining their order in the secondary index.
2. *“Space-filling curves”* heuristics: We order the point data based on space-filling curves according to their geometrical coordinates. With space-filling curves, we can assign each data point in the 3D space one 1D value, by which the data points are ordered in the secondary index with the effect of local proximity in the data space and index. As a discrete data space is required for space-filling curves, we partition the data space up to a predefined resolution and apply different curves, e.g. Z curve, Hilbert curve and Peano curve. An overview over these curves can be found in [15].

To evaluate our secondary index heuristics, we performed queries extracting isosurfaces on the delta wing data set for different scalar values. We chose the results of our experiments with scalar value “density” (cf. figure 6) as representative for the experimental series, as the heuristics’ performances show similar results on the different scalars. “Random” describes the heuristic using a random order of the data points performs worst. The order given by the VTK source file performs significantly better. The space-filling Z curve and Hilbert curve both show similar performance. Best heuristic proved to be the heuristic choosing the first appearance of the data point regarding leaf node level of the primary index, followed by the heuristic choosing the eighth appearance and the Peano curve.

## 7 Experiments

Our graphics data server “IndeGS” offers communication to arbitrary software components through a clearly defined interface, to enable the inexpensive integration in any VR framework. We integrated “IndeGS” in the ViSTA framework [16] which offers advanced interaction and visualization methods for post-processing in numerical simulations. Together with the post-processing toolkit ViSTA Flowlib [17], ViSTA is a freely available and very powerful toolkit for integrating VR technology in technical and scientific applications.

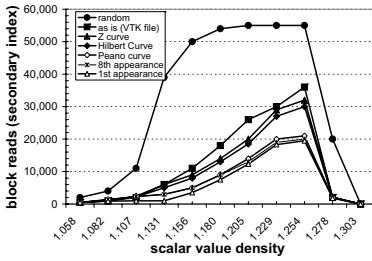


Fig. 6. Secondary index heuristics

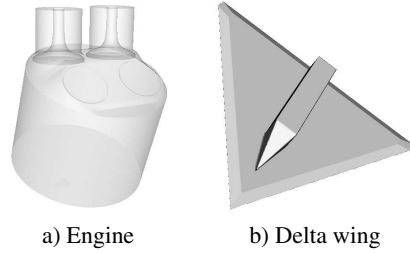


Fig. 7. Sample shapes used in CFD simulations

For our experiments, we used two different CFD data sets: a data set of a simulated fuel injection into a combustion engine cylinder and a data set of simulated aerodynamic flows over the surface and in the surroundings of a delta wing airplane. The basic shapes, around which fluid dynamics are simulated, are shown in figure 7. The engine data set consists of 62 simulated time steps, each time step consisting of 50,000 to 200,000 cells (hexahedra) and 50,000 to 200,000 data points containing 4 scalar values per point, resulting in file sizes between 5 and 30 MB. The delta wing data set comprises of 3 time steps with approximately 15 million cells (tetrahedra and hexahedra) and 4.5 million data points per time step with 8 scalar values each. The file size for the overall delta wing data set is 2.2 GB. The experiments were run on a 2 GHz PC with 2 GB of RAM running Windows XP.

The first experiment shows the effects of the different distance functions on the quality of the user’s first impression of the result. We measure the relative coverage of intermediate results in terms of pixels displayed on a 2D screen at different stages during query execution compared to the number of pixels visualized after query completion. A high percentage of coverage after a short period induces a quick and representative first impression. We performed isosurface extractions using scalar value “mach number” on both CFD data sets. Figure 8 shows the coverage in percent for the different ranking methods, with the elapsed time (including secondary storage IO, communication as well as visualization costs) displayed on the x-axis for the delta wing data set. It can be clearly observed that query processing using our  $h\nu$ -distance covers the largest part of the screen compared to other distance functions at any time during query processing, although it is the most complex to calculate. The unranked processing performs worst, as cells are almost randomly streamed to the visualizer and cells with a very large distance to the viewer or outside the vision field consequently only contribute a small number of pixels to the visible result set. The quadratic forms distance function works slightly better than Euclidean distance, but the absolute advantage of a higher coverage is reduced by the more complex calculations. Heuristic 1 shows similar behavior as the unranked processing, but speeds up after approx. 3 seconds. It populates the screen at a low rate in the beginning, as many cells far away from the viewer are visualized early, which only contribute few pixels. Heuristic 2 provides for a high coverage after a very short time, but slows down after a few seconds and is overtaken by the ranking using Euclidean or quadratic form distances, due to the effect described in section 3.2. Heuristics 3 and 4 show similar behavior and are omitted for visibility reasons. We

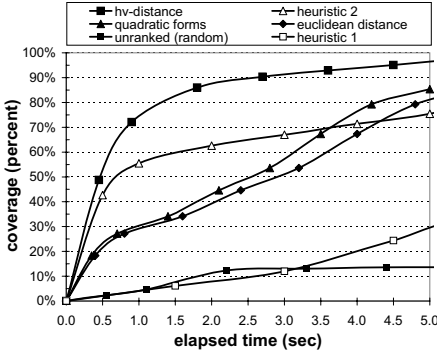


Fig. 8. Coverage “delta wing”

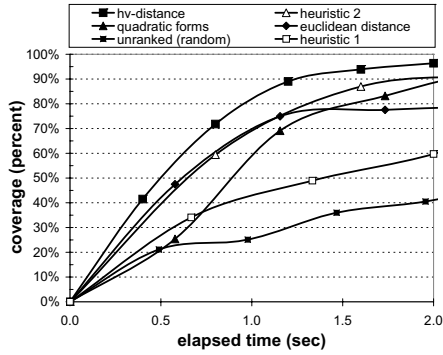
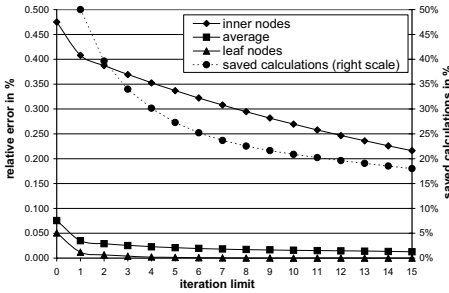


Fig. 9. Coverage “engine”



a) MINDIST-error

iterations	displaced cells	maximum displacement	displaced cells (packet mode)	max. displacement (packet mode)	overall query processing speedup
1	6968	1402	146	14	114.71%
2	2033	1025	31	10	113.97%
3	850	719	11	7	113.37%
4	514	446	5	4	112.95%
5	290	240	2	2	111.94%
6	110	54	1	1	111.43%
7	48	1	0	0	110.48%
8	20	1	0	0	109.55%
9	18	1	0	0	108.33%
10	4	1	0	0	105.41%
11	0	0	0	0	102.31%

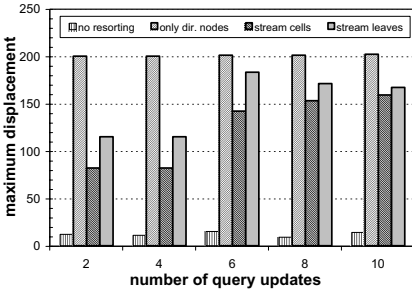
b) effect on query processing (50,000 cells)

Fig. 10. MINDIST approximations evaluation

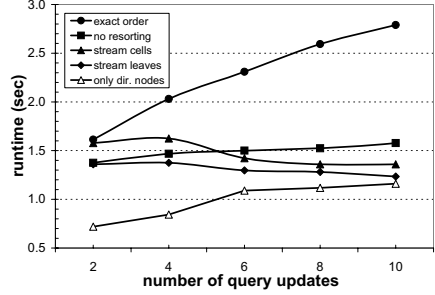
performed a comparable experiment on the engine data set (cf. figure 9), showing that our *hv*-distance outperforms the competing ranking methods. All ranking methods reach higher percentages after a shorter time in comparison to the delta wing results, which is a consequence of the significantly lower complexity of the engine data set. To prove the suitability of our approaches in the context of very large data sets, we focus on the delta wing data set in the following.

In further experiments performed using the delta wing data set, we compare our *MINDIST*-heuristics (cf. section 4) to the correct *MINDIST*. When calculating the exact *MINDIST* using the gradient descent method, a maximum of 132 iterations is needed for a very limited number of *MINDIST*-calculations. When limiting the number of iterations to 15, the calculated *MINDIST*<sub>approx</sub> is on average only 0.012% above the correct *MINDIST*. Figure 10a shows the average error of *MINDIST*<sub>approx</sub> for leaf and inner nodes after 1 to 15 iterations compared to the exact *MINDIST*. The averaged error is dominated by the leaf node error, as the number of leaf nodes visited is up to ≈16 times higher than the number of inner nodes. Figure 10a also displays the percentage of saved *MINDIST* iterations when applying an iteration limit. Figure 10b shows the number of





a) displacement of cells (packet size: 1,000 cells)

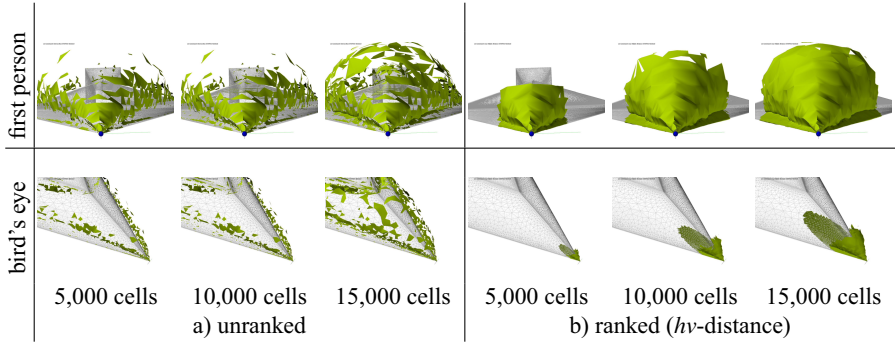


b) heuristic runtimes

Fig. 11. Dynamic query heuristics evaluation

cells (out of an overall amount of 50,000 cells) which are not delivered in their correct ranking position (column “displaced cells”). Column “maximum displacement” shows the maximum difference of displaced cells between the correct position and the position in the approximated ranking. As the cells are delivered over the network in packets (in this experiment: 100 cells per packet to guarantee a smooth visualization), cells that are not displaced in different packets are not perceived as displaced. The next two columns (“packet mode”) show the displacement in the scale of packets, which is minor even with the relative small packet size chosen for this experiment. The last column shows the overall speedup of query processing (including IO costs) that is achieved by limiting the number of iterations. When setting the limit to 7 iterations, no displaced cell is perceived and a speedup of 110% is achieved.

We examined the heuristics for dynamic query adaptation from section 5 by measuring the disorder of the ranking and the runtimes for each of the heuristics. The result set consists of 29,172 cells and the packet size was set to 1,000 cells, and the query point was updated 2 to 10 times during query processing. Figure 11a shows the maximum packet-wise displacement of cells when comparing the result streams with the correct stream, which is produced by a complete regeneration of the priority queue after each query update. The heuristics “no resorting” and “insert only directory nodes in queue” produce displacement maxima independent of query update frequency, whereas the “stream leaves” (equal to heuristic “stream subtree” heuristic from section 5.1 with  $l$  set to leaf node level) and “stream cells” heuristics’ maxima are increasing, as they both remove many elements (leave nodes and cells, respectively) from the queue at query update, streaming a high number of cells in incorrect order, producing a high overall degree of disorder. Figure 11b shows the runtimes for the query execution using exact ranking as well as the heuristics: The complexity for the exact ranking rises with number of updates due to the complexity of distance calculations for each update. Runtimes for “stream cells” and “stream leaves” are getting lower marginally with higher update frequency: the removal of cells and leaf nodes at query update reduces the average queue size and consequently the queue management costs. Recapitulating the experiment results, heuristics with high runtimes perform best (“exact order”), whereas heuristics with low runtimes (e.g. “only directory nodes”) produce higher levels of disorder, affecting the smoothness of visualization.



**Fig. 12.** Query processing “delta wing tip”

Figure 12 shows an exemplary query processing (isosurface extraction on delta wing data set for scalar “density”) at different stages of processing. The simulated query point is located in front of the tip of the delta wing and the screen shots show the visualized partial results as well from first person view as from a bird’s eye perspective. We focused here on the unranked (figure 12a) and ranked processing using our  $hv$ -distance (figure 12b).

## 8 Conclusion and Future Work

In this paper we presented our new index based graphics data server “IndeGS”, a component capable of being integrated in any VR framework and offering indexing and access methods for *computational fluid dynamics* (CFD) data. “IndeGS” enables the handling of data of almost arbitrary size by employing an efficient spatial external memory indexing structure. We hereby overcome the main memory limitations which are a restrictive factor for the effectiveness of post-processing on very large data sets. “IndeGS” also offers an access structure which supports dynamic view-dependent result extraction based on streaming techniques. “IndeGS” significantly increases the productivity of users performing post-processing on CFD data by view-dependent result streaming which enables the user to catch a quick first impression of his query. Instead of waiting for the whole query to be processed, the users can change query type and parameters and freely roam the VR environment while their queries are processed and displayed. For this purpose, we developed a human vision oriented distance function to reflect the characteristics of human vision, where objects in the focus of sight are perceived with greater interest.

In future work, we plan to explore the suitability of index structures utilizing relational database systems, like proposed with the RI-Tree [18] which efficiently answers interval queries on very large data sets. Integrating time as an additional dimension to the data sets offers new possibilities of browsing query results over time. In connection with CFD data sets that were simulated including a time component, the user can dynamically change view parameters as well as visualized time steps. We plan to integrate intelligent prefetching strategies for time steps close to the currently displayed time step.

## Acknowledgments

The authors would like to thank Christian Bischof and Marc Wolter from the *Center for Computing and Communication* of the RWTH Aachen University for providing access to their virtual reality infrastructure and for supporting the integration of “IndeGS” in the *ViSTA* framework. We also thank Christian Klaus and Dennis Meichsner for their valuable work on the basic implementation of the graphics data server.

## References

1. Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: SIGMOD Conf, pp. 47–57 (1984)
2. Sellis, T.K., Roussopoulos, N., Faloutsos, C.: The R+-Tree: A Dynamic Index for Multi-Dimensional Objects.. In: VLDB Conference, pp. 507–518 (1987)
3. Beckmann, N., Kriegel, H.-P., Schneider, R., Seeger, B.: The R\*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In: SIGMOD Conf. pp. 322–331 (1990)
4. Chiang, Y.J., Silva, C.T., Schroeder, W.J.: Interactive out-of-core isosurface extraction. In: VIS Conference, pp. 167–174 (1998)
5. Roussopoulos, N., Kelley, S., Vincent, S.: Nearest Neighbor Queries. In: SIGMOD Conference, pp. 71–79 (1995)
6. Hjaltason, G.R., Samet, H.: Ranking in Spatial Databases. In: Egenhofer, M.J., Herring, J.R. (eds.) SSD 1995. LNCS, vol. 951, pp. 83–95. Springer, Heidelberg (1995)
7. Seidl, T., Kriegel, H.-P.: Efficient user-adaptable similarity search in large multimedia databases. In: VLDB Conference, pp. 506–515 (1997)
8. Iwerks, G.S., Samet, H., Smith, K.P.: Continuous k-nearest neighbor queries for continuously moving points with updates. In: VLDB Conference, pp. 512–523 (2003)
9. Saltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the positions of continuously moving objects. In: SIGMOD Conference, pp. 331–342 (2000)
10. Mokbel, M., Xiong, X., Aref, W.: SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In: SIGMOD Conference, pp. 623–634 (2004)
11. Song, Z., Roussopoulos, N.: K-nearest neighbor search for moving query point. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, pp. 79–96. Springer, Heidelberg (2001)
12. Berchtold, S., Keim, D.A., Kriegel, H.-P.: The X-Tree: An Index Structure for High-Dimensional Data. In: VLDB Conference, pp. 28–39 (1996)
13. Leutenegger, S.T., Edgington, J.M., Lopez, M.A.: STR: A Simple and Efficient Algorithm for R-Tree Packing. In: ICDE, pp. 497–506 (1997)
14. Schroeder, W., Martin, K., Lorensen, B.: The Visualization Toolkit. Kitware Inc. (2004)
15. Sagan, H.: Space-filling curves. Springer, Heidelberg (2006)
16. Reimersdahl, T.v., Kuhlen, T., Gerndt, A., Heinrichs, J., Bischof, C.: ViSTA - a multimodal, platform-independent VR-Toolkit based on WTK, VTK, and MPI. In: IPT Workshop (2000)
17. Schirski, M., Gerndt, A., Reimersdahl, T.v., Kuhlen, T., Adomeit, P., Lang, O., Pischinger, S., Bischof, C.: ViSTA FlowLib - framework for interactive visualization and exploration of unsteady flows in virtual environments. In: EGVE Workshop, pp. 77–85. ACM Press, New York (2003)
18. Kriegel, H.-P., Pötke, M., Seidl, T.: Managing intervals efficiently in object-relational databases. In: VLDB Conference, pp. 407–418 (2000)

# Generalizing the Optimality of Multi-step $k$ -Nearest Neighbor Query Processing

Hans-Peter Kriegel, Peer Kröger, Peter Kunath, and Matthias Renz

Institute for Computer Science, Ludwig-Maximilians Universität München  
{kriegel,kroegerp,kunath,renz}@dbs.ifi.lmu.de  
<http://www.dbs.ifi.lmu.de>

**Abstract.** Similarity search algorithms that directly rely on index structures and require a lot of distance computations are usually not applicable to databases containing complex objects and defining costly distance functions on spatial, temporal and multimedia data. Rather, the use of an adequate multi-step query processing strategy is crucial for the performance of a similarity search routine that deals with complex distance functions. Reducing the number of candidates returned from the filter step which then have to be exactly evaluated in the refinement step is fundamental for the efficiency of the query process. The state-of-the-art multi-step  $k$ -nearest neighbor ( $k$ NN) search algorithms are designed to use only a lower bounding distance estimation for candidate pruning. However, in many applications, also an upper bounding distance approximation is available that can additionally be used for reducing the number of candidates. In this paper, we generalize the traditional concept of  $R$ -optimality and introduce the notion of  $R_I$ -optimality depending on the distance information  $I$  available in the filter step. We propose a new multi-step  $k$ NN search algorithm that utilizes lower- and upper bounding distance information ( $I_{lu}$ ) in the filter step. Furthermore, we show that, in contrast to existing approaches, our proposed solution is  $R_{I_{lu}}$ -optimal. In an experimental evaluation, we demonstrate the significant performance gain over existing methods.

## 1 Introduction

In many database applications such as molecular biology, CAD systems, multimedia databases, medical imaging, location-based services, etc. the support of similarity search on complex objects is required. In general, the user wants to obtain as many true hits as soon as possible. Usually, in all these applications, similarity is measured by metric distance functions. The most popular query types are distance range (or  $\varepsilon$ -range) queries,  $k$ -nearest neighbor ( $k$ NN) queries, and – more recently – reverse  $k$ NN queries. Those queries can be supported by index structures such as the R-tree [1] or the R\*-tree [2] and their variants for Euclidean data or by the M-tree [3] and its variants for general metric data. These index structures are designed for shrinking down the search space of tentative hits in order to scale well for very large databases.

However, index structures usually invoke a large number of distance computations and, thus, do neither account for the increasing complexity of the database objects nor for the costly distance functions used for measuring the similarity. To cope with complex data objects and costly distance functions, the paradigm of multi-step query processing has been defined for spatial queries such as point queries and region queries [4,5]. This paradigm has been extended to similarity search in databases of complex objects performing distance range queries [6,7] and  $k$ NN queries [8,9]. The key idea of multi-step query processing is to apply a so-called *filter step* using a cheaper distance function, the so-called *filter distance*, in order to prune as many objects as possible (as true hits or true drops). For the remaining candidates, for which the query predicate cannot be decided using the filter distance, the exact (more costly) distance needs to be evaluated in the so-called *refinement step*.

In most applications, two different types of filter distances are commonly used for multi-step query processing. First, a lower bounding filter distance produces distances that are always lower or equal to the exact distance and can be used to discard true drops. Second, an upper bounding filter distance produces distances that are always greater or equal to the exact distance and can be used to identify true hits. While both types of filter distances have successfully been used for distance range queries, all existing multi-step  $k$ NN query processing algorithms only use the lower bounding filter distance. Thus, these approaches can only prune true drops, but cannot identify true hits in the filter step. Furthermore, these approaches cannot report true hits already after the filter step, but need to refine the candidates before reporting them as hits. However, in applications where the results are further processed and this processing is quite costly due to the complexity of the data objects, it is desirable to output true hits as soon as possible even if the result set is not yet complete. Obviously, using an upper bounding filter distance may allow to output a first set of true hits already after the filter step before refinement. In addition, using an upper bounding filter distance could significantly reduce the number of candidates that need to be refined and, thus, could clearly improve query execution times. As a consequence, the storage required to manage intermediate candidates during the entire filter-refinement procedure can also be reduced.

In general, using also upper bounding distance information in the filter step yields several advantages as long as no ranking of the  $k$ NNs is needed. However, in many applications, users only want the result of a given  $k$ NN query rather than a ranking. For example, a restaurant owner planning a public relations campaign by sending a fixed number  $k$  of flyers to potential costumers may choose the  $k$  customers with the smallest distance to the restaurant's location. In addition, many data mining algorithms that rely on  $k$ NN computation such as density-based clustering,  $k$ NN classification, or outlier detection only require the result of a  $k$ NN query, but not its ranking. Many of those methods use the result of  $k$ NN queries for further processing steps.

In this paper, we propose a novel multi-step query processing algorithm for  $k$ NN search using both a lower and an upper bound in the filter step. We show

that this algorithm is optimal, i.e. that it produces a minimum number of candidates which need to be refined. For that purpose, we generalize the notion of  $r$ -optimality taking the distance estimations available in the filter step into account. In a broad experimental evaluation, we show that when using our novel multi-step query algorithm, the application of an upper bound in addition to a lower bound in the filter step yields a significant performance gain over the traditional approach using only lower bounding distance approximations. In particular, we show that this performance gain is not only in terms of the number of candidates that need to be refined, implying a runtime improvement, but also in terms of space requirements.

The rest of the manuscript is organized as follows. In Section 2 we discuss existing multi-step  $k$ NN query processing algorithms. A generalized notion of the optimality for multi-step  $k$ NN algorithms is presented in Section 3. Section 4 presents a novel multi-step  $k$ NN algorithm that meets the requirements of our new generalized optimality. Section 5 presents our experimental evaluation and Section 6 concludes the paper.

## 2 Multi-step $k$ NN Query Processing

Let  $\mathcal{D}$  be a database of objects and  $\text{dist}$  be a distance function on these objects. For a given query object  $q$  and a given positive integer  $k \in \mathbb{N}^+$ , a  $k$ -nearest neighbor ( $k$ NN) query on a database  $\mathcal{D}$  retrieves the objects in  $\mathcal{D}$  that have the  $k$  smallest distances to  $q$ , formally

**Definition 1** ( *$k$ NN query,  $k$ NN-distance*). *For a query object  $q$  and a query parameter  $k \in \mathbb{N}$ , a  $k$ NN query in  $\mathcal{D}$  returns the smallest set  $NN^{\mathcal{D}}(q, k) \subseteq \mathcal{D}$  that contains (at least)  $k$  objects from  $\mathcal{D}$ , for which the following condition holds:*

$$\forall o \in NN^{\mathcal{D}}(q, k), \forall o' \in \mathcal{D} - NN^{\mathcal{D}}(q, k) : \text{dist}(q, o) < \text{dist}(o', q).$$

With  $nn_k\text{-dist}(q, \mathcal{D}) = \max\{\text{dist}(q, o) | o \in NN^{\mathcal{D}}(q, k)\}$  we denote the  $k^{\text{th}}$  nearest neighbor distance (also called  $k$ NN-distance) of  $q$  (w.r.t.  $\mathcal{D}$ ).

Since the database  $\mathcal{D}$  is usually clear from context, we write  $NN(q, k)$  and  $nn_k\text{-dist}(q)$  instead of  $NN^{\mathcal{D}}(q, k)$  and  $nn_k\text{-dist}(q, \mathcal{D})$ , respectively.

Let us note that in case of tie situations we include all ties into the result set. Thus, the cardinality of the result set may exceed  $k$  and the result of a  $k$ NN query is deterministic.

A naive solution for answering a given  $k$ NN query is to scan the entire database  $\mathcal{D}$  and test for each object if it is currently among the  $k$ -nearest neighbors. This naive algorithm has a runtime complexity of  $O(N \cdot QP)$ , where  $N = |\mathcal{D}|$  denotes the number of objects in  $\mathcal{D}$  and  $QP$  denotes the cost of evaluating the query predicate for one single object, which is usually dominated by the complexity of the applied distance function  $\text{dist}$ . Obviously, using such a naive solution for  $k$ NN query processing is very expensive and not feasible for a very large set of complex objects. In fact, the problem is two-fold: On one hand, since the

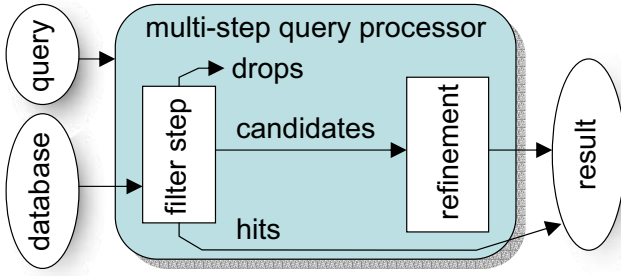


Fig. 1. Multi-step query processor

number of objects  $N$  in a database is usually very large, a sequential scan over all objects to evaluate the query predicate would produce very high I/O cost. On the other hand, due to the complexity of the distance function used in the above mentioned applications, the evaluation of the query predicate  $QP$  of one single object usually demands high CPU cost. In addition, many applications deal with very large objects such as audio or video sequences. As a consequence, the evaluation of the query predicate also invokes I/O cost and, thus, the cost for evaluating a query predicate become the bottleneck during query processing.

Indexing methods (i.e., single-step query processing solutions) that enable to prune large parts of the search space help to reduce the set of objects for which the query predicate has to be evaluated, i.e. address the first problem of high I/O cost due to a sequential scan. Theoretically, using an index, the runtime complexity is decreased to  $O(\log N \cdot QP)$ . However, index structures have two important drawbacks when dealing with complex objects and costly distance functions. First, indexes in general rely on the assumption that the distance function used is a metric. Otherwise, if the distance function defined on the database objects is not metric (in particular if the triangle inequality is not fulfilled), indexes cannot be applied. Second, and more severely, indexes are primarily designed to reduce the number of page accesses, but usually invoke the evaluation of the query predicate for many objects, e.g. during the index traversal and for the evaluation of the candidates reported from the indexing method. Obviously, when dealing with complex objects where  $QP$  is the bottleneck, a single-step query processing strategy is no longer feasible. Rather, a multi-step query processing approach is required reducing the set of result candidates in a filter step using an approximate evaluation of the query predicate which can be computed much faster than the exact evaluation (and optimally does not invoke extra I/O cost). This reduces the  $QP$ -part of the runtime complexity. In the filter step, as many hits and drops as possible (the amount obviously depends on the quality of the approximation) may already be identified. Finally, the remaining candidates have to be exactly evaluated in a refinement step in order to complete the result set. Since the filter step can also be supported by an index structure, additionally the  $N$  part in the runtime complexity is decreased. A schematic description of the multi-step query processor is illustrated in Figure 1.

$k$ -NearestNeighborSearch( $q, k$ )
1 initialize <i>ranking</i> on index $I$
2 initialize $result = \text{sorted\_list}(key, object)$
3 initialize $d_{max} = \infty$ // stop distance
4 <b>while</b> $o = \text{ranking.getnext}()$ <b>and</b> $LB(q, o) \leq d_{max}$ <b>do</b>
5 <b>if</b> $\text{dist}(q, o) \leq d_{max}$ <b>then</b> $result.insert(\text{dist}(q, o), o)$
6 <b>if</b> $result.length \geq k$ <b>then</b> $d_{max} = result[k].key$
7   remove all entries from $result$ where $key > d_{max}$
8 <b>endwhile</b>
9 report all entries from $result$

**Fig. 2.** Multi-step  $k$ NN algorithm proposed in [9]

Obviously, a multi-step  $k$ NN algorithm is correct if the algorithm does neither produce false drops in the filter step, i.e. all drops do not fulfill the query predicate, nor produce false hits, i.e. all hits reported from the filter step really fulfill the query predicate.

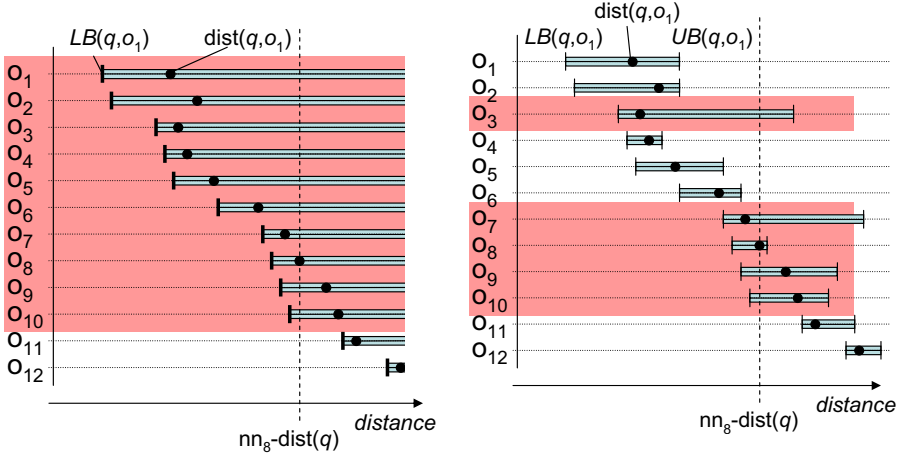
The state-of-the-art multi-step  $k$ NN search method is the algorithm proposed in [9]. It uses a lower-bounding distance estimation  $LB$  in the filter step which is always lower or equal to the exact distance, i.e. for any query object  $q$  the *lower bounding property*

$$\forall o \in \mathcal{D} : LB(q, o) \leq \text{dist}(q, o)$$

holds. A lower bounding filter can be used to prune true drops. The basic idea of the proposed method in [9] is to iteratively generate candidates sorted by ascending lower bounding filter distances to the query object  $q$ . For that purpose, a ranking [10] of the database objects w.r.t. their filter distances to  $q$  is used. The multistep  $k$ NN query processing proposed in [9] is initialized with the first  $k$  objects from the ranking sequence having the  $k$  smallest filter distances. These objects are refined, i.e. their exact distances to  $q$  are computed, and are inserted into the current result set (sorted by ascending exact distances to  $q$ ), representing the  $k$ NN of  $q$  w.r.t. the already refined objects. A so-called stop distance  $d_{max}$  is initialized as the distance of  $q$  to the  $k^{th}$  object in the current result set representing the  $k$ NN-distance of  $q$  w.r.t. the already refined objects. Now, an iteration starts that, in each step, performs the following: First, the next object  $c$  from the ranking sequence is fetched. If this object has a lower bounding distance estimation to  $q$  larger than the stop distance, i.e.  $LB(q, c) > d_{max}$ , the iteration stops. Otherwise,  $c$  is refined, i.e. the exact distance  $\text{dist}(q, c)$  is computed, and, if necessary,  $c$  is added to the current result set and the stop distance  $d_{max}$  is adjusted. When the iteration stops, the current result set contains the  $k$ NN of  $q$ . The pseudo code of this algorithm is depicted in Figure 2.

In [9], the optimality w.r.t. the number of refined objects necessary for multistep  $k$ NN query processing is evaluated and formalized by the concept of  $R$ -optimality. An algorithm is defined to be  $R$ -optimal, if it produces no more candidates for refinement than necessary. It is shown that a multi-step  $k$ NN





(a) only with lower-bounding filter distance.

(b) with lower- and upper-bounding filter distances.

**Fig. 3.**  $k$ -nearest neighbor candidates ( $k=8$ )

algorithm is correct and  $R$ -optimal iff it exactly retrieves the candidate set  $\{o \mid LB(o, q) \leq nn_k\text{-dist}(q, \mathcal{D})\}$  from the filter step.

### 3 Generalizing the Definition of Optimality

As indicated above, the algorithm presented in [9] uses only a lower bounding distance estimation in the filter step. However, it is in general sensible to use additional information, in particular an upper bounding filter distance. An upper bounding filter distance estimation  $UB$  is always greater or equal to the exact distance, i.e. for any query object  $q$  the following *upper bounding property* holds:

$$\forall o \in \mathcal{D} : UB(q, o) \geq \text{dist}(q, o).$$

Using also an upper bounding filter distance yields several important advantages. First, beside pruning true drops with the lower bound we can additionally identify true hits using the upper bounding filter distance. This is illustrated in Figure 3. It depicts for a given query object  $q$  the exact distances  $\text{dist}(q, o)$  for twelve sample objects  $o_1, \dots, o_{12}$  ( $k = 8$ ). We can distinguish two cases of correct candidate sets returned from the filter-step depending on the distance approximations used: Figure 3(a) shows the case where only a lower bounding filter distance  $LB$  is given in the filter-step and Figure 3(b) shows the case where we are given both a lower bounding  $LB$  and an upper bounding  $UB$  filter distance (illustrated by the bars). In both cases, we marked those objects which have to be returned as candidates from the filter-step. In the first case (cf. Figure 3(a)), we have to refine all objects  $o \in \mathcal{D}$  for which the lower bounding distance  $LB(q, o)$

is smaller than or equal to the  $k$ NN-distance of  $q$ , i.e. we have to refine the ten objects  $o_1, \dots, o_{10}$ . In fact, this does not hold for the second case (cf. Figure 3(b)), where the objects  $o_1, o_2, o_4, o_5$  and  $o_6$  can immediately be reported as true hits in the filter step due to the upper bounding distance information. Thus, in contrast to Case 1, the objects  $o_1, o_2, o_4, o_5$  and  $o_6$  need not to be refined.

A second advantage of using also an upper bounding filter distance is that the storage requirements of the  $k$ NN algorithm can be significantly reduced. As discussed above, [9] uses a ranking algorithm (e.g. [10]). Such a ranking algorithm is usually based on a priority queue. For  $k$ NN queries, we can delete true drops (identified using  $LB$ ) from that queue. Analogously, we can also delete the true hits (identified using  $UB$ ) from the queue. Thus, the storage cost during query execution are reduced. We will see in our experiments, that using both an upper and a lower bounding filter distance significantly decreases the size of the priority queue (used for producing the ranking sequence) compared to algorithms that use only a lower bound.

Last but not least, a third advantage of using not only a lower bound but also an upper bound in the filter step is the fact that those true hits, identified already in the filter step, can be immediately reported to the user. Thus, the user may receive a part of the complete result directly after the filter step before the query process is completely finished, sometimes even before the exact evaluation of the query predicate for any object has been carried out. The produced hits in the filter step allow the user to inspect the first results very early which is obviously a big advantage in real applications. Unfortunately, none of the existing multi-step query processors provide this feature because none of these methods use suitable distance estimations in the filter step.

The first obvious question following from these considerations is whether the algorithm proposed in [9] is really  $R$ -optimal. We will see that the answer to this question is “yes” and “no” – and in fact depends on the type of information (only lower bound or upper and lower bound) available in the filter step. In the traditional sense, a multi-step  $k$ NN algorithm is called  $R$ -optimal if it does not produce more candidates in the filter-step than necessary. As discussed above, the number of candidates that definitely need to be refined depends on the distance approximation available in the filter step. Obviously, it is sensible to define “optimality” in the context of which kind of information  $I$  is available in the filter step. In the following, we present the notion of  $R_I$ -optimality as a generalization of the traditional  $R$ -optimality.

**Definition 2 (Generalized Optimality).** *Given an information class  $I$  defining a set of distance approximations available in the filter step, a multi-step  $k$ NN algorithm is called  $R_I$ -optimal if it does not produce more candidates in the filter-step than necessary.*

Interesting information classes are  $I_l = \{LB\}$ , i.e. only a lower bounding distance approximation is available in the filter step, and  $I_{lu} = \{LB, UB\}$ , i.e. both a lower and an upper bounding distance approximation is available in the filter step. In general,  $R_{I_l}$ -optimality corresponds to the traditional concept of  $R$ -optimality proposed in [9]. The lemma given in [9] identifies those algorithms

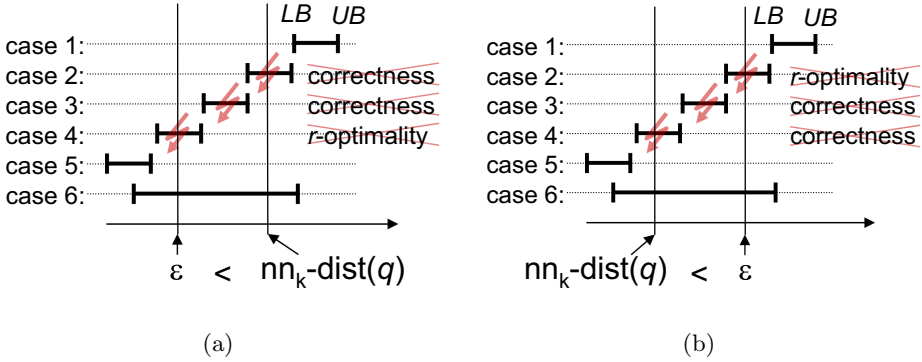


Fig. 4. Illustration of the proof of Lemma 1

which are correct and  $R_{I_r}$ -optimal. It states that a multi-step  $k$ NN algorithm is correct and  $R_{I_r}$ -optimal if and only if it exactly retrieves the candidate set  $\{o \mid LB(q, o) \leq nn_k\text{-dist}(q, \mathcal{D})\}$  from the filter step. In [9], such an  $R_{I_r}$ -optimal algorithm is presented.

For the information class  $I_{lu}$  we can also identify the minimum set of candidates that is produced by a correct and  $R_{I_{lu}}$ -optimal algorithm.

**Lemma 1.** *A multi-step  $k$ NN algorithm is correct and  $R_{I_{lu}}$ -optimal, iff it refines the candidate set*

$$\{o \in \mathcal{D} \mid LB(q, o) \leq nn_k\text{-dist}(q, \mathcal{D}) \leq UB(q, o)\} \tag{Case 1}$$

*if there are more than  $k$  candidates  $c \in \mathcal{D}$  with  $LB(q, c) \leq nn_k\text{-dist}(q, \mathcal{D})$  and, otherwise, it refines the candidate set*

$$\{o \in \mathcal{D} \mid LB(q, o) \leq nn_k\text{-dist}(q, \mathcal{D}) < UB(q, o)\} \tag{Case 2}$$

*from the filter step.*

*Proof.* Assume the following algorithm: For an arbitrary query range  $\varepsilon$ , we obtain the object set  $\mathcal{S} = \{o \in \mathcal{D} \mid LB(q, o) \leq \varepsilon\}$ . The objects in  $\mathcal{D} - \mathcal{S}$  are pruned as true drops. Then, we retrieve the candidate set  $\mathcal{C} = \{o \in \mathcal{S} \mid \varepsilon < UB(q, o)\} \subseteq \mathcal{S}$  which has to be refined in the refinement step, the remaining objects in  $\mathcal{S} - \mathcal{C}$  are immediately reported as hits. We show that this algorithm can only be correct and  $R_{I_{lu}}$ -optimal if  $\varepsilon = nn_k\text{-dist}(q, \mathcal{D})$ .

1. Let  $\varepsilon < nn_k\text{-dist}(q, \mathcal{D})$ :

Then, there may exist an object  $o \in \mathcal{D}$  for which the following estimation chain holds:  $\varepsilon < LB(q, o) \leq \text{dist}(q, o) \leq nn_k\text{-dist}(q, \mathcal{D})$  (cf. Cases 2-3 in Figure 4(a)). The last inequality implies that  $o \in NN(q, k)$ . However, due to the first inequality of the chain, we have  $o \notin \mathcal{S}$ , i.e.  $o$  will be pruned as a false drop. This contradicts the correctness of the algorithm.

Furthermore, there may exist an object  $o \in \mathcal{D}$  for which the following estimation chain holds:  $LB(q, o) \leq \varepsilon < UB(q, o) \leq nn_k\text{-dist}(q, \mathcal{D})$  (cf. Case

4 in Figure 4(a)). The first and second inequalities indicate that  $o \in \mathcal{C}$ , i.e.  $o$  is a candidate that will be refined. However, due to the third inequality, it can be definitely decided that  $o \in NN(q, k)$  and, thus, a refinement of the distance between  $q$  and  $o$  is not necessary which contradicts the  $R_{I_u}$ -optimality.

2. Let  $\varepsilon > \text{nn}_k\text{-dist}(q, \mathcal{D})$ :

Then, there may exist an object  $o \in \mathcal{D}$  for which  $\text{nn}_k\text{-dist}(q, \mathcal{D}) < LB(q, o) \leq \varepsilon < UB(q, o)$  (cf. Case 2 in Figure 4(b)), i.e.  $o \in \mathcal{C}$  will be refined. However, due to the lower bounding property,  $\text{nn}_k\text{-dist}(q, \mathcal{D}) < LB(q, o) \leq \text{dist}(q, o)$  holds. Thus,  $o \notin NN(q, k)$ , and the algorithm cannot be  $R_{I_u}$ -optimal.

Furthermore, there may exist an object  $o \in \mathcal{D}$  for which  $\text{nn}_k\text{-dist}(q, \mathcal{D}) < \text{dist}(q, o) \leq UB(q, o) \leq \varepsilon$  (cf. Cases 3 and 4 in Figure 4(b)). The second and the third inequalities indicate that  $LB(q, o) \leq \varepsilon$  and  $o \in \mathcal{S} - \mathcal{C}$ , i.e.  $o$  is reported as hit without refinement. However, from the first inequality it follows that  $o \notin NN(q, k)$  and, thus, the algorithm cannot be correct.

Thus, only  $\varepsilon = \text{nn}_k\text{-dist}(q, \mathcal{D})$  achieves correctness and  $R_{I_u}$ -optimality does not lead to any contradiction.

Obviously, all objects in the set  $\{o \in \mathcal{D} | LB(q, o) \leq \text{nn}_k\text{-dist}(q, \mathcal{D}) < UB(q, o)\}$  have to be refined in order to determine, whether they fulfill the query predicate or not (Case 2). All objects  $o \in \mathcal{D}$  for which  $UB(q, o) \leq \text{nn}_k\text{-dist}(q, \mathcal{D})$  holds, need not be refined, because  $\text{dist}(q, o) \leq \text{nn}_k\text{-dist}(q, \mathcal{D})$  due to the upper bounding property. However, if the number of candidates  $c$  with  $LB(q, c) \leq \text{nn}_k\text{-dist}(q, \mathcal{D})$  exceeds  $k$  (Case 1), we cannot decide whether those objects  $c$  for which  $UB(q, o) = \text{nn}_k\text{-dist}(q, \mathcal{D})$  holds, are hits or drops. The reason for this is the following: no algorithm can anticipate the real  $\text{nn}_k\text{-dist}(q, \mathcal{D})$  and, thus, other candidates  $c'$  could have a smaller  $\text{dist}(q, c')$ . If so, we would have  $\text{nn}_k\text{-dist}(q, \mathcal{D}) < UB(q, c)$  and, thus,  $c$  would be a true drop. To make this decision in a correct way,  $c$  needs to be refined although  $UB(q, c) \geq \text{nn}_k\text{-dist}(q, \mathcal{D})$ . In tie situations, there may be more such objects  $c$  that need to be refined although  $\text{nn}_k\text{-dist}(q, \mathcal{D}) \geq UB(q, c)$ .  $\square$

At first glance, Case 1 of Lemma 4 may appear to be rather arbitrary. However, as discussed in the proof of Lemma 4, there may be some situations where we need to consider both cases. Let  $o_i$  be the  $k$ NN of a query object  $q$  such that  $UB(q, o_i) = \text{dist}(q, o_i) = \text{nn}_k\text{-dist}(q, \mathcal{D})$ . Let the object  $o_j$  be a candidate with  $LB(q, o_j) \leq UB(q, o_i) = \text{nn}_k\text{-dist}(q, \mathcal{D})$ . Then,  $o_j$  cannot be pruned before the exact  $k$ NN-distance has been computed. In addition, if we have a tie situation, e.g.  $\text{dist}(q, o_j) = \text{nn}_k\text{-dist}(q, \mathcal{D}) = \text{dist}(q, o_i)$ , the  $k$ NN set of  $q$  cannot be determined correctly without the refinement of the object  $o_i$  (contradicting Case 2). The reason for this is that we cannot evaluate the query predicate for  $o_j$  correctly even if we refine  $o_j$  and compute  $\text{dist}(q, o_j)$ . If  $\text{dist}(q, o_i) < \text{dist}(q, o_j)$ , then  $o_j \notin NN(q, k)$ , otherwise, if  $\text{dist}(q, o_i) = \text{dist}(q, o_j)$ , then  $o_j \in NN(q, k)$ . However, the exact value of  $\text{dist}(q, o_i)$  is obviously not known before the refinement of  $o_i$ .

From Lemma 4 it follows, that the algorithm proposed in [9] is  $R_I$ -optimal but not  $R_{I_u}$ -optimal.

```

algorithm  $k$ NN(QueryObject  $q$ , Integer  $k$ , DBIndex  $I$ )
// Step 1: Initialization
SortedList  $result$ ;
SortedList  $candidates$ ;
initialize  $ranking$  on  $I$  w.r.t. lower bounding distance approximation;
fetch the first  $k$  objects from  $ranking$  and add them to  $candidates$ ;
 $d_{min} = k^{th}$  smallest lower bound of the elements in  $candidates$ ;
 $d_{max} = k^{th}$  smallest upper bound of the elements in  $candidates$ ;
 $d_{f\_next} =$  lower bounding distance of the next element in  $ranking$ ;
do {
  update  $d_{min}$ ,  $d_{max}$ , and  $d_{f\_next}$ ;
  // Step 2: Fetch a candidate
  if  $d_{min} \geq d_{f\_next}$  then
    fetch next object from  $ranking \rightarrow candidates$ ; // only if  $d_{max} \geq d_{f\_next}$ 
    update  $d_{min}$ ,  $d_{max}$ , and  $d_{f\_next}$ ;
  // Step 3: Identify true hits and true drops by using  $d_{min}$  and  $d_{max}$ 
  for all  $c \in candidates$  do
    if  $UB(q, c) < d_{min}$  then add  $c$  to  $result$ ;
    if  $LB(q, c) > d_{max}$  then prune  $c$ ;
  // Step 4: Refine a candidate
  if  $|results| + |candidates| > k \vee d_{f\_next} \leq d_{max}$  then
    for all  $c \in candidates$  with  $LB(q, c) \leq d_{min} \wedge d_{max} \leq UB(q, c)$  do
      if  $dist(q, c) \leq nn_k\text{-dist}(q, result)$  then add  $c$  to  $result$ ;
    else
      add all remaining  $c \in candidates$  to  $result$ ;
  } while ( $d_{f\_next} \leq d_{max} \vee |candidates| > 0$ )
return  $result$ ;

```

**Fig. 5.**  $R_I$ -Optimal  $k$ -NN Algorithm

## 4 $R_{I_{lu}}$ -Optimal Multi-step $k$ NN Search

Based on the above observations, we are able to design an algorithm that is  $R_{I_{lu}}$ -optimal. The pseudo-code of our algorithm is depicted in Figure 5. The algorithm iteratively reduces the candidate set, where in each iteration it identifies true drops, true hits and/or refines a candidate for which the query predicate cannot be determined without the refinement.

The algorithm starts with the initialization of the incremental ranking on the used index according to the lower-bounding distances of all objects. Then, the first  $k$  candidates are fetched from the ranking sequence into the candidate list (Step 1). In order to detect which candidate must be refined, we use two variables  $d_{min}$  and  $d_{max}$  generating a lower-bounding and an upper-bounding distance estimation of the exact  $k$ -NN distance, i.e.  $d_{min} \leq nn_k\text{-dist}(q, \mathcal{D}) \leq d_{max}$ . The basic idea of our algorithm is that we can use this restriction of the exact  $k$ -NN distance in order to identify those candidates  $c$  with  $LB(q, c) \leq nn_k\text{-dist}(q, \mathcal{D}) \leq UB(q, c)$  which must be refined due to Lemma 1. Furthermore, as the stop

criterion of the main loop, we initialize the variable  $d_{f\_next}$  reflecting the lower-bounding distance of the top element of the ranking sequence to  $q$ .

In the main loop, we first update the variables  $d_{min}$ ,  $d_{max}$  and  $d_{f\_next}$  as depicted. Then, we fetch the next candidate  $o$  from the ranking sequence into the candidate set *candidates*, if  $d_{min} \geq d_{f\_next}$  holds (Step 2). This condition guarantees that we fetch only the next candidate from the ranking query if the variable  $d_{min}$  does not guarantee the conservative estimation of the exact  $k$ -NN distance any more. This ensures the  $R_{I_u}$ -optimality of the algorithm and guarantees that our algorithm does not produce unnecessary candidates. Then, the lower-bounding distance estimation of the newly fetched candidate must lie on the new  $d_{min}$  value after the update of the  $d_{min}$  variable. Hence, the fetch candidate either is a true hit or covers the exact  $k$ NN-distance, and thus, must be refined. This guarantees, that our algorithm is optimal w.r.t. the number of fetches from the ranking sequence which in turn is responsible for the optimality according to the number of index accesses. Let us note, that our fetch routine additionally hands over the actual  $d_{max}$  value to the ranking query method. This allows us to proceed the exploration of the index only when necessary and to cut the priority queue according to  $d_{max}$  in order to decrease the size of the queue. After fetching a new candidate, we have to update the variables  $d_{min}$ ,  $d_{max}$  and  $d_{f\_next}$  in order to keep the consistency of the used distance estimation variables.

Step 3 of the algorithm identifies the hits and drops according to the  $d_{min}$  and  $d_{max}$  values. Obviously, all candidates  $c$  with  $UB(q, c) < d_{min}$  can be returned immediately as hits and all candidates  $c'$  with  $LB(q, c') > d_{max}$  can be pruned.

Next, if the number of received results plus the remaining number of candidates are greater than  $k$  and if the condition  $d_{f\_next} \leq d_{max}$  holds, then we refine the next candidate (Step 4). The first condition indicates whether it is still necessary to refine a candidate. The reason for this condition is, that, if the remaining candidates definitely must belong to the query result because there are no concurrent candidates available any more, we can stop the refinement and immediately report the remaining candidates as hits. If both conditions hold, the algorithm refines a candidate  $c$  with  $LB(q, c) \leq d_{min}$  and  $d_{max} \leq UB(q, c)$ . As mentioned above, this procedure guarantees the  $R_{I_u}$ -optimality of this algorithm. We will show later that there must always be a candidate that fulfills the above refinement criterion.

If  $d_{f\_next} > d_{max}$ , i.e. the top element of the ranking sequence can be pruned as true drop or if there are no more candidates left, the main loop stops.

In the following, we show that our algorithm  $R_I$ -Optimal  $k$ NN is (1) fetch optimal in the number of fetches from the ranking sequence, (2) correct, and (3)  $R_{I_u}$ -optimal. Let us note, that fetch optimal corresponds to a minimal number of disk accesses of the underlying index on which the ranking sequence is computed when using access optimal ranking query algorithms (e.g. [10]).

We start with showing that the variables  $d_{min}$  and  $d_{max}$  conservatively and progressively approximate the exact  $k$ NN-distance  $\text{nn}_k\text{-dist}(q, \mathcal{S})$ , where  $\mathcal{S} \subseteq \mathcal{D}$  is the set of candidates in a particular iteration of the algorithm.

**Lemma 2.** *Let  $q$  be a query object and  $\mathcal{S} \subseteq \mathcal{D}$  be the set of candidates in a particular iteration of the algorithm. Then,  $d_{min} \leq nn_k\text{-dist}(q, \mathcal{S}) \leq d_{max}$ .*

*Proof.*  $d_{min}$  is the  $k^{\text{th}}$  lower-bounding distance of objects from  $\mathcal{S}$  to  $q$  and  $d_{max}$  is the  $k^{\text{th}}$  upper-bounding distance of objects from  $\mathcal{S}$  to  $q$ .

First, we show that  $d_{min} \leq nn_k\text{-dist}(q, \mathcal{S})$ . We know that there are at least  $k$  objects  $o \in \mathcal{S}$  with  $\text{dist}(q, o) \leq nn_k\text{-dist}(q, \mathcal{S})$ . Consequently, there must be at least  $k$  objects  $o \in \mathcal{S}$  with  $LB(q, o) \leq nn_k\text{-dist}(q, \mathcal{S})$ , and thus,  $d_{min} \leq nn_k\text{-dist}(q, \mathcal{S})$ .

The second property  $nn_k\text{-dist}(q, \mathcal{S}) \leq d_{max}$  can be shown in a similar way. We know that there are at least  $k$  objects  $o \in \mathcal{S}$  with  $UB(q, o) \leq d_{max}$ . Consequently, there must be at least  $k$  objects  $o \in \mathcal{S}$  with  $\text{dist}(q, o) \leq d_{max}$ , and thus,  $nn_k\text{-dist}(q, \mathcal{S}) \leq d_{max}$ .  $\square$

*Fetch-optimality.* In order to verify that our novel algorithm is fetch optimal, we have to show that the lower-bounding distance estimation  $LB$  of the newly fetched candidate in Step 2 is equal to the new  $d_{min}$  value after the update of the  $d_{min}$  variable.  $d_{min}$  corresponds to the  $k^{\text{th}}$ -smallest lower-bounding distance of the candidates which are already fetched from the ranking sequence. Let  $c$  denote the already fetched candidate for which  $LB(q, c) = d_{min}$  actually holds. We only fetch the next candidate  $c'$  if  $LB(q, c') \leq d_{min}$ . Then, either  $LB(q, c') = d_{min}$  which trivially fulfills the criterion, or  $LB(q, c') < d_{min}$ . In the last case,  $LB(q, c)$  would not be the  $k^{\text{th}}$ -smallest lower-bounding distance estimation anymore, because  $c'$  is an additional already fetched candidate with  $LB(q, c') < LB(q, c)$ . Hence,  $d_{min}$  has to be set to  $LB(q, c')$ . Consequently, as mentioned above, the fetched candidate  $c'$  either is a true hit or certainly covers the  $k$ NN-distance and, thus, must be refined.

*Correctness.* Due to Lemma 2 the candidates  $c$  with  $UB(q, c) \leq d_{min}$  can safely be reported as hits because  $\text{dist}(q, c) \leq UB(q, c) \leq d_{min}$ . Similarly, candidates  $c$  with  $LB(q, c) > d_{max}$  can be safely pruned, since  $\text{dist}(q, c) \geq LB(q, c) \geq d_{max}$ . In summary, our algorithm is correct, i.e. does not produce false hits or false drops.

*$R_{I_u}$ -optimality.* We can prove that our algorithm is  $R_{I_u}$ -optimal by showing that we only refine candidates whose lower- and upper-bounding filter distances cover the exact  $k$ NN-distance. In fact, we only refine those candidates  $c$  with  $LB(q, c) \leq d_{min}$  and  $d_{max} \geq UB(q, c)$ . Thus, according to Lemma 2 the  $R_I$ -optimality is guaranteed. However, this works only if in Step 4 of the algorithm there exists at least one candidate  $c$  with  $LB(q, c) \leq d_{min}$  and  $d_{max} \leq UB(q, c)$ .

**Lemma 3.** *Let  $q$  be the query object and  $\mathcal{S} \subseteq \mathcal{D}$  be a set of candidates for which the lower-bounding and upper-bounding distance estimations ( $LB(q, c)$  and  $UB(q, c)$  for all  $c \in \mathcal{S}$ ) are known. Furthermore, let  $d_{min}$  denote the  $k^{\text{th}}$ -smallest lower-bounding distance estimation and  $d_{max}$  denote the  $k^{\text{th}}$ -smallest upper-bounding distance estimation in  $\mathcal{S}$ . Then, the following statement holds:*

$$\exists o \in \mathcal{S} : LB(q, o) \leq d_{min} \leq d_{max} \leq UB(q, o).$$

*Proof.* Obviously, there must exist at least one candidate  $o \in \mathcal{S}$  with  $d_{min} = LB(q, o)$  and at least one candidate  $p \in \mathcal{S}$  with  $d_{max} = UB(q, p)$ . Let us assume, that the statement in Lemma 3 does not hold, then for all candidates  $c \in \mathcal{S}$  it holds that  $LB(q, c) > LB(q, o) \vee UB(q, c) < UB(q, p)$ , i.e. this also holds for  $o$  and  $p$ . Thus, if  $LB(q, o) < LB(q, p)$  and  $UB(q, o) < UB(q, p)$  it follows that  $o \neq p$ .

As a consequence, for  $k = 1$ , we have  $d_{min} = LB(q, o)$  and  $d_{max} = UB(q, o) \neq UB(q, p)$  which contradicts the assumption about  $d_{max}$ .

Analogously, for  $k > 1$ , there must be at least  $k$  candidates  $c \in \mathcal{S}$  with  $LB(q, c) \leq LB(q, o)$  and there must be at most  $k - 1$  candidates  $c \in \mathcal{S}$  with  $UB(q, c) < LB(q, p)$ . Consequently there must be at least one candidate  $c \in \mathcal{S}$  with  $LB(q, c) \leq LB(q, o)$  and  $UB(q, c) \geq LB(q, p)$ , which contradicts our above assumption.  $\square$

In summary, assuming that a lower- and upper-bounding filter distance is available for each processed object, our novel multi-step  $k$ NN algorithm is correct, requires the minimal number of index page accesses and is optimal w.r.t. the number of refinements required to answer the query.

## 5 Experimental Evaluation

We conducted our experiments on Windows workstations with a 32-bit 3.2 GHz CPU and 4 GB main memory. All evaluated methods were implemented in Java.

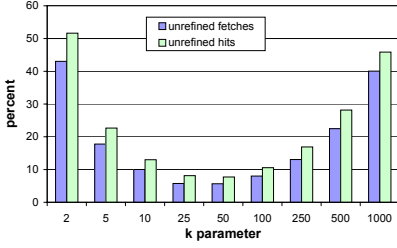
### 5.1 Setup

Our experimental testbed contains four real-world datasets with different characteristics summarized in Table 1. We applied a special form of Lipschitz embedding [11] for the first three datasets using randomly chosen singleton reference sets in order to derive upper and lower bounds. For the timeseries dataset, we generated lower- and upper-bounding distance approximations for the Dynamic Time Warping (DTW) distance as described in [12] where we set the size of the Sakoe-Chiba band width to 10%. Let us note, that for many applications there may exist filter distance measures that yield an even better pruning power in the filter step. We processed 50 randomly selected  $k$ NN queries for the particular dataset and averaged the results.

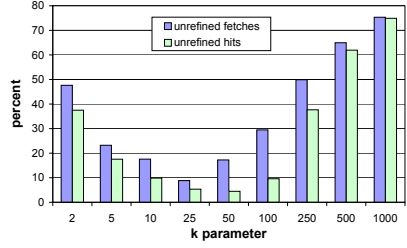
**Table 1.** Summary of real-world test datasets

Dataset	description	# objects	distance	ratio of the cost of filter vs. refinement
San Joaquin	road network	18,263 nodes	Dijkstra	1/300
Protein	protein graph	1,128 proteins	graph kernel	1/2,000
Plane	voxelized 3D CAD	35,950 voxel	Euclidean	1
Timeseries	audio timeseries	2400 clips	DTW	1/150

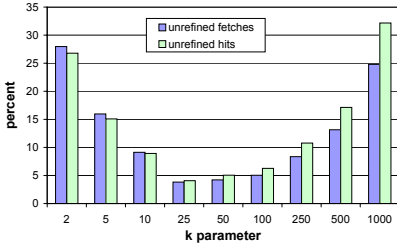




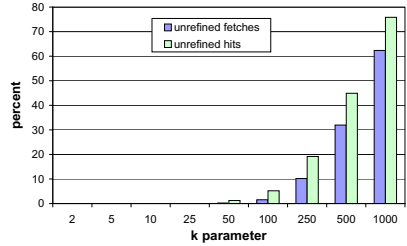
(a) San Joaquin



(b) Protein



(c) Plane

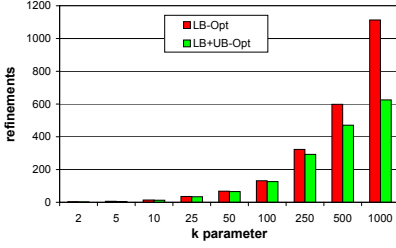


(d) Timeseries

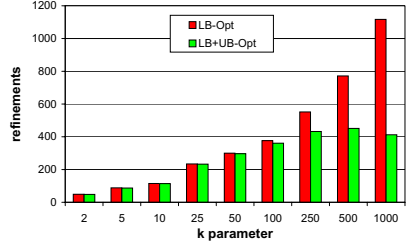
Fig. 6. Relative number of unrefined candidates

## 5.2 $R_{I_u}$ -Optimality vs. $R_I$ -Optimality

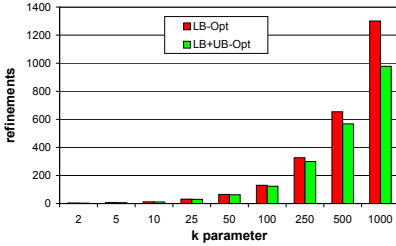
In the first experiment we demonstrate the superiority of our novel  $R_{I_u}$ -optimal algorithm based on lower- and upper-bounding distance estimations over the traditional  $R_I$ -optimal algorithm that uses only lower-bounding distance estimations in the filter step. Figure 6 shows the results of multi-step  $k$ NN queries on our four datasets for different settings of the  $k$  parameter. In particular, the number of hits and the number of fetches (in percent) that both need no refinement are depicted. Note, that an  $R_I$ -optimal algorithm has to refine all fetched candidates, and thus, produces zero unrefined candidates. The results show that, due to the use of an upper-bounding filter distance, a significant amount of the hits does not need to be refined. If we consider that the filter step is 150 (time series), 300 (road network), and 2,000 (proteins) times faster than the refinement step, the runtime improvement is drastic. For three datasets we observe that the amount of unrefined hits and fetches first decreases with increasing  $k$  and then again increases. This is due to the characteristics of the datasets and the used Lipschitz embedding: When increasing  $k$ , the distances first increase very quickly, then stabilize at some point and finally increase again very quickly when  $k$  converges to the number of objects in the dataset. As a consequence, for very low and very high values of  $k$ , the distance approximations produce rather selective stop criteria, whereas for medium values of  $k$ , the pruning power decreases.



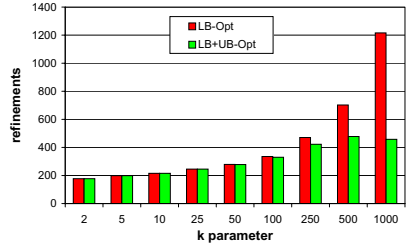
(a) San Joaquin



(b) Protein



(c) Plane



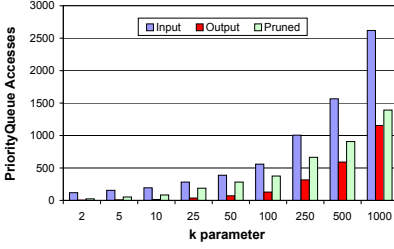
(d) Timeseries

**Fig. 7.** Absolute number of needed refinements of  $R_{I_l}$  and  $R_{I_u}$  approach

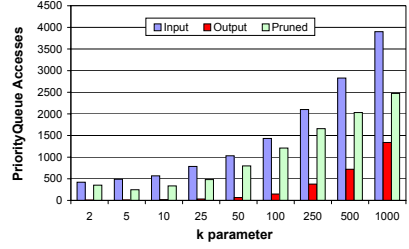
Figure 7 compares the  $R_{I_u}$ -optimal algorithm and an  $R_{I_l}$ -optimal algorithm according to the absolute number of refinement operations. The refinement reduction using the upper-bounding filter distance was clearly improved. In particular, for high  $k$  settings we have to refine only about half of the objects in comparison to competing techniques using only the lower bound filter. For all four datasets, we can also observe that the approximation qualities of the upper bound and the lower bound are rather similar.

### 5.3 Size of the Priority Queue of the Ranking Query

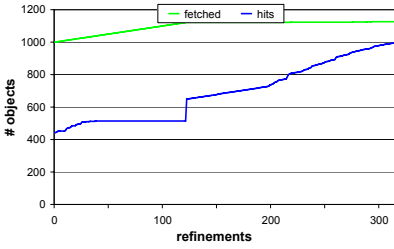
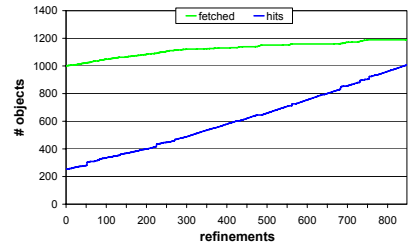
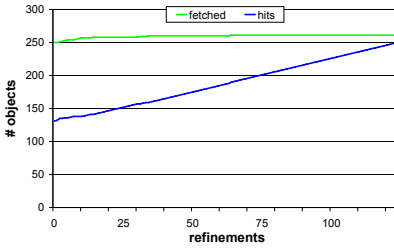
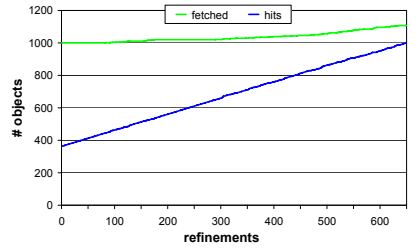
In the next experiment, we examine the influence of the  $d_{max}$  value on the size of the priority queue of the ranking query. Figure 8 depicts the size of *Input*, *Output* and *Pruned*. *Input* denotes the objects that are inserted into the priority queue while traversing the index. *Output* denotes the objects removed from the priority queue while fetching the next candidate. *Pruned* denotes the objects in the priority queue that can be pruned according to the  $d_{max}$  value during the execution of our novel algorithm. It can be observed that we achieve a significant reduction of the priority queue. On the average, we can save more than 50% of memory space when pruning the queue using  $d_{max}$ .



(a) San Joaquin



(b) Plane

**Fig. 8.** Pruning of the priority queue by means of  $d_{max}$ (a) Protein ( $k=1000$ )(b) Plane ( $k=1000$ )(c) San Joaquin ( $k=250$ )(d) San Joaquin ( $k=1000$ )**Fig. 9.** Number of reported results against the number of query iterations

#### 5.4 Early Output of Result Tuples

As mentioned in Section 4, the proposed upper bound filter allows the early output of some true hits even before refinement. Our last experiment evaluates the capability of early outputs for queries with  $k = 1000$  and  $k = 250$ . Figure 9 depicts the number of fetches and true hits detected by our  $R_{I_{tu}}$ -optimal algorithm against the number of refinements. The number of refinements corresponds to the number of iterations in the main loop of our algorithm (cf. Figure 5). It can be observed that already about 45% of the results of the *Protein* dataset can

be reported before starting the refinement of the first object. On that dataset this corresponds to a speed-up of approximately 2,000 for each of these objects. After the 25<sup>th</sup> refinement, we have reported 500 of 1000 results. Similar results can be seen for the experiments on the *San Joaquin* and *Plane* datasets. In summary, a significant portion of the result could be reported very early. Very few refinements are sufficient in order to report more than half of the entire results. Note, that the traditional  $R_{I_1}$ -optimal algorithm proposed in [9] could be adapted such that it would generate the first results before the need to refine the first  $k$  candidates. However, it would be impossible to report more results than there are refined candidates. As we can see in the experiments, our algorithm reports significantly more results than required refinements. Thus, the user can already evaluate the results before the query execution is finished. In an application scenario where the first results are already sufficient for the user, e.g. a doctor wants to confirm his diagnosis drawn from an X-ray image by comparing the actual image to some of the  $k$  most similar images in his database, our algorithm would yield a very high performance gain as very few refinements would be necessary before the user stops the query execution procedure after getting enough intuition about the result.

## 6 Conclusion

In this paper, we generalized the traditional notion of  $R$ -optimality in order to capture the optimality of multi-step  $k$ NN query processing using both lower and upper bounding filter distances. We proposed a novel  $k$ NN multi-step query algorithm and showed that this algorithm is  $R$ -optimal in the generalized sense, correct and fetch-optimal, i.e. requires a minimum number of fetch operations on the underlying ranking algorithm. In our experiments, we demonstrated the superiority of our novel query processing algorithm in comparison to state-of-the-art competitors. In particular, we showed that our approach drastically reduces the number of refinement operations and, thus, the query execution time since the refinement is usually three orders of magnitude slower than the filter step. Our approach features a considerably decreased storage requirement compared to existing solutions and can be used to report first hits as early as possible even before any object has been refined.

## References

1. Guttman, A.: R-Trees: A dynamic index structure for spatial searching. In: Proc. SIGMOD, pp. 47–57 (1984)
2. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R\*-Tree: An efficient and robust access method for points and rectangles. In: Proc. SIGMOD, pp. 322–331 (1990)
3. Ciaccia, P., Patella, M., Zezula, P.: M-Tree: an efficient access method for similarity search in metric spaces. In: Proc. VLDB (1997)

4. Orenstein, J., Manola, F.: Probe spatial data modelling and query processing in an image database application. *IEEE Trans. on Software Engineering* 14(5), 611–629 (1988)
5. Brinkhoff, T., Horn, H., Kriegel, H.P., Schneider, R.: A storage and access architecture for efficient query processing in spatial database systems. In: Abel, D.J., Ooi, B.-C. (eds.) *SSD 1993*. LNCS, vol. 692, Springer, Heidelberg (1993)
6. Agrawal, R., Faloutsos, C., Swami, A.: Efficient similarity search in sequence databases. In: Lomet, D.B. (ed.) *FODO 1993*. LNCS, vol. 730, Springer, Heidelberg (1993)
7. Faloutsos, C., Manolopoulos, M R.a.Y.: Fast subsequence matching in time series database. In: *Proc. SIGMOD* (1994)
8. Korn, F., Sidiropoulos, N., Faloutsos, C., Siegel, E., Protopapas, Z.: Fast nearest neighbor search in medical image databases. In: *Proc. VLDB* (1996)
9. Seidl, T., Kriegel, H.P.: Optimal multi-step k-nearest neighbor search. In: *Proc. SIGMOD* (1998)
10. Hjaltason, G.R., Samet, H.: Ranking in spatial databases. In: Egenhofer, M.J., Herring, J.R. (eds.) *SSD 1995*. LNCS, vol. 951, Springer, Heidelberg (1995)
11. Samet, H.: *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, San Francisco (2006)
12. Keogh, E.: Exact indexing of dynamic time warping. In: *Proc. VLDB* (2002)

# S-GRID: A Versatile Approach to Efficient Query Processing in Spatial Networks

Xuegang Huang, Christian S. Jensen, Hua Lu, and Simonas Šaltenis

Department of Computer Science, Aalborg University  
Fredrik Bajers Vej 7E, DK-9220, Aalborg, Denmark  
{xghuang, csj, luhua, simas}@cs.aau.dk

**Abstract.** Mobile services is emerging as an important application area for spatio-temporal database management technologies. Service users are often constrained to a spatial network, e.g., a road network, through which points of interest, termed data points, are accessible. Queries that implement services will often concern data points of some specific type, e.g., Thai restaurants or art museums. As a result, the relatively few data points are relevant to a query in comparison to the number of network edges, meaning that queries, e.g.,  $k$  nearest-neighbor queries, must access large portions of the network.

Existing query processing techniques pre-compute distances between data points and network vertices for improving the performance. However, pre-computation becomes problematic when the network or data points must be updated, possibly concurrently with the querying; and if the data points are moving, the existing techniques are inapplicable. In addition, multiple pre-computed structures must be maintained—one for each type of data point. We propose a versatile pre-computation approach for spatial network data. This approach uses a grid for pre-computing a simplified network. The above-mentioned shortcomings are avoided by making the pre-computed data independent of the data points. Empirical performance studies show that the structure is competitive with respect to the existing, more specialized techniques.

## 1 Introduction

In step with the emergence of an infrastructure that enables the deployment of mobile services, the database research community has begun to consider the challenges brought on by scenarios where services are delivered to large populations of mobile users. In one important setting, the service users are constrained to a spatial network such as a road network, and the services involve nearest-neighbor queries on points of interest, which we term data points, that are accessible via the network.

As an example, consider Figure 1 where we aim to find the nearest restaurant for a mobile user  $q$  among six restaurants  $R_1, R_2, \dots, R_6$ . To address this type of problem, we model the spatial network as a graph structure. Specifically, a spatial network is modeled as a directed and labeled spatial graph  $RN = (V, E)$ , where  $V = \{v_0, v_1, \dots, v_m\}$  is a finite set of vertices and  $E$  is a finite set of edges. Vertices model intersections and starts and ends of roads, and each vertex has an associated point position in two-dimensional Euclidean space.

An edge models the part of a road in-between two vertices. It is a three-tuple  $e_{i,j} = (v_i, v_j, l)$ , where  $v_i, v_j \in V$  is the start and end vertex of the edge, respectively, and  $l$  captures the travel length of the edge (for simplicity, we assume only bi-directional edges, i.e., edge  $e_{i,j}$  is equivalent to edge  $e_{j,i}$ , but are oppositely directed).

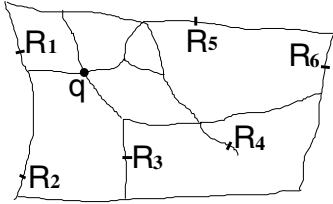


Fig. 1. Example Spatial Network

Two sets of points, termed *query points* and *data points* are also assumed. Each such point  $p$  is either a vertex or a two-tuple  $s = (e_{i,j}, pos)$  consisting of an edge  $e_{i,j}$  and a travel length  $pos$  from  $v_i$  along edge  $e_{i,j}$ .

Next, the graph and data points must be stored on disk using an appropriate data structure (e.g., the CCAM structure [14]). Then a query such as the example query from above is processed by accessing this data structure.

The example spatial network and data points in Figure 1 are represented as shown in Figure 2. To find the nearest neighbor among  $dp_1, \dots, dp_6$  of the query point, which is located at vertex  $v_6$ , a graph search is performed. For example, an algorithm similar to Dijkstra’s algorithm or the A\* algorithm [11] can be used to incrementally search the graph starting at  $v_6$  until the nearest neighbor is found.

To be more specific, the INE search algorithm [12] uses two priority queues,  $Q_v$  for adjacent vertices and  $Q_{dp}$  for data points. Given the query point  $q = v_6$ , adjacent vertices of  $q$  are visited and out into  $Q_v$  (i.e.,  $Q_v = \langle (v_7, 1), (v_5, 2), (v_9, 3), (v_2, 3) \rangle$ ). As no data points are found, the vertex  $v_7$  in  $Q_v$  is dequeued, and its adjacent vertices  $v_3$  and  $v_8$  are inserted, i.e.,  $Q_v = \langle (v_5, 2), (v_3, 2), (v_8, 2), (v_9, 3), (v_2, 3) \rangle$ . Data point  $dp_1$  is found when  $v_5$  is dequeued and its adjacent vertices are accessed ( $Q_{dp} = \langle (dp_1, 3) \rangle$ ). The process continues until the minimum distance from  $q$  to a vertex in  $Q_v$  is no smaller than the distance to the nearest data point (i.e., 3).

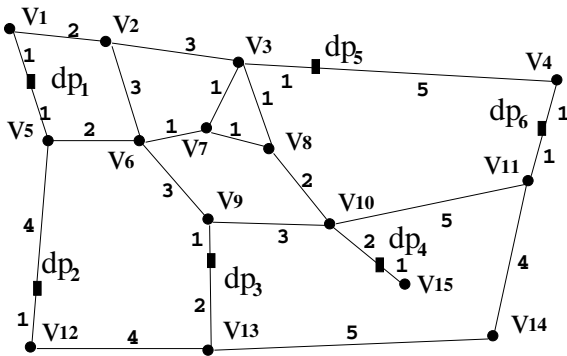


Fig. 2. Example Network

This process, termed *incremental network expansion*, has been used as an ingredient in most existing algorithms for spatial-network queries, including *(continuous) k nearest neighbors ((C)KNN)*, *reverse nearest neighbors (RNN)*, *k closest pairs*, *e-distance join*, and *aggregate nearest neighbors (ANN)*.

Nearest neighbor queries have been studied extensively in settings where the Euclidean

distance is assumed and R-trees are used. The problem of computing KNN queries in spatial network databases has only been addressed more recently. Early work in this direction presented a general framework for supporting NN queries, that included a detailed data model and hierarchical search algorithms [8]. Subsequent work has

considered the optimization of the disk accesses to the network data and data points during query processing.

The INE algorithm and extensions of it have been used for computing range queries, KNN queries,  $k$  closest pair queries, and e-distance joins [12]. This approach works well for dense data points, but it entails excessive accesses to the network data when the data points are sparse. To improve performance, pre-computation techniques have been proposed for primarily KNN and CKNN queries. In the next section, we review three such proposals, namely the VN3 [9], Islands [3], and SPIE [5] approaches.

While these approaches represent significant advances, they also have shortcomings. The data points are assumed to be known to the system prior to any queries. While this assumption will work in some (important) settings, it does not make for a versatile solution. Another limitation is that all data points are assumed to be relevant for all queries, while each query will generally concern only certain types of data points. Next, the approaches do not contend well with updates. Queries have to be “frozen” until the updates (including costly re-pre-computations) have been performed. The query performance then depends on the performance of the updates.

Motivated by these limitations and the need for *versatile* techniques, we propose a novel and more general pre-computation structure, the S-GRID (Scalable Grid), that enables the efficient computation of a broad range of query types in spatial networks. Results of experimental studies show that the S-GRID provides excellent performance in comparison to its competitors.

In summary, the S-GRID approach optimizes the network expansion inherent to many query processing algorithms in spatial networks, while offering the following features.

1. Unlike the existing pre-computation approaches, the S-GRID does pre-computation solely on the network data, and data points are associated with the pre-computed data only at query time.
2. With the existing approaches, updates may cause queries to pause until the updates are complete. With the S-GRID, queries affected by an update are able to expand on the original network data inside the grid cell being updated.
3. While the handling of traffic restrictions such as one-way streets and turn restrictions at intersections is difficult with the existing approaches, the S-GRID encapsulates these into a simple, virtual network so that the query processing can be kept simple and efficient.

The rest of the paper is organized as follows. Section 2 explores previous techniques for efficient KNN query processing in spatial networks. The next section presents the details of the solution. Section 4 empirically compares this solution to existing algorithms. Finally, Section 5 summarizes the paper and suggests research directions.

## 2 Related Work

As mentioned already, network expansion has been used in a range of query processing algorithms, including algorithms for KNN [12, 4], CKNN [1], RNN [18], and ANN [17] queries, as well as for data clustering [16]. We proceed to consider briefly two early



proposals for KNN query processing and then proceed to consider three additional proposals in some detail.

The first proposal is to transform a road network to a high-dimensional Euclidean space in which the traditional KNN search algorithms can be applied [13]. This transformation involves the off-line pre-computation of the network distances between all pairs of vertices, and it uses high-dimensional spatial indexes. As a result, the proposal is of limited interest in our setting.

The next proposal involves two approaches to KNN query processing, namely the INE approach already explained and an approach called IER that exploits Euclidean distances for achieving better performance [12]. To find the KNNs of a query point  $q$ , this approach uses an incremental KNN algorithm to find the nearest neighbors in Euclidean distance. These are then sorted in ascending order of their network distance to  $q$  and the distance of the  $k$ -th neighbor is denoted as  $d_{max}$ . These KNNs and  $d_{max}$  are being maintained while subsequent Euclidean neighbors are retrieved incrementally, until the next Euclidean nearest neighbor has larger Euclidean distance than  $d_{max}$ . It is shown that the INE approach clearly outperforms the IER approach. Further, the IER approach is inapplicable for all notions of distance such as travel time.

Inspired by the use of Voronoi diagrams in Euclidean spaces, the Voronoi-based Network Nearest Neighbor approach (VN3 [9]) generates a Network Voronoi Diagram based on a given set of data points and pre-computes the network distances within each generated Voronoi polygon. Nearest neighbor computations can then utilize the pre-computed distances.

Figure 3 shows the network Voronoi diagram for our example. The Voronoi polygon of data point  $dp_3$  contains four border points:  $b_1, b_2, b_3, b_4$ . The first nearest neighbor of query point  $q$  can be directly found as  $dp_3$  because  $q$  is inside the Voronoi polygon of  $dp_3$ . To find the next nearest neighbors, the data points of the neighboring Voronoi polygons are collected as the candidate set.

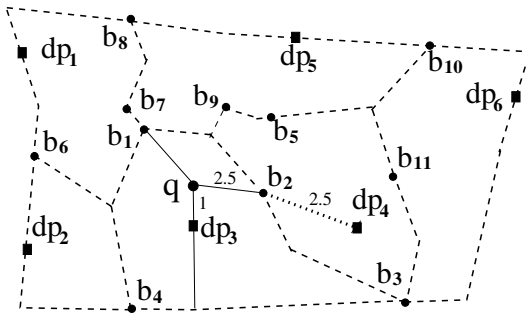


Fig. 3. NN Search with VN3

adjacent polygons. As shown in Figure 3, since the distance from  $q$  to  $b_2$  is 2.5 and the pre-computed distance from  $b_2$  to  $dp_4$  is 2.5, the network distance from  $q$  to  $dp_4$  is 5.

The VN3 approach excels when there are few data points. One limitation is that a Voronoi diagram cannot be generated without knowledge of all the data points. Another is that it does not contend well with queries that concern only data points of certain types. Thus, the approach of generating a Voronoi diagram for each type of data points,

polygons are collected as the candidate set. Then a refinement is applied to find the actual network distance from  $q$  to these data points. Specifically, a network expansion is made to find the network distances from  $q$  to the border points of  $dp_3$ . Then the distance from  $q$  to the neighboring data points can be found by adding the query-to-border distances to the pre-computed border-to-data point distances of the

such as Thai restaurants, museums, and shopping malls, renders it difficult to process “multi-type” queries such as “find the  $k$  nearest Thai and Chinese restaurants and museums” (or “sub-type” queries such as “find the  $k$  nearest modern art museums”). A service that identifies all nearest friends will need a diagram for each unique set of friends.

The Islands approach provides versatility as it enables to control the amount of pre-computation done in preparation for computing KNN queries [3]. First “islands” are pre-computed: starting from each data point, all vertices that are within a given radius  $r_{min}$  to the data point are part of the data point’s island, and the distance to the data point for each such data point is recorded.

A KNN query processing algorithm then makes network expansions from the query point while using the pre-computed “islands” encountered during the expansions.

Using a radius of 3, Figure 4 shows the islands generated for our example. Here, query point  $q$  is inside the island of  $dp_3$ , which is the nearest neighbor of  $q$ . To find subsequent nearest neighbors, a network expansion is made from  $q$ . Since vertex  $v_{10}$  is inside the island of  $dp_4$  and  $v_6$  is inside the islands of  $dp_1$  and  $dp_5$ , we can find their network distance to  $q$  as  $D_N(q, dp_4) = 6$ ,  $D_N(q, dp_1) = 7$ ,  $D_N(q, dp_5) = 7$ . The expansion continues until the distance from  $q$  to the next vertex plus  $r_{min}$  is no smaller than the distance to the  $k$ th nearest neighbor.

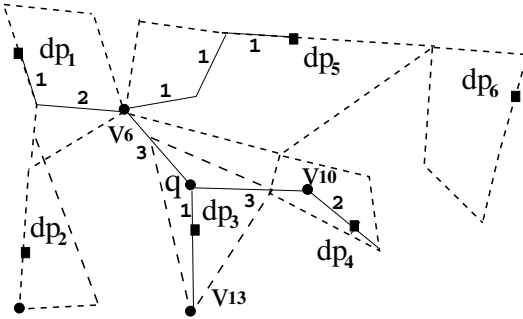


Fig. 4. NN Search with Islands

In the Islands approach, it is possible to control the sizes of the islands (i.e., the radius value  $r_{min}$ ). This offers flexibility in balancing the amount of pre-computation data, the cost of updating the pre-computed data, and the efficiency of the KNN queries. But as for the VN3, the Islands approach requires a pre-knowledge of all the data points. And when updating the pre-computation data in both approaches, all the KNN queries

“covering” the network area that is being updated must wait for the update to complete. Thus, the update performance affects the query performance.

The SPIE approach reduces the network into a set of inter-connected *shortest path trees* (SPTs) [5]. Dijkstra’s algorithm is adapted to grow the SPTs, which are later transformed into SPIEs (Shortest Path tree with horizontal edges and triangular Inequality Edges). Making the assumption that the data points are located on the network vertices, an index is built that stores, for each tree node, the nearest data points in its descendants as well as the distances. The KNN queries are then processed on the SPIEs instead of the real network.

The assumption that the data points are located on network vertices is a limitation in many situations. In practice, data points are located on the edges and are represented using linear referencing [2]. Adding a vertex for each data point on an edge will substantially increase the size of the network. Next, transportation networks often involve

one-way streets, u-turn restrictions, and turn-restrictions at intersections. As a result, although the SPIE approach yields a nice reduction of a network that is a simple undirected graph, the same reduction process does not apply when the constraints and restrictions on edges and vertices are considered.

Previous work on hierarchical structures for path-finding in road networks [7] is not closely related to this paper’s contribution. While that work focuses on shortest-path computations, this paper focuses on KNN and other queries. Also, the S-GRID does not use a hierarchical structure, but a simple 2D grid to organize the pre-computation process and direct the KNN queries.

Summarizing the existing solutions, three basic techniques exist that aim to reduce the cost of expensive network expansions. First, by pre-computing network distances between the vertices of the network and the data points in a specific area of the network, the network expansion covering this area can be avoided by instead looking up the distance values. For example, in the VN3 approach, expansions in each Voronoi cell can be avoided as there is only one data point in each cell and the pre-computed border-to-border and border-to-data distances are used. Second, by linking more data points with each vertex, the query algorithm has knowledge of more candidate nearest neighbors at earlier stages of a network expansion, which restricts the expansion scope. For example, in the Islands approach, the  $k$  nearest neighbors can be found immediately if the query point is inside at least  $k$  islands. Third, by simplifying the network (as in the SPIE approach), queries are processed more efficiently because the expansions are applied on a sparser network.

Common to all of the above approaches is that the pre-computations are data-point dependent: distances to data points are pre-computed (in all three approaches) and the network is subdivided based on the positions of the data points (in the VN3 approach). However, as mentioned in the introduction, this dependence on the data points is often either undesirable or not possible at all. Thus, in contrast to the previous research, we make the fundamental assumption that the data points and their positions are known to the system only at query time. This yields a much more versatile solution. The challenge then becomes one of achieving competitive performance while using only data-point independent pre-computations.

The idea of network partitioning and network connectivity indexing is extensible to other application domains where graphs are queried. For instance, a recent paper [6] introduces a similar divide-and-conquer approach for keyword searches on graphs. The proposed BLINKS approach uses heuristic-based algorithms to partition the graph. In contrast, the S-GRID takes advantage of the spatial embedding of the network and employs a regular spatial grid to partition the network.

### 3 The S-GRID Approach

The S-GRID approach is so named because it employs a 2-dimensional grid for “summarizing” a network and performing pre-computations. In particular, distance pre-computations are made that involve the intersection points between the grid and the network. These grid-based pre-computations usually simplify the network—the grid-network intersection points together with the connections among these points form a simpler,

virtual network. At query time, it is possible to link, in a simple and efficient way, queries and data points to the pre-computations.

By varying the number of cells, the trade-offs between query performance, update performance, the pre-computation costs, and the size of the pre-computation data can be tuned. Although grids have been used for a variety of purposes in many contexts in spatial databases (e.g., in [15]), we believe that our use of a grid in the context of a spatial network database is novel. We proceed with the details of the new approach.

### 3.1 Grid Partitioning and Pre-computation

As described, we apply a 2-dimensional grid to the spatial network. The part of the network inside a grid cell forms one or more mutually disconnected sub-networks. A vertex belongs to a grid cell if its coordinates place it inside the cell. If the two vertices of an edge belong to different cells, we define the midpoint of this edge as a *border point* between the two cells. A vertex with coordinates that intersect with a grid boundary is also a border point. For example, points  $p_1, p_2, \dots, p_7$  in Figure 5(a) are the border points when we apply the shown  $2 \times 2$  grid to the network in Figure 2 (note that  $p_7$  is a vertex while the others are midpoints). We model each grid cell as a three-tuple  $ce = (V, BP, DP)$  where  $V$  is the set of vertices belonging to the cell,  $BP$  is the set of border points of the cell, and  $DP$  is the set of data points inside the cell. Next, if a vertex or a border point is connected to another border point through a sub-network of the cell, the length of the shortest path connecting them in the sub-network is termed their *connected distance* in the cell. For instance, in Figure 5(b), the connected distance between vertex  $v_2$  and border point  $p_2$  in Cell<sub>1</sub> is 4.5.

For each cell, we pre-compute two types of distance values: (a) the connected distance for each pair of connected border points; (b) the connected distance for each pair of a connected vertex and a border point. The spatial network as well as the pre-computation data are stored on disk in the *Vertex-Edge*, *Cell-Border*, and *Vertex-Border* components. The *Vertex-Edge* component corresponds to a similar structure in the INE and Islands approaches [3,12]: adjacency lists of vertices are mapped to disk pages based on the Hilbert values of the vertices.

For our example, page  $pg_1$  in Figure 6 contains the adjacency lists of vertices  $v_1$  and  $v_2$ . Each entry in an adjacency list corresponds to an edge in the graph and contains the

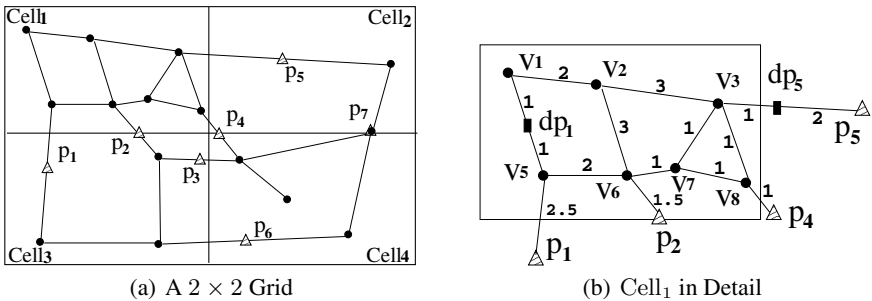


Fig. 5. Example Grid Partition

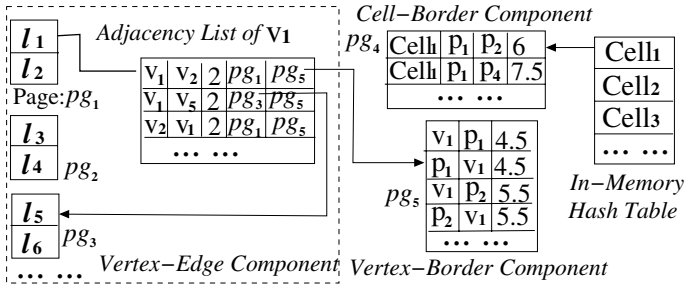


Fig. 6. Example Data Structure

identifications of the start vertex (e.g.,  $v_1$ ) and the end vertex (e.g.,  $v_5$ ), the length of the edge ( $e_{1,5} = 2$ ), a pointer to the disk page ( $pg_3$ ) containing the adjacency list of the end vertex ( $v_5$ ), and a pointer to the disk page ( $pg_5$ ) in the *Vertex-Border* component that stores the connected distances between the start vertex ( $v_1$ ) and the border points of the cell to which this vertex belongs (Cell<sub>1</sub>).

The *Cell-Border* component stores the distances between the border points of the grid cells. For example, in Figure 6 the entries in page  $pg_4$  record the connected distances between border points  $p_1, p_2, p_4, p_5$  of Cell<sub>1</sub> in Figure 5(b). The in-memory hash table links each grid cell to its corresponding disk pages in the *Cell-Border* component. Note that no data points are represented in these data structures.

The computation of connected distances among the border points and vertices in a cell is simple. From each border point, a network expansion is made in the sub-network of the cell following the edges in the reverse direction to find the distances from the reachable vertices. For each vertex discovered in the expansion, the distance to the border point is recorded in the *Vertex-Border* component. The expansion stops when an edge contains another border point; in this case, the connected distance between the two border points is recorded in the *Cell-Border* component.

The resulting pre-computation data structure has a number of features useful when processing queries and performing network updates.

First, with the 2-dimensional grid and the *Vertex-Border* component, it is easy to link the data points with the pre-computation data. We introduce a function **discoverDataPoints**( $DP, ce$ ) that returns all the data points from the set  $DP$  that belong to the grid cell  $ce$ . The function scans the data points in  $DP$ . A data point  $dp = (e, pos)$  belongs to a cell if its nearest vertex on edge  $e$  belongs to this cell. The cell memberships of data points can also be maintained dynamically as data points are inserted, deleted, or updated. Finally, for each of the returned data points  $dp$ , the function returns a set of entries  $(p, dp, d(p, dp))$ , where  $p$  is a border point and  $d(p, dp)$  is the connected distance from  $p$  to  $dp$  in  $ce$ . It is straightforward to compute these distances, since distances from end-vertices of edge  $e$  to border points of the cell can be found in the *Vertex-Border* component.

Second, the border points together with the links among them form a *virtual network*. Unless the grid is very dense, the shortest path connecting two border points in the virtual network has fewer edges than the shortest path connecting the same points in the

underlying network. Note that if no pre-computation is done, to find a data point that is one of the nearest neighbors of a query point, the shortest path between the query point and this data point has to be traversed in the underlying network. If this shortest path traverses border points  $p_1, \dots, p_b$  (where  $b \geq 2$ ), a sub-path connecting border points  $p_i$  and  $p_{i+1}$  is a shortest path between these two points in some cell of the grid, i.e., it corresponds to an edge in the virtual network. Thus, once a network expansion reaches a border point, it can proceed more efficiently in the virtual network. The above-described **discoverDataPoints** function extends the virtual network with data points so that they can be discovered via expansion in the virtual network.

Third, update operations to the pre-computation data are local and data-point independent. Specifically, when a vertex or an edge is added, modified, or deleted from the *Vertex-Edge* component, network expansions from each of the border points of the corresponding grid cell should be made to refresh the distances. Since we only compute the connected distance inside one cell, the cost of update operations is limited to the sub-networks inside this cell. Thus, by varying the number and thus size of the grid cells, the cost of updates can be controlled. In addition, when the network expansions of queries visit cells that are being updated, these expansions can use the underlying network instead of the virtual network in these cells. This way, network update operations do not block querying.

Finally, as mentioned, the number of grid cells can be varied. If the data point density is low, a sparse grid (i.e., with large cells) improves query performance as the expansion process is on a virtual network with fewer vertices. In contrast, a dense grid will have better update performance since the part of the network influenced by an update becomes smaller. In the two extremes, i.e., when there is only one cell in the grid or the grid cells become too small, the approach is not efficient. However, by tuning the cell size, it is possible to achieve improved update and query performance.

We proceed to describe how S-GRID is used to process the KNN query.

### 3.2 KNN Query Processing

We adapt the INE algorithm [12] to compute the KNN query using the S-GRID. Briefly, given a query point  $q$ , a value  $k$ , and a set of data points  $DP$ , we first start a network expansion, termed the *inner expansion*, in the cell where  $q$  is located. Whenever a border point is reached, the *outer expansion* proceeds from that point. The outer expansion is an expansion on the virtual network formed by the border points and their links.

If the cell holds no data points or when the shortest paths to all data points inside the cell have been discovered, the inner expansion is stopped. The *Vertex-Border* component is used to traverse directly from inside a cell and into the virtual network. When the outer expansion visits a border point, the **discoverDataPoints** function is used to find all data points in the cells that share this border point. This process continues until  $k$  nearest neighbors are found.

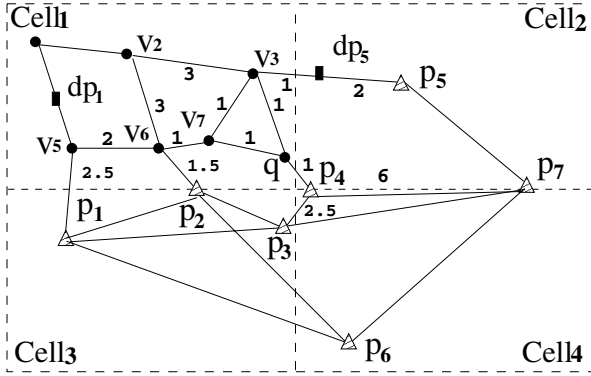
We provide the pseudo code of the KNN algorithm in the following. In addition to the three above-mentioned parameters, the algorithm gets a set of grid cells  $CE$  as a parameter. We use two priority queues,  $Q_{dp}$  and  $Q_v$ , to record, respectively, data points and vertices (or border points) together with their distance to the query point, denoted

as  $d(q, dp)$  and  $d(q, v)$ . Both queues sort elements by the distance value and do not allow duplicate data points or vertices. The size of  $Q_{dp}$  is limited to  $k$  elements.

Both queues have *update* and *dequeue* operations. The *update*( $dp/v, dist$ ) operation inserts a new data point or vertex and the corresponding distance into the queue. If this data point or vertex is already in the queue then, if *dist* is smaller than the distance stored in the queue, the distance value in the queue is updated to *dist*. The *dequeue* operation removes a vertex or a border point with the smallest distance and returns it. Another in-memory list  $L$  caches all the “discovered” data points returned by the **discoverDataPoints** function. Queues  $Q_v$  and  $Q_{dp}$  and list  $L$  are assumed to be empty initially.

- (1) **procedure**  $KNN(q, k, DP, CE)$
- (2) *sort*( $DP, CE$ ) // put  $DP$  into subsets based on cells
- (3) Let the subsets of  $DP$  be  $ce_1.DP, \dots, ce_m.DP$
- (4)  $ce_q \leftarrow findcell(q)$
- (5) **for each**  $dp \in findDP(q.e, ce_q)$ :  $Q_{dp}.update(dp, d(q, dp))$
- (6) **for each**  $v \in \{q.e.v_s, q.e.v_e\}$ :  $Q_v.update(v, d(q, v))$
- (7) **for each**  $bp \in ce_q.BP$ :  $Q_v.update(bp, d(q, bp))$  // *Vertex-Border* is used
- (8)  $Q_{dp} = \langle (dp_1, d(q, dp_1)), \dots, (dp_k, d(q, dp_k)) \rangle$
- (9)  $d_k \leftarrow d(q, dp_k)$  //  $d_k \leftarrow \infty$  if  $dp_k = \emptyset$
- (10)  $v_x \leftarrow Q_v.dequeue$ , mark  $v_x$  as visited
- (11) **while**  $d(q, v_x) < d_k \wedge Q_v \neq \emptyset$
- (12) **if**  $v_x$  is a vertex
- (13) **for each** non-visited adjacent vertex  $v_y$  of  $v_x$
- (14) **for each**  $dp \in findDP(e_{x,y}, ce_q)$
- (15)  $Q_{dp}.update(dp, d(q, v_x) + d(v_x, dp))$
- (16) **if**  $v_y \in ce_q.V$ :  $Q_v.update(v_y, d(q, v_x) + e_{x,y}.l)$
- (17) **if**  $ce_q.DP = \emptyset \vee (ce_q.DP \subset Q_{dp} \wedge \forall dp \in ce_q.DP, d(q, dp) \leq d(q, v_x))$   
// shortest paths found to all  $dp \in ce_q.DP$
- (18) prune each  $(v, d(q, v))$  from  $Q_v$  if  $v$  is a vertex
- (19) **else** //  $v_x$  is a border point; switch to the virtual network
- (20) **for each**  $ce_{ki} \in findcells(v_x, CE)$
- (21) **if**  $ce_{ki} \neq ce_q \wedge |ce_{ki}.DP| > 0 \wedge ce_{ki}$  is undiscovered
- (22)  $L \leftarrow L \cup \mathbf{discoverDataPoints}(DP, ce_{ki})$
- (23) mark  $ce_{ki}$  as discovered
- (24) **for each** non-visited adjacent border point  $v_y \in ce_{ki}$  of  $v_x$
- (25)  $Q_v.update(v_y, d(q, v_x) + d(v_x, v_y))$  // *Cell-Border* is used
- (26) **for each**  $(v_x, dp_{xi}, d(v_x, dp_{xi})) \in L$
- (27)  $Q_{dp}.update(dp_{xi}, d(q, v_x) + d(v_x, dp_{xi}))$
- (28)  $d_k \leftarrow d(q, dp_k)$
- (29)  $v_x \leftarrow Q_v.dequeue$ , mark  $v_x$  as visited
- (30) **return**  $Q_{dp}$

The algorithm first partitions the data points in  $DP$  according to the cells they belong to. Then the network expansion begins with the part of the network inside the cell  $ce_q$  (lines 12–16). The border points of  $ce_q$  are treated as additional vertices of the network. When the shortest paths to all the data points in  $ce_q$  have been computed, the inner expansion on the actual network is completed (lines 17–18), and the algorithm continues only with the outer expansion on the virtual network (lines 19–27). When border points are visited by the outer expansion, the algorithm discovers data points in



(a) 2NN at  $q$

Step	$Q_v$	$Q_{dp}$	$d_k$
1	$(v_7, 1), (v_3, 1), (p_4, 1), (p_2, 3.5), (p_5, 4), (p_1, 6.5)$	$\emptyset$	$\infty$
2	$(v_3, 1), (p_4, 1), (v_6, 2), (p_2, 3.5), (p_5, 4), (p_1, 6.5)$	$\emptyset$	$\infty$
3	$(p_4, 1), (v_6, 2), (p_2, 3.5), (p_5, 4), (v_2, 4), (p_1, 6.5)$	$(dp_5, 2)$	$\infty$
4	$(v_6, 2), (p_2, 3.5), (p_3, 3.5), (p_5, 4), \dots$	$(dp_5, 2), (dp_4, 4)$	4
5	$(p_2, 3.5), (p_3, 3.5), (p_5, 4), \dots$	$(dp_5, 2), (dp_4, 4)$	4
6, 7	$(p_5, 4), \dots$	$(dp_5, 2), (dp_4, 4)$	4

(b) Running Steps

Fig. 7. Example KNN Query

the cells that share the border points (lines 20–23). Note that it is possible for inner and outer expansions to run concurrently, which may happen, for example, if the query point is quite close to the border of a cell. The algorithm guarantees that both expansions will stop if KNNs are found.

The algorithm uses three auxiliary functions. Function  $findDP(e_{x,y}, ce)$  returns the data points in  $ce.DP$  that are located on one of the edges  $e_{x,y}$  and  $e_{y,x}$  and also belong to cell  $ce$ . Function  $findcell(q)$  returns the cell that  $q$  belongs to, i.e., the cell that its nearest vertex, either  $q.e.v_s$  or  $q.e.v_e$ , belongs to. Finally,  $findcells(p, CE)$  returns the cells that have  $p$  as a border point.

To illustrate the working of the algorithm, consider a 2NN query at vertex  $v_8$  in Figure 2. The algorithm first scans the network inside the cell of query point  $q = v_8$ —see Figure 7(a). The border points of the cell are also inserted into  $Q_v$  based on their distance to  $q$ . The steps in computing the query are listed in Figure 7(b). In step 3, since the border point  $p_4$  is closer to  $q$  than other vertices such as  $v_6$ , the algorithm discovers data points in cell<sub>4</sub>, and data point  $dp_4$  is found through  $p_4$ . After data points  $dp_5$  and  $dp_4$  are found, steps 6 and 7 continue and visit adjacent points (in the virtual network) of  $p_2$  and  $p_3$ , and then the algorithm stops.

The KNN algorithm with the S-GRID improves the efficiency of the INE algorithm in three ways. First, the inner expansion is avoided fully or in part, if there are no data points in the cell of the query point or if all these data points have been reached, respectively. Second, by doing the inner and outer expansions concurrently, more data



points are inserted into  $Q_{dp}$  early in the algorithm, which restricts the expansion scope. Finally, the expansion on the virtual network formed by the border points and their links utilizes the pre-computation data to link data points with border points, which makes it possible to find these data points in the virtual network.

In some cases, these optimizations will not take effect. For instance, when there are more than  $k$  data points in the cell of the query point and the query point is close to the center of the cell, all the nearest neighbors needed may be found by the inner expansion, in which case the efficiency of the S-GRID algorithm is equal to that of the INE algorithm. To improve this, the system can maintain several S-GRIDS with different cell sizes and assign a proper S-GRID to run the KNN algorithm based on the location of the query point and the data point density.

### 3.3 Extensions

The S-GRID approach is useful for the computation of many different kinds of queries. To illustrate this, we describe how the S-GRID can be used for computing range and CKNN queries.

**Range Query.** The *range* query retrieves all data points that are within a given network distance  $R$  of a query point. Intuitively, the same network expansion process can be used for the range query as for the KNN query, except that the termination condition in line 11 of the *KNN* algorithm has to be changed to  $d(q, v_x) < R \wedge Q_v \neq \emptyset$ . Note that the S-GRID is used in the same way to maintain the inner and the outer expansions.

**CKNN Query.** The *CKNN* query retrieves  $k$  nearest data points along a given query path, i.e., it finds  $k$  nearest neighbors to any point of a given path in the network. Existing solutions for CKNN query in spatial networks [110] depend on an efficient algorithm for the static KNN query. Specifically, as indicated by the most recent proposal, UNICONS (UNIQUE Continuous Search) [1], to perform a CKNN query on a path  $n_i, n_{i+1}, \dots, n_j$ , it is sufficient to retrieve data points directly on the path and then run a static KNN query at each vertex  $n_k$  on the path ( $i \leq k \leq j$ ) [1, Lemma 2]. To improve the efficiency of such KNN queries, the UNICONS approach pre-computes and stores KNNs of a selected nodes in the network. The S-GRID approach can be used for processing the static KNN queries at the path nodes. Similar to the UNICONS approach, we can also store KNNs of every border point of the S-GRID so that when a KNN query reaches a border point, it can re-use the KNNs of this border point and does not need to expand further from this point.

**Accommodating Traffic Regulations.** The S-GRID approach only requires few modifications in order to be able to contend with traffic regulations when computing KNN queries. Specifically, when one-way roads, streets with u-turn restrictions, and road junctions with turn restrictions have to be considered in the expansion process, only the inner expansion needs to check these constraints. The outer expansion on the virtual network needs not contend with such restrictions, as they have already been addressed during pre-computation.

## 4 Empirical Evaluation

### 4.1 Settings

To gain insight into the performance of the S-GRID, we conduct experiments with two datasets. The first represents the real-world road network and points of interest in Aalborg (AAL), Denmark, and it contains 11,300 vertices, 13,375 bi-directional edges, and 279 data points. The second dataset is the road network data of San Francisco (SF) (downloaded from <http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/>), which contains 175,343 vertices and 223,140 bi-directional edges. For the SF dataset, we use synthetic data points that are generated randomly with a density of 0.1% (the density is the number of data points versus the number of bi-directional edges in the network).

The road network and pre-computation data are arranged into disk pages based on the data structures described in Section 3.1. We set the page size to 4k and use an LRU buffer for caching the disk pages read by the algorithms. The total size of the LRU buffer is 15% of the network data (i.e., the *Vertex-Edge* component). The AAL and SF datasets contain, respectively, 129 and 4,023 pages in the *Vertex-Edge* component.

We compare with the INE and the Islands approaches [3], and consider the performance of these approaches in terms of the CPU running time and the amount of disk I/O operations. All the tested approaches are implemented in C++ and performed on a Pentium IV 1.3 GHz processor with 512 MB of main memory and running Windows 2000. Query points are randomly generated in all the experiments. Each reported performance number is the average number obtained after measuring the performance in several runs of the experiment.

Two series of experiments are conducted. The first series studies the performance of the KNN query comparing the S-GRID, the INE, and the Islands approaches. In these experiments, we vary  $k$ , the density of the data points, and the number (and size) of grid cells. The second series of experiments examines the cost of pre-computation and update operations in the Islands and S-GRID approaches.

### 4.2 Experiments on KNN Query Performance

In this experiment, we examine how the performance of KNN queries is related to the value of  $k$ , the density of data points, the size of the grid cells in the S-GRID approach,

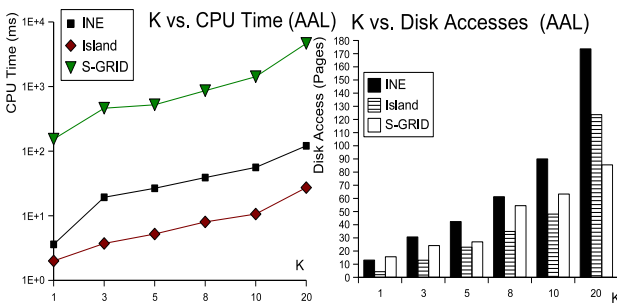


Fig. 8. Effect of K

and the island size in the Islands approach. We set  $k = 5$  for the experiments with the density, the grid cell size, and the island size. The AAL and SF networks use grids of size  $8 \times 8$  and  $20 \times 20$ , respectively.

To express the island size, we define the maximum Euclidean distance

between all vertices in the road network as  $D_{max}$ . The island radius used is then represented as a fraction of  $D_{max}$ . In all experiments, the islands of the same network have the same radius. In the case where the island size is less than the edge length, we set the island to cover the edge of the data point. The AAL network uses an island size of  $0.05D_{max}$  while the SF network uses  $0.01D_{max}$ .

As shown in Figures 8 and 9, the KNN algorithm with the S-GRID requires more CPU time than the INE and Islands approaches. This is because each vertex in the

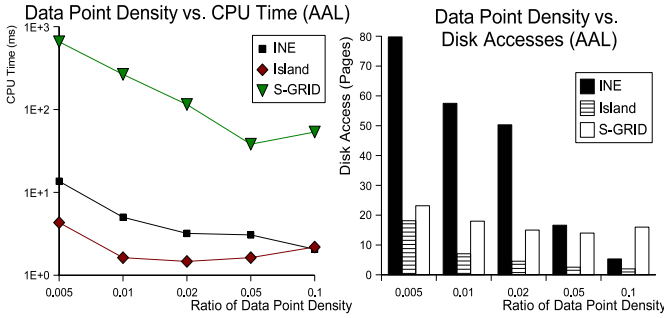


Fig. 9. Effect of Data Point Density

virtual network has more adjacent edges than the original spatial network. This increases the insertion and sorting times in queue  $Q_v$  of the network expansion algorithm. The S-GRID requires fewer disk accesses than INE, but is slightly worse than the Islands approach. The superior performance of the Islands approach (when compared to the S-GRID) is due to the usage of pre-computed distances to the queried data-points in the Islands approach. The slightly lower performance of the S-GRID is the price that is paid for the flexibility of not having to know the set of data points at the time of pre-computation.

Figure 10 reports on experiments where the S-GRID cell size is varied. As expected, the results of the experiments show that when the number of cells increases, the performance of the KNN queries improves. However, as illustrated in Figure 10, when the cells get too small,

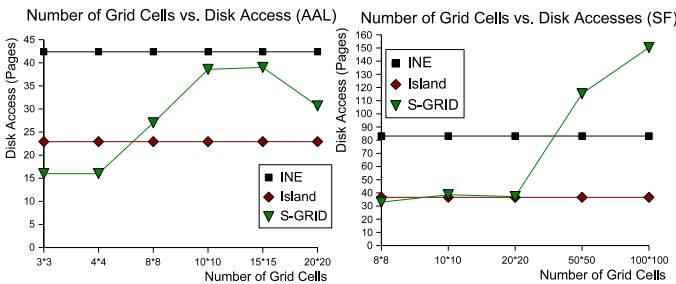


Fig. 10. Effect of Number of Grid Cells

there are too many border points and links, which increase the cost of expansions on the virtual network to the point where it becomes even more expensive than just making expansions on the original network.

### 4.3 Experiments with Pre-computation and Update

With the objective of exploring the cost of pre-computations with the Islands and S-GRID approaches, the second series of experiments measure the disk I/O and number of generated data items (i.e., the amount of border points, links, and pairs of distances)

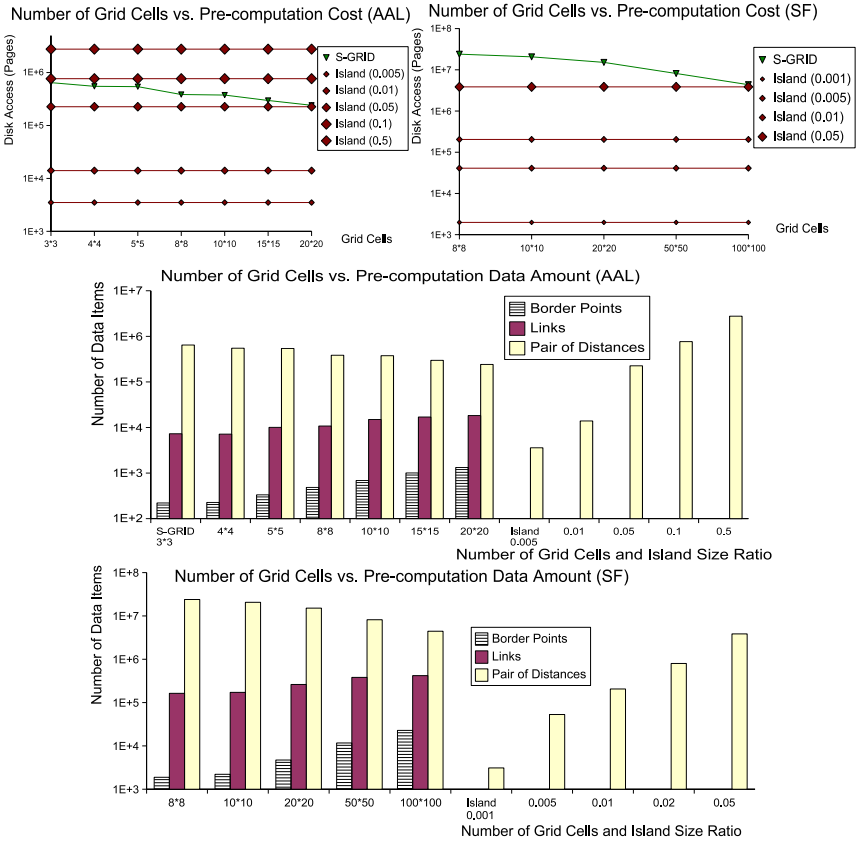


Fig. 11. Pre-computation and Storage Costs—S-GRID and Islands

during pre-computation. We do not report the time for writing the pre-computation data to disk as this time is proportional to the number of generated items. Updates to the network edges and vertices are relatively rare when only the spatial coordinates and the topology of the network are considered. However, for applications where the distance in the road network is measured as the travel time, the frequency of network updates can surpass even the frequency of queries. That is why we also study the network update performance of the compared pre-computation approaches.

The “pair of distances” recorded in Figure 11 shows the number of distances (i.e., the distances from vertices to data points or border points) collected during pre-computation. The figure shows that the pre-computation cost for the S-GRID is higher than the corresponding cost for the Islands approach for small islands, but that it decreases with increasing numbers of grid cells since the network expansion scope from each border point gets smaller.

To illustrate the difference between the original network and the virtual network of the S-GRID, we list the number of vertices and edges (edges  $e_{i,j}, e_{j,i}$  are counted as one) in the original AAL and SF networks as well as the corresponding virtual networks.

As shown in Figure 12, the  $3 \times 3$  and  $8 \times 8$  grids on the AAL network have fewer vertices and edges, which can lead to improved KNN query performance (as in Figure 10). Note that the  $15 \times 15$  grid on the AAL network produces fewer vertices, but more edges, when compared to the original network. Nevertheless, an order of magnitude reduction in the number of vertices (for the  $15 \times 15$  grid) results in improvements of the query performance (compared to the INE in Figure 10). Similar to this, the  $10 \times 10$  and  $20 \times 20$  grids on the SF network improve the query performance. When the grid is too dense, i.e.,  $50 \times 50$  or  $100 \times 100$ , there are too many edges in the virtual network, which negatively effects the efficiency of the query algorithms. The space consumption of the S-GRID, while dependent on the number of grid cells, is generally larger than the space consumption of the Islands approach (see Figure 11). Again, the space is sacrificed for the flexibility of the S-GRID. To discover the appropriate number of grid cells for the specific network and data sets, an iterative approach can be used which, by running a certain amount of test queries over several different grid partitionings, chooses the number of grid cells that results in the most efficient execution of the test queries.

To examine the cost of updates in the S-GRID and the Island approaches, we randomly pick one edge in the AAL network and vary its length so as to collect the CPU and disk I/O costs of the re-computations of pre-computed data. We vary the amount of grid cells and the size of islands and measure the update cost.

Network	Vertices	Edges
AAL	11,300	13,375
AAL ( $3 \times 3$ Grid)	221	7,282
AAL ( $8 \times 8$ Grid)	485	10,774
AAL ( $15 \times 15$ Grid)	1,006	16,986
SF	175,343	223,140
SF ( $10 \times 10$ Grid)	2,213	172,275
SF ( $20 \times 20$ Grid)	4,726	262,187
SF ( $50 \times 50$ Grid)	11,692	381,705
SF ( $100 \times 100$ Grid)	22,822	419,307

Fig. 12. Reduction of Network Size

As illustrated in Figure 13, the update cost in the S-GRID decreases as the cells get smaller since the cost of doing network expansions is smaller for smaller cells. The cost of doing updates on the islands increases dramatically as the islands increase in size. With small islands, it is very likely that an edge is not in any island, in which case the update cost is close to zero (one only needs to update the network data). When the island size increases, each edge is likely associated with more than one island so that the update operation has to re-generate more islands.

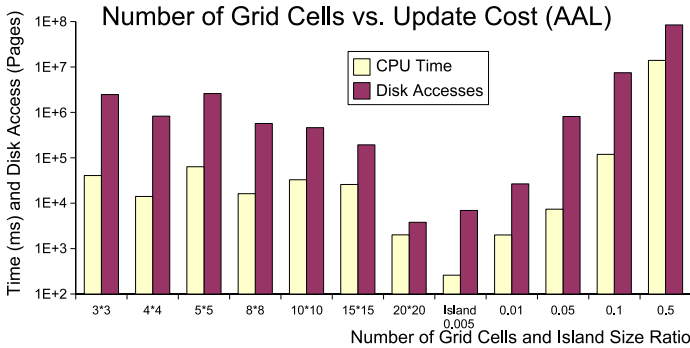


Fig. 13. Update Cost of S-GRID

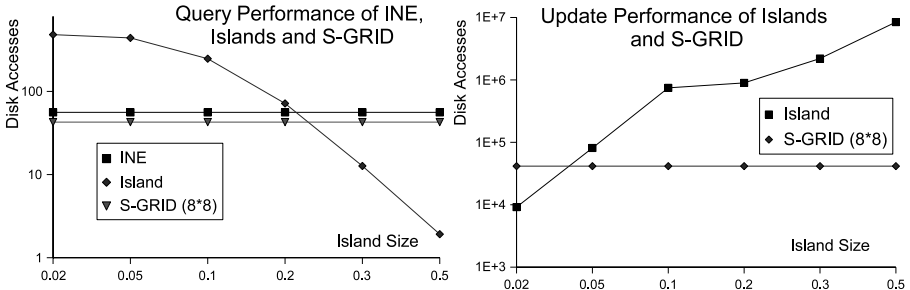


Fig. 14. “Sub-Category” Query and Update

To test how efficiently the S-GRID and other approaches perform “sub-type” KNN queries that search for NNs belonging to a sub-type of data points in the dataset, we randomly divide the 279 data points in AAL network into 10 groups (with 28 or 27 data points in each group) and implement KNN query algorithms that use the INE, S-GRID, and Islands approaches for finding KNNs in one of the groups. The default value of  $k$  is 5 and we use an  $8 \times 8$  grid. We vary the size of the islands.

To support sub-type queries, the Islands approach creates islands for all the data points and the online expansion algorithm checks if a newly-discovered island belongs to the target category, to determine whether further expansion is necessary. In addition, we show the update cost (changing one edge weight) of the Islands and S-GRID approaches. As demonstrated in Figure 14, for the KNN query, the S-GRID approach requires fewer disk accesses than the INE, and the island size has to grow to  $0.3D_{max}$  for the Islands approach to have better performance than the INE and S-GRID. However, the cost of updates in the Islands approach with islands of even smaller size (e.g., 0.05, 0.1, 0.2 of  $D_{max}$ ) is worse than the update cost of S-GRID. Thus, in terms of overall performance of processing sub-type queries and processing updates, the S-GRID is better than the Islands approach. Note that, in the reported experiments, all data points are divided only into 10 “sub-types.” In real applications, the data points may be divided into much more “sub-types,” which further increases the advantage of the S-GRID over the Islands approach.

## 5 Summary and Future Work

Spatial network databases have gained substantial attention with the development of advanced positioning and mobile communication and computing technologies. One current focus is on how to reduce the amount of disk accesses needed for executing spatial and spatio-temporal queries in spatial network databases.

In particular, different approaches to pre-computation has been studied with the purpose of achieving efficient query processing. Motivated by the limitations of existing pre-computation approaches, this paper proposes a more versatile approach, termed the S-GRID, to pre-computation.

In a world where few query processing and indexing techniques proposed by the research community are finding their way into products, and where software vendors tend to prefer versatile and robust techniques over more specialized ones, even though the latter perform factors better, we believe that the S-GRID is significant.

The key new benefit of the S-GRID is that it offers competitive query performance without making the assumption that all data points are known in advance, i.e., before the pre-computation can be accomplished in preparation for the processing. As another benefit, the S-GRID also enables query processing to proceed in parallel with updates to, and the consequent re-pre-computation on, the spatial network. Yet another benefit is that it is easy to integrate support for traffic regulations into the S-GRID approach.

Several directions for future work exist. First, it is of interest to perform analytical cost modeling of the S-GRID and to compare with the existing VN3, Island, and SPIE approaches. Second, a uniform two-dimensional grid has been used in the S-GRID. Since a non-uniform grid can capture more appropriately a network with dense and sparse regions, it is of interest to consider if or how a non-uniform grid can be used with the S-GRID approach. In addition, the partitioning can be made much more “network-aware” [\[6\]](#) in order to reduce the number of boundary points, which in turn may reduce the space consumption and the running time of the S-GRID approach. Third, this paper has hinted at how traffic regulations of real-world road networks can be accommodated in pre-computation. We believe, however, that real-world complexities such as those that stem from traffic regulations should be considered in more detail.

## References

1. Cho, H.J., Chung, C.W.: An Efficient and Scalable Approach to CNN Queries in A Road Network. In: Proc. VLDB, pp. 865–876 (2005)
2. Hage, C., Jensen, C.S., Pedersen, T.B., Speicys, L., Timko, I.: Integrated Data Management for Mobile Services in the Real World. In: Proc. VLDB, pp. 1019–1030 (2003)
3. Huang, X., Jensen, C.S., Šaltenis, S.: The Islands Approach to Nearest Neighbor Querying in Spatial Networks. In: Bauzer Medeiros, C., Egenhofer, M.J., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633, pp. 73–90. Springer, Heidelberg (2005)
4. Huang, X., Jensen, C.S., Šaltenis, S.: Multiple  $k$  Nearest Neighbor Query Processing in Spatial Network Databases. In: Manolopoulos, Y., Pokorný, J., Sellis, T. (eds.) ADBIS 2006. LNCS, vol. 4152, pp. 266–281. Springer, Heidelberg (2006)
5. Hu, H., Lee, D.L., Xu, J.: Fast Nearest Neighbor Search on Road Networks. In: Grust, T., Höpfner, H., Illarramendi, A., Jablonski, S., Mesiti, M., Müller, S., Patranjan, P.-L., Sattler, K.-U., Spiliopoulou, M., Wijsen, J. (eds.) EDBT 2006. LNCS, vol. 4254, pp. 186–203. Springer, Heidelberg (2006)
6. He, H., Wang, H., Yang, J., Yu, P.: BLINKS: Ranked Keyword Searches on Graphs. In: Proc. SIGMOD (to appear, 2007)
7. Jing, N., Huang, Y.W., Rundenstener, E.: Hierarchical Optimization of Optimal Path Finding for Transportation Applications. In: Proc. CIKM., pp. 261–268 (1996)
8. Jensen, C.S, Kolář, J., Pedersen, T.B., Timko, I.: Nearest Neighbor Queries in Road Networks. In: Proc. ACM GIS., pp. 1–8. ACM Press, New York (2003)
9. Kolahdouzan, M., Shahabi, C.: Voronoi-based Nearest Neighbor Search for Spatial Network Databases. In: Proc. VLDB., pp. 840–851 (2004)
10. Kolahdouzan, M., Shahabi, C.: Alternative Solutions for Continuous  $k$  Nearest Neighbor Queries in Spatial Network Databases. *GeoInformatica* 9(4), 321–341 (2005)

11. Pearl, J.: *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison Wesley, Reading (1984)
12. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query Processing in Spatial Network Databases. In: *Proc. VLDB*, pp. 802–813 (2003)
13. Shahabi, C., Kolahdouzan, M., Sharifzadeh, M.: A Road Network Embedding Technique for K-Nearest Neighbor Search in Moving Object Databases. *GeoInformatica* 7(3), 255–273 (2003)
14. Shekhar, S., Liu, D.: CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations. *TKDE* 19(1), 102–119 (1997)
15. Xiong, X., Mokbel, M.F., Aref, W.G.: SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-Temporal Databases. In: *Proc. ICDE*, pp. 643–654 (2005)
16. Yiu, M.L., Mamoulis, N.: Clustering Objects on A Spatial Network. In: *Proc. SIGMOD*, pp. 443–454 (2004)
17. Yiu, M.L., Mamoulis, N., Papadias, D.: Aggregate Nearest Neighbor Queries in Road Networks. *TKDE* 17(6), 820–833 (2005)
18. Yiu, M.L., Papadias, D., Mamoulis, N., Tao, Y.: Reverse Nearest Neighbors in Large Graphs. *TKDE* 18(4), 540–553 (2006)



# Efficiently Mining Regional Outliers in Spatial Data

Richard Frank, Wen Jin, and Martin Ester

School of Computing Science  
Simon Fraser University  
Burnaby B.C., Canada V5A 1S6  
{rfrank,wjin,ester}@cs.sfu.ca

**Abstract.** With the increasing availability of spatial data in many applications, spatial clustering and outlier detection have received a lot of attention in the database and data mining community. As a very prominent method, the spatial scan statistic finds a region that deviates (most) significantly from the entire dataset. In this paper, we introduce the novel problem of mining regional outliers in spatial data. A spatial regional outlier is a rectangular region which contains an outlying object such that the deviation between the non-spatial attribute value of this object and the aggregate value of this attribute over all objects in the region is maximized. Compared to the spatial scan statistic, which targets global outliers, our task aims at local spatial outliers. We introduce two greedy algorithms for mining regional outliers, growing regions by extending them by at least one neighboring object per iteration, choosing the extension which leads to the largest increase of the objective function. Our experimental evaluation on synthetic datasets and a real dataset demonstrates the meaningfulness of this new type of outliers and the greatly superior efficiency of the proposed algorithms.

**Keywords:** Data mining, Spatial outliers, Efficient algorithms, Delaunay-triangulation.

## 1 Introduction

Spatial data is being collected, made available and used more and more both for research and commercial purposes. For the automatic analysis of such data, spatial data mining methods have received a lot of attention in the database and data mining community [21] [27], in particular methods for spatial clustering and outlier detection. Spatial clustering aims at partitioning the spatial region into sub-regions with high intra-region similarity and inter-region difference [13], the goal of spatial outlier detection is to find objects inconsistent with their spatial neighbours even though they may not be significantly different from the entire set of objects [27]. Hotspot analysis is related to both of the above tasks, attempting to discover spatial regions with densities or attribute values that are significantly different from the whole dataset. Important applications are the discovery of disease outbreaks, of crime hotspots or of acts of bio-terrorism.

The goal of the spatial scan statistic [16] is to find such hotspots and various efficient methods for it have been proposed in the literature [10] [20]. Just as

non-spatial outliers can be categorized into global [17] and local [7] outliers, so can spatial outliers. The spatial scan statistic is a global method, targeting a region that deviates (most) significantly from the entire dataset. In many applications however, users are interested in finding local outliers, spatial regions that enclose exceptional knowledge because they contain objects within the region that are outliers relative to the region itself. For example, according to *Tips & Traps when buying a home* [14], if someone wishes to multiply their chances for making money when reselling a home they have to buy an inexpensive house in the most expensive neighbourhood they can afford. As the neighbourhood appreciates over time, the least inexpensive property will appreciate more than its neighbours, relative to its price; the reverse is also true and purchasing an attractive house in a bad neighbourhood will not yield a good investment [30]. By searching for local spatial outliers in property data, it would be possible to find the least expensive properties within the best neighbourhoods, which could become promising investment opportunities.

In this paper we introduce the problem of mining regional outliers in spatial data. A spatial regional outlier is defined as a (rectangular) region with a spatial object in this region such that the value of the objective function, which measures the degree of outlierness of the object within the region, is maximized. For example, in our motivating real estate application, the object would be one (expensive) property, and the rectangle would be the equivalent to its (inexpensive) neighborhood. In a crime dataset, rectangles (neighbourhoods) could be found that have, for example, locations with very different crime rates than the neighbourhood.

A Naïve algorithm enumerating all possible rectangles has a runtime complexity of  $O(n^4)$  which does not scale to large datasets as they appear in many practical applications. Therefore, greedy algorithms are presented which take advantage of the implicit neighbourhood relationships between objects and the different algorithms use different neighbourhood definitions to prune the search-space. In our methods however, the rectangles are "grown" from a seed and at each iteration are extended to include the object which causes the largest increase in the objective function.

The main contributions of our work are as follows:

- 1) We introduce the novel problem of mining regional outliers in spatial data.
- 2) We propose two greedy algorithms for efficiently mining such outliers in large datasets, reducing runtime from  $O(n^4)$  to  $O(n^2)$  compared to the Naïve algorithm.
- 3) Our extensive experimental evaluation on both synthetic and real datasets demonstrates the meaningfulness of spatial regional outliers and the efficiency of the proposed algorithms.

The rest of the paper is organized as follows. In Section 2, we discuss related work. Section 3 introduces our problem definition and presents the corresponding algorithms, including a comparative analysis of these algorithms. In Section 4 we report the results of our experiments on both synthetic and real datasets; Section 5 concludes the paper with a summary and outlook on future work.

## 2 Related Work

There are three parts of the literature related to our study: outlier detection, spatial scan statistics, and the use of Voronoi/Delaunay calculations. We survey them below.

**(Outlier detection)** Outliers can be defined as “data objects that appear inconsistent with respect to the remainder of the database” [5]. Outlier detection methods include *distribution-based* using standard statistical distributions, *depth-based* which map data objects into an  $m$ -dimensional information space and *distance-based approaches* which calculate the proportion of database objects that are a specified distance from a target object [13].

Statistical approaches to outlier detection [5] include distribution-based and depth-based methods. However, the first method has the problem that it must assume the dataset owns some probability distribution even though it is difficult to know the underlying data distribution. The second method is not efficient for high dimensions.

The concept of global distance-based outliers is first introduced in [17], which defines an object  $p$  being an outlier, if at most  $n$  objects are within distance  $d$  of  $p$ . It generalizes the notion of statistical outlier tests, but the running time of the proposed method is still exponential to the number of dimensions. Ramaswamy et al. [25] use the distance of the  $k^{\text{th}}$ -nearest neighbor to rank outliers and give an efficient algorithm for mining top- $n$  global outliers. Bay and Schwabacher [8] improved the method in [17] by using the block nested loop strategy with pruning and randomization techniques. Recently, Tao et al. [29] presented a disk-resident algorithm of finding global outliers with a linear I/O cost. Shekhar et al. [27] studied spatial outliers, which refer to spatially referenced objects whose non-spatial attributes are significantly inconsistent with its neighbors, even though they may not be significantly different from the entire objects. Their neighbourhood relationship however is limited to adjacent locations along directed edges in the underlying spatial graph and not derived from the natural spatial relationships between regions.

Breunig et al. introduced the concept of local outlier, a kind of density-based outlier, which assigns each data a local outlier factor LOF of being an outlier depending on their neighborhood [7]. The outlier factors can be computed very efficiently only if some multi-dimensional index structures such as R-tree and X-tree [6] are employed. A top- $n$  based local outlier mining algorithm which uses distance bound micro-cluster to estimate the density was presented in [15]. Lazarevic and Kumar [18] proposed a local outlier detection algorithm with a technique called “feature bagging”.

Some clustering algorithms like DBSCAN [9] consider identifying outliers, but only to the point of ensuring that they do not interfere with the clustering process. Further, outliers are only by-products of clustering algorithms.

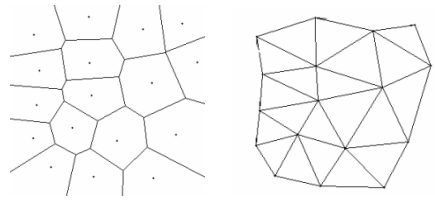
**(Spatial Scan Statistics)** The task of spatial scan statistics, which computes the maximum discrepancy region by scanning a set of circular regions with different radius in the spatial space [16], has received much attention in the data mining community. Authors in [10] proposed a greedy method to find a sub-region  $R$  of the input domain  $S$  for which the mean value of  $R$  is as large as possible. Neill and Moore [20] aimed to find a 2-dimensional rectangular or square region with highest density given an  $n \times n$  grid of rectangles/squares. As an extension work, assuming a uniform, multidimensional grid of bivariate data, where each cell of the grid has a count  $C_i$  and a baseline  $B_i$ , they aim to find spatial regions (d-dimensional rectangles) where the  $C_i$  are significantly higher than expected given  $B_i$  [21]. Agarwal et al. [2] studied the problem of largest discrepancy region in a domain, and present a new exact algorithm, which has the same asymptotic running time as the algorithm of Neill and Moore [20], but with much simpler implementation. Authors in [3] present algorithms

for maximizing statistical discrepancy functions over the space of axis-parallel rectangles with provable approximation guarantees, both additive and relative. Their methods apply to any convex discrepancy function.

**(Voronoi Diagrams)** With spatial datasets, Voronoi diagrams and Delaunay triangulations represent the spatial relationships between the objects [1], [12]. The dataset is partitioned into regions, called Voronoi cells, containing all the points that are closest to the object in the Voronoi cell [19]. For point data, the bisector segments will be the perpendicular bisectors of neighbouring pairs of sites while for spatially extended data, the borders will be circular arcs or arcs of parabolas [19].

Without a loss of usefulness the distance measure could be exchanged for any distance measure, for example, the Manhattan distance or the distance covered by visiting  $k$  shops, as illustrated in [23]. [12] defined the distance measure as ‘furthest-distance’ indicating regions of *least* influence. [24] applied Voronoi diagrams to measure flow in population samples and then model profitability of destination points (i.e.: stores). Voronoi diagrams are also used to describe the internal structure of objects in [19] while [11] uses it for two applications: a) the catchment area of each object and b) denoting the largest polluter of the object in the Voronoi cell.

The dual of the Voronoi diagram, the Delaunay triangulation, is the structure that is the result of connecting all objects with neighbouring Voronoi cells (Fig. 1). [19] defines the Delaunay triangulation of a set of points  $S$  as “a partition of the convex hull of  $S$  into polytopical regions whose vertices are the points in  $S$ . The convex hull of the nearest neighbour set of a Voronoi vertex  $v$  is called the Delaunay cell of  $v$ .”



**Fig. 1.** Voronoi Diagram and corresponding Delaunay triangulation

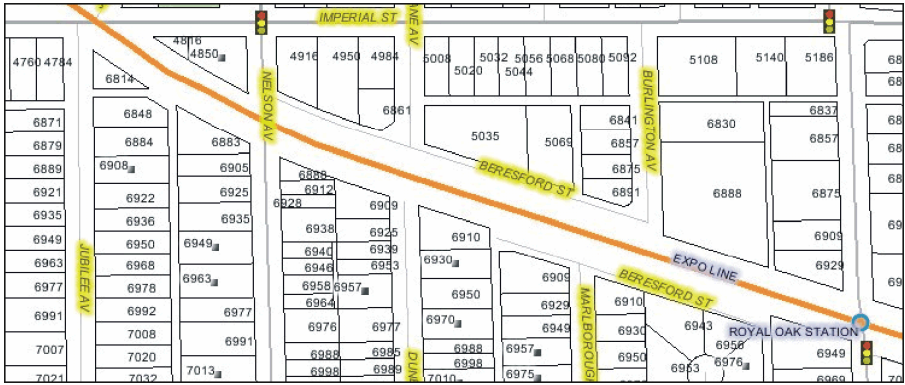
### 3 Mining Regional Outliers in Spatial Data

In this section, we introduce the problem of mining regional outliers in spatial data (Section 3.1). We present a Naïve algorithm that enumerates all possible solutions to search for the best regional outlier (Section 3.2). Section 3.3 proposes a greedy algorithm, Global Neighbourhood Algorithm (GNA), which at each iteration considers adding to the region the object which causes the largest increase in the objective function. Section 3.4 introduces another greedy algorithm, Local Neighbourhood Algorithm (LNA), which prunes the search-space even further by only considering objects which are direct neighbours. The most expensive operation of the greedy algorithms is the calculation of the objective function for each rectangle evaluated, which is efficiently supported by a method of caching (Section 3.5).

#### 3.1 Problem Definition

According to Tobler’s First Law of Geography ‘everything is related to everything else; but that near things are more related than those far apart’ [28]. Hence it is

expected that within the spatial data relationships exist between objects that are near each other, but not necessarily between those that are far apart. The existence of the relationships causes neighbourhoods to be formed within the spatial data. The neighbourhoods can exhibit spatial trends which illustrate the correlation of one or more non-spatial attributes and the distance away from a central object [26]. A spatial outlier is an object which is inconsistent with its spatial neighbours even if the non-spatial values are normal for the rest of the objects of the same class. Due to this, non-spatial outlier detection methods cannot work accurately without somehow taking into account the spatial location [13].



**Fig. 2.** Sample of the BC Assessment dataset

Fig. 2 shows a representation of a small part of the British Columbia Assessment Authority dataset, consisting of properties with spatial attributes (street-address and object polygons) and non-spatial attributes (various values, for example the total value of the property and building). The street-addresses are converted to a longitude/latitude and used for outlier detection. In the case of overlapping or containment between different objects, when the objects are geocoded they could geocode to the same point or different points and a neighbourhood relationship would be established between them. In the following, we formally define the problem.

**Definition 1.** A spatial dataset  $D$  is a set of objects  $P$  with 2 dimensional coordinates  $(X,Y)$  and at least one non-spatial descriptive attribute value  $v$ .

We find rectangular regions, which contain the most deviating outlier given the values of all the objects in the region. This is equivalent to finding, for example, the most expensive, or cheapest, building in a neighbourhood. For each region that is considered, an objective function  $f$  calculates and assigns to the region a value which indicates the degree to which the region is an outlier. This is then maximized across the entire dataset to determine the best region with the largest outlier value. The proposed approach and algorithms work equally well for regions of other shapes, but for cities with grid-like road-networks, rectangular blocks are appropriate. Unlike other methods, such as [7], which can also be applied to spatial datasets, our proposed approach does not rely on any index-structures or database constructs.

**Definition 2.** Given a spatial dataset  $D$ . A region  $R$  is an axis-parallel rectangular area  $R=(P_L, P_U)$ ,  $P_L, P_U \in \mathbb{h}^2$ , where  $P_L$  denotes the lower-left vertex  $(X_L, Y_L)$  and  $P_U$  the upper-right vertex  $(X_U, Y_U)$ . For every edge of  $R$ , there exists an object  $P \in D$  that lies on the edge. More precisely, for each pair of neighbouring vertices  $(X_i, Y_i)$  and  $(X_j, Y_j)$  of  $R$  there is a  $\lambda \in \mathbb{h}$ , such that  $(X_i, Y_i) + \lambda((X_j, Y_j) - (X_i, Y_i)) = P$ ,  $\lambda \in [0, 1]$ . The set of objects in  $R$  is given by  $S_R = \{P \in D, P = (X, Y) \mid X_L \leq X \leq X_U \wedge Y_L \leq Y \leq Y_U\}$ . The complimentary set is given by  $S_{\bar{R}} = D - S_R$ .

**Definition 3.** Given a spatial dataset  $D$  and a region  $R$ . The value of the region under consideration,  $V_R \in \mathbb{h}$ , is the result of applying the objective function  $f$  to  $R$  and all objects  $P \in S_R$ .

The objective function has to compare the range of values within the region against the value of a certain individual object in the same region. This is done by aggregating the attribute values of all objects in the region and comparing the individual against the aggregate value. For example, some alternate objective functions are shown below, where  $v_i$  is the value of object  $i$  which is in the region  $R$ :

- (average of all objects in  $R$ ) – (lowest value of an object):

$$AVG_{i=1}^{|R|}(v_i) - MIN_{i=1}^{|R|}(v_i) \quad (1)$$

- (average of all above-average objects) – (lowest value of an object):

$$AVG_{i=1, v_i > AVG_{i=1}^{|R|}(v_i)}^{|R|}(v_i) - MIN_{i=1}^{|R|}(v_i) \quad (2)$$

- (average of above-average objects) – (average of below-average objects):

$$AVG_{i=1, v_i > AVG_{i=1}^{|R|}(v_i)}^{|R|}(v_i) - AVG_{i=1, v_i < AVG_{i=1}^{|R|}(v_i)}^{|R|}(v_i) \quad (3)$$

- (average of top-k highest) – (average of top-k lowest):

$$AVG_{i=1, i \in TOP(k)}^{|R|}(v_i) - AVG_{i=1, i \in BOTTOM(k)}^{|R|}(v_i) \quad (4)$$

- ABS[(average of all objects in  $R$ ) – (value of most extreme object)]:

$$MAX[AVG_{i=1}^{|R|}(v_i) - MIN_{i=1}^{|R|}(v_i), MAX_{i=1}^{|R|}(v_i) - AVG_{i=1}^{|R|}(v_i)] \quad (5)$$

**Definition 4.** Given a dataset  $D$  and an objective function  $f$ , a Spatial Regional Outlier is the region  $R$  such that its  $V_R$  is maximum over all possible regions. The top-k Spatial Regional Outliers for  $D$  are the top-k regions with the k-highest  $V_R$  values.

As an example, using Fig. 9, the Spatial Regional Outlier is the region enclosing the set of points  $\{34, 92, 46\}$ , it is the region with the highest  $V_R$  out of all possible regions. Although any of the objective functions above could be used in our problem definition, we used function (5) in our experimental evaluation. It is an intuitive and understandable objective function that would find the largest deviation from the average value. The result would be, for example, the least/most expensive property in the most/least expensive neighbourhood. The pseudo-code for the algorithm that calculates  $V_R$  according to  $f$  is shown in Fig. 3.

**INPUT:** region  $R$ , eval function  
**OUTPUT:** value of rectangle

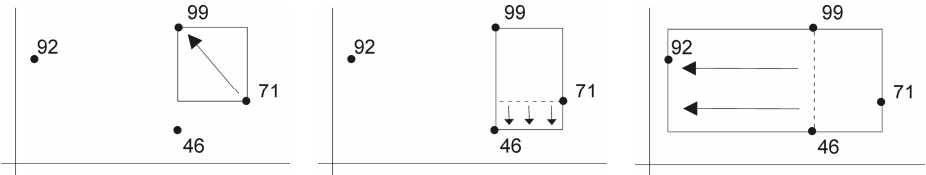
**Method GetRectangleValue( $R, f$ )**  
**1** Get the bin that  $R$  would in  
**2** If  $R$  already in bin  
**3**  $V_R \leftarrow$  retrieve  $R$  from cache  
**4** else  
**5** retrieve objects  $S_R$  in  $R$   
**6**  $V_R \leftarrow$  value of  $S_R$  using  $f$   
**7** add  $\{R, V_R\}$  to cache  
**8** Return  $V_R$

**Fig. 3.** The function that calculates, and caches, the values of each region  $R$

**INPUT:** dataset  $D$ , eval function,  $k$   
**OUTPUT:** TOP- $k$  outlier regions

**Algorithm Naïve( $D, f, k$ )**  
**1**  $V_R = 0, V_{R'} = 0, \text{TOP-}k = \{\}$   
**2** For  $P_1 \in D$   
**3** For  $P_2 \in D$   
**4** For  $P_3 \in D$   
**5** For  $P_4 \in D$   
**6**  $(X_L, Y_L) = (\text{MIN}(X_1, X_2, X_3, X_4), \text{MIN}(Y_1, Y_2, Y_3, Y_4))$   
**7**  $(X_U, Y_U) = (\text{MAX}(X_1, X_2, X_3, X_4), \text{MAX}(Y_1, Y_2, Y_3, Y_4))$   
**8**  $R \leftarrow$  rectangle  $(\{X_L, Y_L\}, \{X_U, Y_U\})$   
**9**  $V_R \leftarrow$  GetRectangleValue( $R, f$ )  
**10** If  $V_R > \text{MIN}(V_R)$  for  $R \in \text{TOP-}k$   
**11** Add  $R$  to TOP- $k$   
**12** Return TOP- $k$

**Fig. 4.** Naïve algorithm. Function  $f$  is a user-defined objective function.



**Fig. 5.** Sample rectangles created by the Naïve algorithm with seed-point:  $\{71\}$

### 3.2 Naïve Algorithm

The Naïve algorithm, Fig. 4, enumerates all possible rectangles defined by at most four objects from the dataset, line 2-5, with one object lying on at least one edge of the rectangle. Assume four objects would make up a rectangle:  $P_1=(X_1, Y_1)$ ,  $P_2=(X_2, Y_2)$ ,  $P_3=(X_3, Y_3)$ ,  $P_4=(X_4, Y_4)$ ; the left bottom vertex of the rectangle can be given by:

$$(X_L, Y_L) = (\text{MIN}(X_1, X_2, X_3, X_4), \text{MIN}(Y_1, Y_2, Y_3, Y_4))$$

and the right upper given by

$$(X_U, Y_U) = (\text{MAX}(X_1, X_2, X_3, X_4), \text{MAX}(Y_1, Y_2, Y_3, Y_4)).$$

The naïve algorithm has a run-time complexity of  $O(n^4)$ , given that there are  $n$  objects in the dataset. This runtime complexity applies both to the worst case and the average case. Fig. 5 shows a sample execution of the naïve algorithm.

### 3.3 Greedy Global Neighbourhood Algorithm

The naïve algorithm is too inefficient for large datasets. We propose to improve the efficiency, although at the expense of guaranteeing optimality, through an iterative greedy algorithm. Using any of the objects as seed, we iteratively grow the rectangle by including one or more object at a time, always choosing the one that leads to the highest increase of the objective function. In the Global Neighbourhood Algorithm (GNA), each object is defined to be a ‘neighbour’ to each other object, hence the entire dataset consists of a ‘global neighbourhood’. At each iteration the current region  $R$  is extended by adding to  $R$  the object which yields the largest increase in  $f$  until there are no objects in  $S_{\bar{R}}$  that increase the value of  $f$ . For the pseudo-code of algorithm GNA see Fig. 6 and Fig. 7.

**INPUT:** dataset  $D$ , eval function,  $k$   
**OUTPUT:** TOP- $k$  outlier regions

**Algorithm GNA( $D, f, k$ )**

```

1 for each seed-object  $O$  in  $D$ 
2  $V_R=0, S_R = \{O\}, \text{TOP-}k=\{\}$ 
3 loop
4  $R \leftarrow$  bounding region for  $S_R$ 
5  $O' \leftarrow$  FindExtensionG( $D, R, f$ )
6  $S_R' \leftarrow$  add  $O'$  to  $S_R$ 
7  $R' \leftarrow$  bounding region for  $S_R'$ 
8  $V_R' \leftarrow$  GetRectangleValue( $R', f$ )
9 If  $V_R' > V_R$  then
10  $V_R = V_R', S_R \leftarrow$  add  $O'$  to  $S_R$ 
11 Else exit
12 If  $V_R > \text{MIN}(V_R)$  for  $R \in \text{TOP-}k$ 
13 Add  $R$  to TOP- $k$ 
14 Return TOP- $k$ 
```

**Fig. 6.** Pseudo-code of the GNA algorithm

**INPUT:**  $D, R, f$   
**OUTPUT:** object with highest  $f$

**Method FindExtensionG( $D, R, f$ )**

```

1  $S_{\bar{R}} \leftarrow$  objects in  $R$ 
2  $S_{\bar{R}} \leftarrow D - S_R$ 
3 for object  $O$  in  $S_{\bar{R}}$ 
4  $R' \leftarrow$  expand  $R$  to include  $O$ 
5  $V_R' \leftarrow$  GetRectangleValue( $R', f$ )
6 if  $V_R > V_R'$  then
7  $V_R = V_R', O' = O$ 
8 Return  $O'$ 
```

**Fig. 7.** Method to find best extension

Given a starting set of objects  $S_R$  in region  $R$ , which in the initial iteration only contains a single object called the seed, the greedy algorithm considers all objects in  $S_{\bar{R}}$  as possible extensions to  $R$  (Fig. 7, line 3). During each iteration, a locally optimal object  $O'$ , yielding the highest  $V_R$  from  $S_{\bar{R}}$ , is selected after which  $O'$  is added to  $S_R$  (Fig. 7 line 5-10). The addition of an object  $O$  causes  $R$  to expand, and since  $R$  is limited to a rectangular shape, it will also add all intermediate objects to  $S_R$  between the original  $R$  and  $O$ . For example, given the shaded region  $R$  in Fig. 8 the extension adding the circled object would create a much larger rectangle which includes all intermediary objects. Hence when calculating the value of the objective function for a potential extension, the entire rectangle needs to be constructed and all objects falling into it considered.

#### Analysis

Algorithm GNA has a worst-case runtime complexity of  $O(n^3)$ , and expected runtime of  $O(n^2)$ . For the worst-case scenario assume that at each iteration a single object is added into  $S_R$ . Given a seed-object, the first iteration will consider all  $(n-1)$  objects in  $S_{\bar{R}}$ , the



second iteration will consider  $(n-2)$  objects, the third iteration  $(n-3)$ , etc, with the last iteration considering a single object. Since each iteration only adds a single object to  $S_R$ , thus there are a total of  $n$  iterations, each considering on average  $\frac{n}{2}$  objects and this is repeated for all  $n$  seed-objects. Thus the worst-case runtime is  $O(n^3)$ .

For the proof of the  $O(n^2)$  expected run-time, we first analyze the runtime complexity per seed object. Assume that with each iteration  $p$  percent of the data are additionally covered. Hence, in the first iteration  $n-1$  objects are considered, of those  $pn$  are added to  $S_R$  and removed from  $S_R$ . With the second iteration  $n-pn-1$  objects are left and considered, with  $pn$  added to  $S_R$ . At the  $i^{\text{th}}$  iteration  $n-(i-1)pn-1$  are considered. Hence the expected number of rectangles,  $T$ , that are considered until iteration  $i$  is:

$$T = (n-1) + (n-pn-1) + (n-2pn-1) + \dots + (n-(i-1)pn-1) = in - i - \sum_{j=1}^i [(j-1)np] = in - i - np \sum_{j=1}^i (j-1)$$

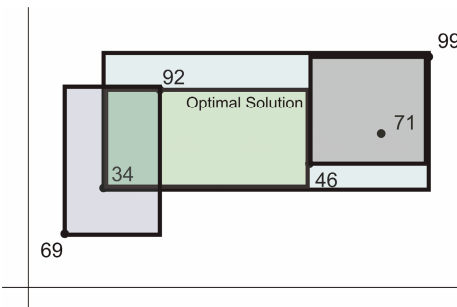
Since at each iteration  $pn$  objects are removed, hence there are at most  $i = \frac{1}{p}$  iterations. Thus,

$$T = \frac{n}{p} - \frac{1}{p} - \frac{n}{p} \sum_{j=1}^i (j-1) = \frac{n}{p} - \frac{1}{p} - \frac{i^2 n}{p} = \frac{n}{p} - \frac{1}{p} - \frac{n}{p^3} \rightarrow O(n)$$

Since there are  $n$  seed-objects, hence the expected runtime of GNA is  $O(n^2)$ .



Fig. 8. Original region and its extension, optimal object for extension is circled



**Optimal solution:**  
 Object-values: {34, 92, 46}  
 $V_R = 34.66$

**Solution of GNA:**  
 Seed-object is object 34  
 {34, 92, 46, 71, 99}  $\rightarrow V_R = 34.4$   
 Seed-object is object 92  
 {92, 34, 69}  $\rightarrow V_R = 31$   
 Seed-object is object 46  
 {46, 71, 99}  $\rightarrow V_R = 27$

Fig. 9. Algorithm GNA cannot guarantee optimality

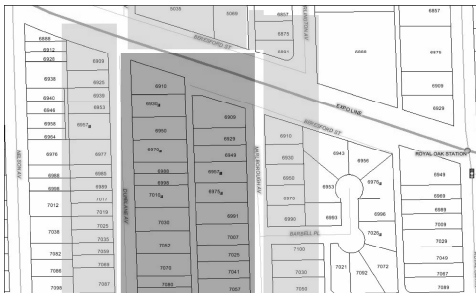
The drawback to the GNA algorithm is that it cannot guarantee finding the optimal solution. This is best illustrated with a counter-example (Fig. 9). In this case, the optimal solution consists of the set of three objects with attribute values  $\{34, 46, 92\}$ . By definition, starting at any object not in this set cannot yield the optimal value. Starting at any of the objects in the set, the greedy algorithm chooses a local-optimal object which happens not to be in the solution-set and hence any further extension can never be optimal.

The reason for the inability of the greedy algorithm to find the optimal solution is because: starting from any object in the optimal solution, two objects have to be added to the region simultaneously in order for the optimal solution to be found. The greedy algorithm is only able to consider adding one object at a time. Adding two objects at a time would require considering pairs of objects at each iteration, yielding an  $O(n^4)$  algorithm. However, even this algorithm would not be able to find rectangles where three objects have to be added simultaneously.

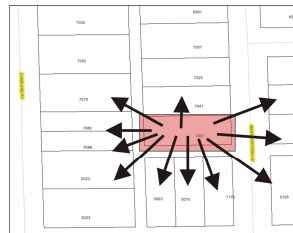
### 3.4 Greedy Local Neighbourhood Algorithm

Algorithm GNA, at every iteration, attempts to extend the region  $R$  by considering all other objects in  $S_{\bar{R}}$ . This means  $R$  could grow arbitrarily large during any given extension. An alternative approach is to limit the number of possible extensions considered in the iterations which would also allow  $R$  to grow at a much more controlled pace. A uniform and consistent way of accomplishing this would be to allow only locally neighbouring objects of  $R$  to be considered for extension. Since each iteration only extends  $R$  locally, hence the greedy Local Neighbourhood Algorithm (LNA) has to consider a smaller number of objects at each iteration.

**Definition 5.** Let the set of all objects be divided into 3 subsets. The objects in  $R$  are still called  $S_R$  with the complimentary set  $S_{\bar{R}}$  now being split into two subsets: those objects neighbouring objects in  $R$  are denoted by  $S_{RN}$ , and those not neighbouring  $R$  are



**Fig. 10.** Region under consideration,  $S_R$  in dark, with its neighbors,  $S_{RN}$  in light-shading.



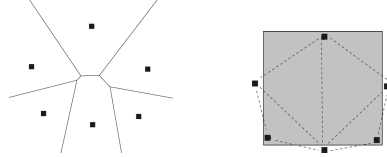
**Fig. 11.** A single object in  $S_R$  is related to many objects in  $S_{RN}$

Everything outside  $S_R$  and  $S_{RN}$  is called  $S_{\bar{RN}}$ .

**INPUT:** region  $R$ , eval function  
**OUTPUT:** object with highest  $f$

**Method FindBestExtensionL( $D, R, f$ )**  
**1**  $S_{RN} \leftarrow$  neighbours to objects in  $R$   
**2** for object  $O$  in  $S_{RN}$   
**3**  $R' \leftarrow$  expand  $R$  to include  $O$   
**4**  $V_R \leftarrow$  GetRectangleValue( $R', f$ )  
**5** if  $V_R > V_{R'}$  then  
**6**  $V_{R'} = V_R, O' = O$   
**7** Return  $O'$  corresponding to  $V_{R'}$

**Fig. 12.** Method to find best extension



**Fig. 13.** Example dataset where LNA will not find the optimal solution (neighbourhood relationships are dashed, optimal solution in grey)

denoted by  $S_{\overline{RN}}$ , i.e.:  $S_{\overline{RN}} = D - S_R - S_{RN}$ . The neighbourhood relationships are established via the dual of the Voronoi diagram, the Delaunay triangulation.

According to [4], a non-random dataset's Voronoi structure has complexity  $O(n)$  in 2 dimensions and more specifically, on average, in a random or non-random dataset there are 6 Voronoi neighbours [22] for each object. Given that multiple objects in  $R$  could share the same neighbour and that some objects on the inside of  $R$  will only have neighbours that are also in  $R$ , hence the number of actual neighbours to  $R$  is much less than  $6 \cdot |R|$ .

Given any region  $R$ , only objects that are direct neighbours to at least one object in  $R$  would be considered as possible extensions. For example, given the original region, shaded dark-grey, in Fig. 10, this approach would only consider the objects shaded in light-grey as further extensions instead of all other objects in the dataset. Each object could have multiple neighbours and the union of neighbours of  $S_R$  is  $S_{\overline{RN}}$  (Fig. 11). The main algorithm is the same as GNA's, Fig. 6, except on line 5 it uses the method FindBestExtensionL (Fig. 12), instead of FindBestExtensionG. FindBestExtensionL determines the next best object to add to the region taking into account the neighbourhood relationships.

**Analysis**

The worst-case runtime for this approach is also  $O(n^3)$  because there are at most  $n$  objects considered at each iteration, and assuming that a single object is added to  $R$  at each iteration, there will be  $n$  iterations. The algorithm also analyzes each of the  $n$  seed-objects independently and hence the worst-case run-time is also  $O(n^3)$ . The expected runtime complexity is similar to the GNA, but since at each iteration only the local neighbourhood is analyzed rather than the global neighbourhood, hence the constant ratio is much smaller.

Optimality is not guaranteed with this approach either, since it prunes the search space even more strictly than GNA. LNA restricts the extension-search to only the

local neighbourhood, but if the optimal solution contains objects which are not connected according to the Delaunay triangulation (i.e.: not direct neighbours), then this approach will not find them. As a counter-example assume the dataset contains 6 objects (Fig. 13) with the optimal solution being disconnected. With this approach, that region could not be discovered since the objects in the region do not share neighbourhood relationships (shown as dashes) and hence would never be considered.

This method might also not find the optimal solution if the data is connected. For example, the 6 data-objects (dashed lines indicate neighbourhood relationships)



will not yield an optimal solution using the LNA. Starting at any of the objects in subset {10-8-4-8} will lead to a local-optimal solution of “10-8-4-8”, while starting with any of the objects in {8-7-0} will lead to a local-optimal solution of “8-7-0”. Although this algorithm theoretically cannot guarantee finding the optimal solution, in practice it often finds target outliers that are very close to the optimal. For a discussion on the optimality of our approaches based on experiments, see Section 4.3.

### 3.5 Rectangle Caching

One of the most expensive operations within the algorithm was the calculation of  $V_R$  for each region  $R$  that had to be evaluated. One solution to this is to keep a cache of all the regions that have been evaluated.

Each region can be described by two pairs of (X,Y) (or longitude/latitude) coordinates, an (X,Y) pair for the lower-left and upper-right corners of the region. Let this region be represented by the 4 numbers  $\{X_1, Y_1, X_2, Y_2\}$ . Since the numbers could be arbitrary, unbound and possibly negative, and not necessarily bound longitude/latitude values, hence guaranteeing a 4D matrix representation is not possible. All 4 coordinates however can be binned by normalizing to an integer value between 1 and 10 after which they can be represented as a 4D matrix or a tree. For example,  $\{X_1, Y_1, X_2, Y_2\}$  could become bin number  $\{5, 4, 6, 8\}$ . There will be multiple  $\{X_1, Y_1, X_2, Y_2\}$  regions that will map into the same bin, but for any given  $\{X_1, Y_1, X_2, Y_2\}$  only the bin it is mapped into would have to be scanned to see if it has already been evaluated. If it exists in the bin then retrieve the value, otherwise evaluate the region and place it into the bin. By modifying the number of bins, it is possible to significantly influence the number of comparisons that must be done, depending on the dataset. This caching is shown in Fig. 3.

## 4 Experimental Evaluation

The experiments were run on both synthetic and real data. The synthetic data was generated using a uniform distribution of  $n$  objects and was used to compare the three approaches in order to evaluate the efficiency of each. This is presented in Section 4.1. In order to test non-uniformly distributed data, and to find neighbourhoods where an individual property is most different from the neighbourhood, experiments (Section 4.2) were also run on the real-life British

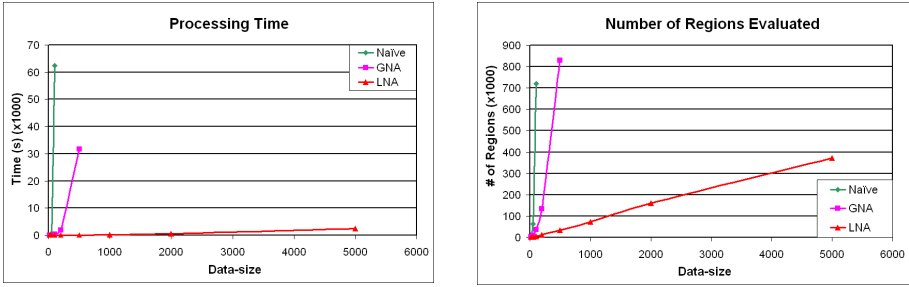


Fig. 14. Synthetic Results: Run-time results and number of regions vs. dataset size

Columbia Assessment Authority (BCAA) dataset. The dataset consists of 667,734 properties, and includes the location and assessed values of each property in BC, Canada.

All experiments were performed on an Intel Core2Duo 6300 @ 2.5GHz with 2GB of RAM. The implementation did not take advantage of multi-threading. We searched for the top 5 outlier regions in each dataset. Our rectangle-cache size was 500 bins for each of the four coordinates describing a region. The neighbourhood relationships were calculated via a call to MatLab, which is treated as a black-box.

### 4.1 Synthetic Datasets

Different size datasets were generated randomly with both uniformly distributed (X,Y) coordinates and descriptive attribute values. The data-size was doubled for each consecutive run. The results are shown in Fig. 14. The runtime for the naive algorithm quickly becomes prohibitive, running a small dataset of 160 objects took 2.5 hours to process and each iteration increased the number of rectangles evaluated as well as runtime by approximately a factor of 16. The runtime for the GNA was much better, but it also quickly became prohibitive as anything above 1000 records already required hours to run. With the GNA, the number of rectangular regions that must be evaluated also increased exponentially. The LNA approach however had a super-linear runtime and was able to process datasets larger than 20,000. This was due to the much smaller neighbourhood it had to evaluate at each iteration.

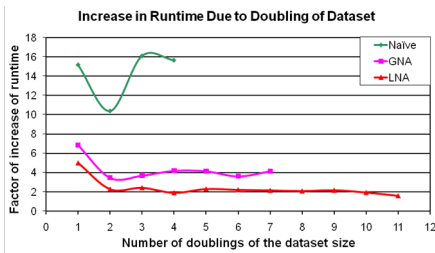


Fig. 15. Factor of increase in runtime as a result of a doubling the dataset

Datasize	City	Type of Property
13	Vancouver	Shopping Centre - Neighbourhood
23	Vancouver	2 Acres Or More - Single Family Dwelling
33	Vancouver	Printing & Publishing Industry
46	Vancouver	Shopping Centre - Regional
70	Vancouver	Multi-Family - Apartment & Row Housing
152	Burnaby	Office Building (Primary Use)
220	Vancouver	Churches & Bible Schools
483	Vancouver	Store(S) And Offices
1462	New West	Single Family Dwelling With Basement Suite
3122	Langley	Single Family Dwelling
4794	North Van	Single Family Dwelling

Fig. 16. Property types and cities used

Since the naïve algorithm become infeasible even with a small dataset, the effect on the run-time as a result of doubling the dataset was investigated and is presented in Fig. 15. ‘Number of Doublings’ of 0 corresponds to a dataset size of 10. It is clear in these results that the naïve complexity approximately increases by a factor of 16 when the dataset is doubled:  $O(n^4)$ . It becomes prohibitive after only 4 doublings (dataset size of 160). The GNA is two factors better at  $O(n^2)$  but it still became prohibitive after 7 doublings (corresponding to a dataset size of 1280). The LNA algorithm was surprisingly much better because a doubling of the dataset led to a straight doubling of the run-time implying that this is actually an  $O(n)$  algorithm; it became prohibitive after 11 doublings (dataset size of 20480).

## 4.2 Real Dataset

The experiments were run on the 2005 dataset of the British Columbia Assessment Authority (BCAA). The dataset consists of the street-address, assessed property and building values of all taxable properties (homes, businesses, etc) within the borders of British Columbia. Each plot was also categorized into 191 types of properties. The original dataset consisted of 667,734 such records. 15,268 of those properties had no specified street-address since they were classified as ‘vacant’ and hence are not assigned addresses. As a preprocessing step, the addresses were converted into a latitude and longitude coordinate value, a process known as geo-coding. This was accomplished with the use of Microsoft MapPoint 2006 (MMP). The locations were determined to within 10 decimal places. After geo-coding was complete, it was found that 27,331 addresses existed in the source data but not on the street-network of MMP, hence were discarded, leaving 625,135 entries for outlier detection. Through the geo-coding process, each address was mapped to a single (X,Y) point. Note that our problem definition requires spatial objects that are points, not polygons.

Selecting a single large dataset and creating samples of different sizes from within that set to evaluate our performance would not at all have yielded correct results. For example, creating equal sized non-overlapping random samples from the set of ‘Stores and Offices’ in Vancouver resulted in value-ranges in one subset of (\$180,400 → \$1,795,000) while another subset had a value-range of (\$186,600 → \$10,977,000) while a third subset had a range of (\$215,300 → \$42,848,000). Performing analysis on these subsets would clearly have been impacted by the sampling.

Considering all types of properties together also would not yield relevant results. For example, Simon Fraser University, with an assessed value of \$468million, would immediately be flagged as an outlier since it is mainly surrounded by residential properties with values between \$100,000 and \$500,000. Hence outlier detection is performed only within each type of property. Different types of properties were extracted from the BCAA dataset to create the data that our experiments would be run on. The criteria for the different datasets used for experimentation is shown in Fig. 16. All algorithms were run on each dataset unless the runtime was infeasible.

As can be seen in Fig. 17, the run-time of both the GNA and LNA is still significantly better than the naïve algorithm, with the LNA significantly outperforming both. Whereas the naïve was only able to process a data-size of 70 properties (of type ‘Multi-Family - Garden Apartment & Row Housing’) in a reasonable time due to its  $O(n^4)$  behaviour, the GNA was able to process 220

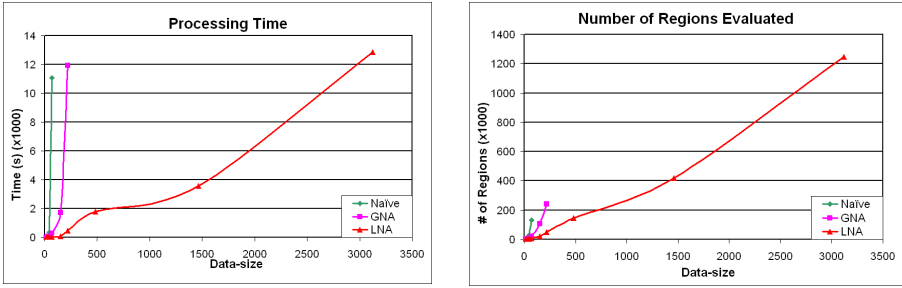


Fig. 17. BCAA Dataset Results: Runtime and number of regions vs. dataset size

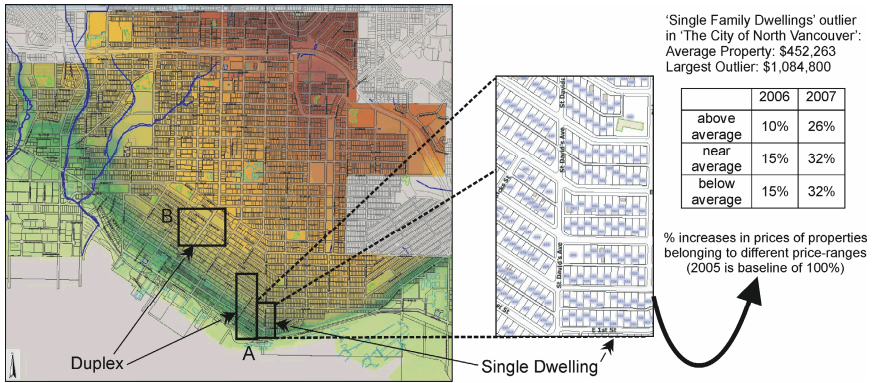


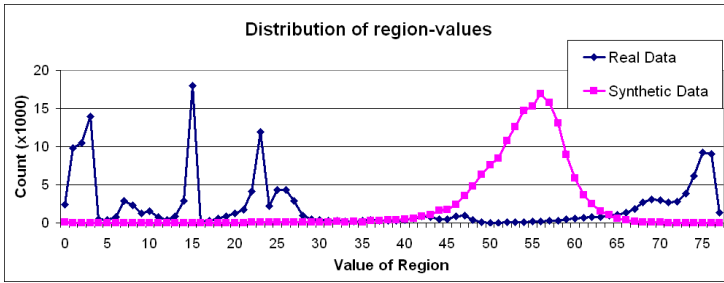
Fig. 18. Three spatial regional outliers in the BCAA dataset

('Churches & Bible Schools') and was roughly  $O(n^3)$ . With LNA however we were able to get results for a data-size of 4794 properties (of type Single Family Dwelling) since it still behaved near-linearly.

The time required to process real data was significantly larger than with uniformly distributed data. This was due to one major difference between the uniformly distributed and real datasets. The real datasets included properties that shared the same address and hence geo-coded to identical coordinates, such as condominiums which could include hundreds of such residences. If all coordinates are unique then the neighbourhood relationship between them is straightforward: there is one neighbourhood relationship. However, between two neighbouring condominiums, let's say each with 100 properties, there will be  $100 \cdot 100 = 10,000$  neighbourhood relationships significantly increasing the runtime since each will be evaluated.

The naïve method could only evaluate 12 regions/second while the LNA was able to evaluate 150 regions/second, hence although the number of regions is small for the naïve it was expensive compared to the LNA. This was due to our use of the cache (section 3.5); with the naïve, no region was evaluated twice whereas lots of duplicate regions were not evaluated with the LNA due to the cache, saving considerable time.

Fig. 18 shows three regional outliers found in the BCAA dataset. The 'Single-Dwelling' region identified had an average property value of \$452,263 but had a single property in it worth over \$1million and was the largest regional outlier



**Fig. 19.** Distribution of  $V_R$  both on real and synthetic data

identified. One of the properties within the 'Duplex' regional outlier (rectangle A, Fig. 18) had a value of \$959,000, more than \$200,000 more than the next most-expensive property in the neighbourhood where the average price was \$587,052. Another outlier region of the same type (rectangle B) had a single property valued at \$552,000 in a region with average values of \$392,337.

The test-dataset only included a single year of data, but by retrieving multi-year property assessment data from the City Of North Vancouver website<sup>1</sup>, it was possible to further investigate. By taking 2005 as the baseline, it was possible to calculate percentage increases in some below, near and above,-average-priced properties for the 'Single-Dwelling' property type. Interestingly, the results indicated that the more expensive properties increased at a lower-rate than, while the below-average properties kept pace with, properties valued close to the average value within the region. This perfectly illustrates the significance and interestingness of our results: that purchasing an inexpensive property in a relatively expensive neighbourhood is a good investment. These results would be useful for anyone interested in purchasing a single-dwelling in North Vancouver.

### 4.3 Optimality

Since the naïve approach exhaustively tests each possible region, it is guaranteed to find the optimal solution. However, in the analysis of our proposed greedy algorithms it was found that neither is able to make the same guarantee. In order to determine how close GNA and LNA are to the true optimal solution, we compared the greedy optimal solutions to all possible region values ( $V_R$ ). The set of all  $V_R$  values for a sample experiment is shown in Fig. 19 as a distribution graph. It illustrates that the bulk of the rectangles are sub-optimal. In our tests, for about half the cases, the greedy approaches were not able to find the global optimal solution, but they did consistently find local optima that were very close to the global optimal. In almost all cases where the global-optimal solution was not found there were less than 10 possible regions, equivalent to less than 0.0008% of all regions, which yielded a more optimal solution than the local-optima found by the greedy algorithm. Had the distribution graph of Fig. 19 followed an exponential curve, the argument also could have been made that the majority of regions are close-to-optimal and a simple random algorithm could outperform GNA or LNA. This however was not the case. These results illustrate that our greedy approaches sacrifice very little in optimality and are much better than the baseline random algorithm.

<sup>1</sup> <http://www.cnv.org/?c=3&i=167>



## 5 Conclusion

With the increasing availability of spatial data in many applications, methods for clustering and outlier detection in spatial data have received a lot of attention in the database and data mining community. In this paper, we have introduced the novel problem of mining regional outliers in spatial data. A spatial regional outlier is defined as a (rectangular) region which contains an outlying object such that the deviation between the non-spatial attribute value of this object and the aggregate value of this attribute over all objects in the region is maximized. In a real estate application, for example, these outliers could represent the least expensive properties within the best neighbourhoods, which could become promising investment opportunities for investors or consumers. We have proposed two greedy algorithms for efficiently mining such outliers in large datasets, reducing the runtime from  $O(n^4)$  to  $O(n^2)$  compared to the Naïve algorithm that enumerates all possible rectangles. Our algorithms grow regions starting from a seed object and extend them by at least one neighboring object per iteration, always choosing the extension which leads to the largest increase of the objective function. An extensive experimental evaluation has been conducted, using synthetic datasets as well as the BC Assessment dataset. Our experimental results demonstrate the meaningfulness of spatial regional outliers. They also show that the proposed greedy algorithms scale much better to large datasets than the Naïve algorithm, while producing results that are close to the optimum solution.

There are several interesting directions for future research. In this paper we have considered only simple rectangular regions. This definition is very generic and widely applicable, but in certain applications other types of regions may be more appropriate. We plan to explore, for example, regions that are taking into account an underlying road-network or system of waterways. In the context of spatio-temporal data, it seems to be promising to investigate temporal aspects as well to find objects that have clearly deviated from their region over time, e.g. properties which have demonstrated a historic trend of outperforming their neighbourhoods.

## References

- [1] Aurenhammer, F.: Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys* 23(3), 345–405 (1991)
- [2] Agarwal, D., McGregor, A., Phillips, J.M., Venkatasubramanian, S., Zhu, Z.: Spatial scan statistics: approximations and performance study. In: *KDD* (2006)
- [3] Agarwal, D., Phillips, J.M., Venkatasubramanian, S.: The hunting of the bump: on maximizing statistical discrepancy. In: *Proc. 17th Ann. ACM-SIAM Symp. on Disc. Alg.* pp. 1137–1146 (2006)
- [4] Bereg, S.: Recent Developments and Open Problems in Voronoi Diagrams. 3rd International Symposium. In: *ISVD '06* (2006)
- [5] Barnett, V., Lewis, T.: *Outliers in Statistical Data*. John Wiley & Sons, Chichester (1994)
- [6] Berchtold, S., Keim, D., Kriegel, H.-P.: The X-tree: An efficient and robust access method for points and rectangles. In: *VLDB* (1996)
- [7] Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: LOF: Identifying Density-Based Local Outliers. In: *SIGMOD 2000* (2000)

- [8] Bay, S.D., Schwabacher, M.: Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In: KDD (2003)
- [9] Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A Density-based Algorithm for Discovering Clusters in Large Spatial Databases. In: KDD (1996)
- [10] Friedman, J.H., Fisher, N.I.: Bump Hunting in High-dimensional Data. *Stat. and Comp.* 9(2), 123–143 (1999)
- [11] Graf, M., Winter, S.: Netzwerk-Voronoi-Diagramme. *Österreichische Zeitschrift für Vermessung und Geoinformation* 91(3), 166–174 (“Network Voronoi Diagrams”, english translation available at [www.sli.unimelb.edu.au/winter/pub.htm](http://www.sli.unimelb.edu.au/winter/pub.htm)) (2003)
- [12] Hoff, K., Culver, T., Keyser, J., Lin, M., Manocha, D.: Fast computation of generalized voronoi diagrams using graphics hardware. In: *Proceedings of ACM SIGGRAPH 1999*, ACM Press, New York (1999)
- [13] Han, J., Kamber, M., Tung, A.K.H.: Spatial clustering methods in data mining: A survey. In: Miller, H., Han, J. (eds.) *Geographic Data Mining and Knowledge Discovery*, Taylor & Francis, Abington (2001)
- [14] Irwin, R.: *Tips and Traps When Buying a Home*, 3rd edn. McGraw-Hill, New York (2003)
- [15] Jin, W., Tung, A.K.H., Han, J.W.: Mining Top-n Local Outliers in Large Databases. In: KDD (2001)
- [16] Kulldorff, M.: A spatial scan statistic. *Comm. in Stat.: Th. and Meth.* 26, 1481–1496 (1997)
- [17] Knorr, E.M., Ng, R.T.: Algorithms for Mining Distance-Based Outliers in Large Datasets. In: VLDB (1998)
- [18] Lazarevic, A., Kumar, V.: Feature Bagging for Outlier Detection. In: KDD (2005)
- [19] Mayya, N., Rajan, V.T.: Voronoi diagrams of polygons: A framework for shape representation. *Journal of Mathematical Imaging and Vision* 6(4), 355–378 (1996)
- [20] Neill, D.B., Moore, A.W.: Rapid Detection of Significant Spatial clusters. In: KDD (2004)
- [21] Neill, D.B., Moore, A.W., Pereira, F., Mitchell, T.: Detecting significant multidimensional spatial clusters. Saul, L.K. et al. (eds.) *Adv. Neur. Info. Proc. Sys* 17, 969–976 (2005)
- [22] Naor, M., Wieder, U.: Novel architectures for P2P applications: the continuous-discrete approach. In: *ACM Symposium on Parallel Algorithms and Architectures*, ACM Press, New York (2003)
- [23] Ohyama, T.: Some Voronoi diagrams that consider consumer behavior analysis. *Industrial Mathematics of the Japan Journal of Industrial and Applied Mathematics* (July 2005)
- [24] Ohyama, T.: Application of the Additively Weighted Voronoi Diagram to Flow Analysis. In: *The 2nd International Symposium on Voronoi Diagrams in Science and Engineering*, Seoul, Korea, (October 2005)
- [25] Ramaswamy, S., Rastogi, R., Shim, K.: Efficient Algorithms for Mining Outliers from Large Data Sets. In: SIGMOD (2000)
- [26] Santos, M.Y., Amaral, L.A.: Geo-spatial data mining in the analysis of a demographic database, *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 9(5) pp. 374–384 (May 2005)
- [27] Shekhar, S., Lu, C.T., Zhang, P.: A unified approach to detection spatial outliers. *GeoInformatica* 7, 139–166 (2003)
- [28] Tobler, W.R.: A computer movie simulating urban growth in the Detroit region. *Economic Geography* 46, 234–240 (1970)
- [29] Tao, Y.F., Xiao, X.K., Zhou, S.G.: Mining Distance-based Outliers from Large Databases in Any Metric Space. In: KDD (2006)
- [30] Weiss, M.B.: *The Everything Homebuying Book - All the Ins and Outs of Making the Biggest Purchase of Your Life* (Paperback). In: Adams Media Corporation, 2nd edn. (February 2003)

# A Two Round Reporting Approach to Energy Efficient Interpolation of Sensor Fields

Brian Harrington<sup>1,\*</sup> and Yan Huang<sup>2,\*\*</sup>

<sup>1</sup> Yahoo! Corporation  
brh@yahoo-inc.com

<sup>2</sup> University of North Texas  
huangyan@cs.unt.edu

**Abstract.** In-network aggregation has been proposed as one of the main mechanisms for reducing messaging cost (and thus energy) in prior sensor network database research. However, aggregated values of a sensor field are of limited use in natural science domains because many phenomena, e.g., temperature and soil moisture, are actually continuous and thus best represented as a continuous surface over the sensor fields. Energy efficient collection of readings from all sensors became a focus in recent research literature. In this paper, we address the problem of interpolating maps from sensor fields.

We propose a spatial autocorrelation aware, energy efficient, and error bounded framework for interpolating maps from sensor fields. Our work is inspired by spatial autocorrelation based interpolation models commonly used in natural science domains, e.g., kriging, and brings together several innovations. We propose a two round reporting framework that utilizes spatial interpolation models to reduce communication costs and enforce error control. The framework employs a simple and low overhead in-network coordination among sensors for selecting reporting sensors so that the coordination overhead does not eclipse the communication savings. We conducted extensive experiments using data from a real-world sensor network deployment and a large Asian temperature dataset to show that the proposed framework significantly reduces messaging costs.

## 1 Introduction

Sensor networks are expected to form a digital nervous system embedded in physical spaces to extend human beings' "tactile" sensations to every corner of the world. In recent years, coin-to-palm sized, programmable sensors have begun to be able to locate their positions, self-organize into a network, and communicate through multi-hop protocols with a gateway which incorporates long-haul communication capacity. This enables the deployments of robust distributed networks of hundreds to thousands of sensors to interact with the physical world.

---

\* The work was completed while at the University of North Texas.

\*\* This work was partially supported by the Texas Advanced Research Program under Grant No. 003594-0010-2006 and by the National Science Foundation under Grant No. OCI-0636421 and Grant No. CNS-0709285.

The feasibility of abstracting a sensor network as a database has been documented and prototyped in pioneer sensor database systems [4,29,22]. In an acquisitional sensor network database, a collection of sensors of the same type may be treated as a table, e.g. *lightSensors*, in a database. The rows of the table are distributed among sensors in a physical space. Each sensor generates records in the format of  $\langle \text{sensorID}, \text{reading}, \text{time} \rangle$ . Users can interact with the network using declarative database query languages. The query is inserted into the network by either broadcasting or targeted routing. For example, users can issue queries such as “*SELECT avg(readings) FROM lightSensors WHERE location IN P EVERY 5 seconds*” where  $P$  is a given polygon.

On the one hand, full duty cycle operations on a sensor node, e.g. Berkeley Mica Motes, will deplete its energy supply in a few days. In-network aggregation [11,25] has been proposed as one of the main mechanisms to reduce messaging cost due to the fact that communication is much more expensive than computation in sensors. For example, the power consumed by a sensor to transmit 1 bit of data is equivalent to 220 - 2,900 instructions on different architectures [29]. On the other hand, the average/sum/max readings of a spatial region is often of limited use to domain scientists. Many phenomena in natural science, e.g., temperature, hydraulic head, soil moisture, and ocean current velocity, are actually continuous, and thus best represented as a continuous surface over the sensor fields. In fact, a raster surface map, e.g. soil moisture map, is frequently created by domain scientists using interpolation upon receiving readings from sensors, and is fundamental for many subsequent field based operations. Energy efficient algorithms that allow domain scientists to interpolate the surface/map of the sensor fields for months to years are critical to future large scale deployments of sensor networks.

In this paper, we address the problem of interpolating maps from sensor fields. The naive way to interpolate the continuous spatial phenomena at the sink is to have all sensors report their readings and perform an interpolation at the sink. This approach depletes the energy of the sensor network very quickly. An alternative way is to utilize spatial autocorrelation to select a subset of sensors to report, and then use them to interpolate a map at the sink. However, for the sensors that do not report, the estimation should be under a user given error bound. Error bounded data collection is sufficient, especially considering that for mapping interpolation residual errors are generally considered common. This problem is also referred to as the “SELECT \*” problem with error bound in recent research publications [5]. In this paper, we focus on utilizing spatial autocorrelation to perform energy efficient and error bounded map interpolation.

One way to utilize spatial autocorrelation is to divide the sensors into groups and let the group leader aggregate/select the readings to report in the group and represent the whole group. Unfortunately, the group leader selection process is non-trivial and usually incurs substantial messaging cost if done dynamically in the sensor field. In many cases, the benefit of grouping can not offset the overhead of dynamic group leader selection. When the group selection is static and once-

for-all, the grouping may not be able to adapt to the dynamic topological changes in the field and frequent sensor failures.

We propose a spatial autocorrelation aware, energy efficient, and yet error bounded framework for interpolating maps from sensor fields. The framework utilizes a simple probabilistic selection process to determine the sensors that need to report in the first round and relies on a second round to control reporting errors for all other sensors. The error bound is achieved in second round by pushing spatial interpolation models used by the sink to sensors in the field allowing the sensors to make decisions on the importance of their readings. The model utilizes qualitative measurements in spatial autocorrelation models, e.g. variograms, to allow a simple, localized, and energy efficient in-network coordination scheme among sensors so that the coordination overhead does not eclipse the communication savings.

We performed extensive experiments using two datasets. One is a real world sensor network deployment from the Intel Berkeley Research Lab [20]. This dataset is small with only 54 sensors. To further evaluate our framework, we evaluated various schemes using a large dataset consisting of thousands of points for 600 months. We compare the proposed model with 6 other simple models for approximate data collection from sensor networks. Our experimental results show that the proposed model provides significant savings.

The rest of the paper is organized as follows. We discuss related work in section 2. In section 3, we first formally define the problem and present an overview of our framework. Then we look into the details of using our framework with kriging as the spatial interpolation method along with a short discussion of other interpolation schemes. An extensive experimental study evaluating our proposed framework is presented in section 4. The paper is concluded with a discussion of possible future extensions of this work in section 5.

## 2 Related Work

We classify related work in the broad area of sensor network databases into four categories: in-network aggregation, correlation based sensor reporting, data compression, and interrogation.

TinyDB [22] is a sensor network database system with a traditional SQL like interface. Due to the resource and communication constrained nature of current sensors, query optimization schemes to reduce the energy consumption are the focus of much research effort [23, 6, 21, 26, 25]. In particular, in-network aggregation is considered an effective way to reduce the messaging cost for aggregation queries (e.g. sum and average) at the cost of simple in-network computations. The rationale is that communication is much more expensive than computation. In the TAG system [21], an aggregation tree is created when a query is broadcast to the sensors. The tree is used to aggregate the sensor readings from children to a parent all the way up to the root where the query originates. For distributive (e.g. sum, min, and max) and algebraic (e.g. average) aggregations, the TAG method significantly reduces the messaging cost by reducing message

hops. Recent work [25] pointed out that aggregation without considering the area that a sensor is representing may not be adequate for spatial aggregations such as average. With aggregation queries, detailed locational information is lost. Our work focuses on representing a field as faithful as possible while reducing communication cost.

Correlation based sensor reporting techniques utilize spatial and/or temporal correlation. Traditional temporal suppression schemes from stream processing that utilize approximate caching [24], time-series models [19], and Kalman Filters [14] have been adapted for mote size sensors. Approximate caching [24] relies on cached values to reduce the number of reports needed from the data stream to the sink. More sophisticated time-series models to capture the temporal trends of the sensor readings have been used in [19,12]. Each sensor node calculates a function based on past readings to predict the readings of the node in the near future and sends it to the sink. In the case of high temporal autocorrelation, a time series is condensed into a single function, thus reducing communication cost. Our work focuses more on utilizing spatial autocorrelations and is orthogonal to models that use temporal autocorrelation. The approach to incorporate temporal models to the framework proposed in this paper will be discussed in the extended version of this paper.

Utilizing spatial autocorrelation has been suggested in prior research work. The clustering based approaches [12,27,13] group sensor nodes according to the spectrum of sensing values or spatial proximity, and then select leaders to represent the group. Election and voting algorithms are important in selecting the representatives for a group of value correlated sensors [12,31]. These algorithms must be distributed and localized in order to scale well for large sensor networks. Energy needs to be budgeted among representative election and communication of selected sensors. Spatial suppression was suggested through clustering sensors into groups and letting a group leader represent the whole group. The challenging problem of dynamic grouping and leader rotation were left for future research in [12].

Snapshot [16] investigated various heuristics for for electing a small set of representative nodes in the network in a localized manner to form a snapshot of the network and provide quick approximate answers to user queries. Unlike the scheme we propose in this paper, the representative selection process in Snapshot can only be performed very infrequently to achieve overall saving. In Ken [5], sensors are partitioned into disjoint cliques with one sensor in each clique selected as the leader of the clique. The leader assumes the duty of selecting a subset of data to be sent back to the sink according to a dynamic probabilistic predication model. The dynamic probabilistic predication model is obtained by a set of training data and is maintained by both the sink and the sensor field. Thus the sink can calculate the expected readings for sensors that do not report using the same predication model as the sensor field. Compared to Ken, our model has a simple probabilistic voting process to select sensors to report in the first round and relies on spatial interpolation models to control errors in the second

round. Our scheme can be extended to incorporate temporal compression and be compared with Ken in future work.

Data compression may be performed spatially or temporally in sensor networks. The information theoretical approaches [8] aim to find an optimal rate to compress redundant information in individual sensor readings. The joint routing and source coding approach [17] attempts to compress redundant information along the routing paths to reduce the number of bits transmitted. For these techniques, the number of transmitted messages are condensed but not reduced. Data compression is orthogonal to and may be applied on top of correlation based data suppression.

Selectively interrogating sensors is another way to avoid requiring all sensors to report. Bash *et al.* proposed an energy efficient uniform sampling scheme for sensor networks [3]. A random sampler from the central station probes the sensor network to select the set of samples. A sensor uses its Voronoi cell to decide a probability to accept or reject the probe. Uniform sampling is useful for application domains such as querying the average sensor battery life. For other application domains such as finite element analysis uniform sampling may not be suitable. Other approaches include the Binocular system [10]. This approach divides sensors into working and sleeping sets and only collects data from the working set. A system model is used to estimate the values for the sleeping set. To avoid error accumulation the system model is updated by having all sensors report at some specified interval. Deshpande et al. [9] proposed a model based probabilistic approach in the central site to answer queries which samples a few sensors when necessary to improve the estimation and achieve a confidence level guarantee. In general the interrogation based approach is “pull based” (compared with a “push based approach” such as Ken [5] and our model). A “pull based approach” does not provide an error guarantee and is insensitive to outliers which are more important for application domains such as environmental monitoring.

### 3 E2K Framework

In this section we will provide a formal definition of the problem and present our proposed E2K (Error Bounded Energy Efficient Kriging) framework. In addition, we discuss how to select an appropriate spatial interpolation model which is very important in our framework.

Once we have the readings from all sensors within some error threshold, the map interpolation problem becomes routine. To obtain a map, simply grid the space using a given spatial resolution, then interpolate using the readings from sensors to determine a value for each cell. So the problem is reduced to the following “SELECT \* FROM sensorType FREQUENCY  $f$  WITHIN  $\epsilon$  FOR  $t$ ”, or more formally:

**Problem Statement:** Let  $S$  be a set of spatially distributed sensors that monitor some attribute  $A$  at a time instance  $t \in T$ , and for  $s \in S$  let  $Z_t(s)$  be the value of  $A$  for sensor  $s$  at time  $t$ . Let  $C$  be a central collection sink that processes

the information received from  $S$ , and let  $Z_t^*(s)$  be the value  $C$  estimates for  $A$  for sensor  $s$  at time  $t$ . Devise an algorithm for  $C$  and the sensors in  $S$ , such that for all  $s, t$  the estimated value is within a user specified error threshold  $\epsilon > 0$  from the actual value, i.e.  $|Z_t(s) - Z_t^*(s)| < \epsilon$ , with the objective of reducing the total messaging cost.

Reducing total messaging cost is chosen as the objective because sending and receiving messages dominate the energy consumption in current sensor networks [5,9,21]. When a message is sent from a sensor, all the neighbors of that sensor will receive the message even though in many cases only a subset of the neighbors are intended destinations. Because sending and receiving have similar energy cost, it is fair to use the total message count sent or relayed from all sensors as the message cost (the actual total message cost that includes receiving and sending messages will be some multiple of the total message cost that we use).

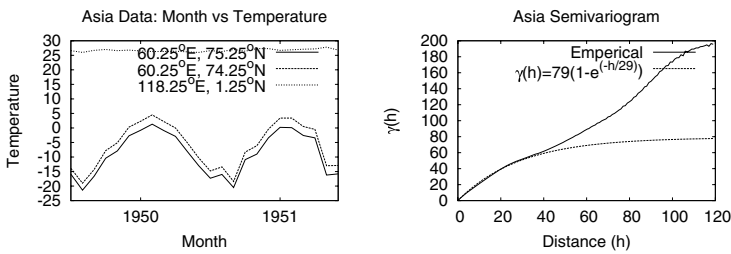


Fig. 1. Sample Spatial Autocorrelation and Empirical and Theoretical Variograms

Tobler’s first law of geography [7] states that in space everything is related to everything else, but nearby things are more related than distant things. For example, the left chart in figure 1 shows temperatures of three locations from Asia for 2 years with a monthly sampling rate [1] (this dataset will be described further in section 4). The location (118.25°E, 1.25°N) is close to the equator and far away from the other two locations. The two close-by locations show very high correlation while the far-away location shows very little or no correlation with the other two.

Various models, e.g. variograms, Moran’s I, and Geary’s C [7], have been developed to quantify this phenomena (formally spatial autocorrelation). Spatial autocorrelation models have been incorporated into spatial interpolation models to create maps in many natural science domains. The spatial autocorrelation based interpolation models create “better maps” in the sense that the sum of the residual errors of the created map is closer to zero.

Our work is mainly inspired by spatial interpolation models utilizing spatial autocorrelation. The main thrust is a two round reporting framework featuring a probabilistic first round reporting and a spatial interpolation model based error control scheme in the second round.



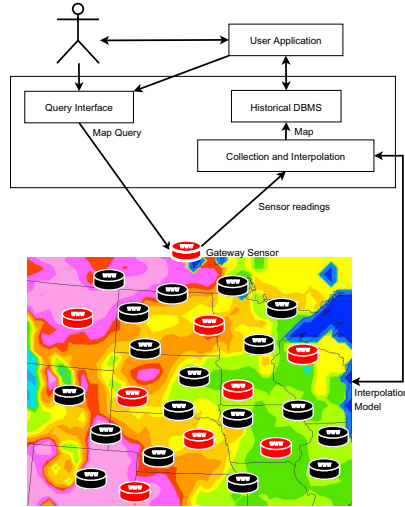


Fig. 2. System Overview

### 3.1 The Two Round Reporting Framework

The system level process is illustrated in Figure 2. Map queries are injected from a user interface into the sensor field. Each sensor makes a local decision about the importance of its reading and decides if it needs to report. As a result only a subset of the sensors (red or gray sensors in Figure 2) will actually report. The reporting sensors route their readings through any multi-hop protocol [30, 15], e.g. GPSR [15], to the gateway sensor or sink. The sink estimates the values for non-reporting sensors, and then interpolates a raster map from the sensor values using the same spatial interpolation model that was used in the field.

The key challenge here is the tradeoff between the complexity of coordination among sensors in dynamically deciding which ones should report, and the savings that result from having fewer sensors report while maintaining an error threshold from all sensors. We need to make sure that the coordination overhead does not eclipse the communication savings. Furthermore, it is desirable to have an error guarantee for each sensor. The three, often conflicting, goals are: (1) minimizing the number of sensors that report and thus save energy; (2) minimizing coordination costs among sensors; and (3) allowing the sink to interpolate readings for non-reporting sensors within an error threshold  $\epsilon$  in the face of possible error propagation due to simple coordination schemes.

A naive probabilistic approach would let each sensor report with a probability  $p$ . This approach does not require coordination. However, there is no error bound for sensors that do not report. An alternative would be to have all sensors send their value to neighbors. Each sensor interpolates its own reading using the readings from its neighbors. If the interpolated value deviates from the real reading by more than  $\epsilon$ , then the sensor reports. This approach again does not

have error guarantee for non-reporting sensors because of concurrent decision making and error propagation in sensor fields. We propose a two round reporting framework called E2K that is both energy efficient and has an error guarantee. E2K consists of algorithm 1 for individual sensors and algorithm 2 for the central site.

---

**Algorithm 1.** Sensor ( $s_0$ ) Algorithm
 

---

```

1:  $Z_t(s_0) \leftarrow$  value of  $A$  for this sensor at time  $t$ .
2:  $rand \leftarrow$  a random number  $\in [0, 1]$ 
3: if ( $rand < p$ ) then
4:   {Round 1}
5:   Report ( $Z_t(s_0), round_1$ ) to the central site and neighbors within distance  $r$ .
6: else
7:   {Round 2}
8:    $R \leftarrow$  the set of readings from sensors within distance  $r$  that reported in first
      round.
9:    $Z_t^*(s_0) \leftarrow interp(R)$ 
10:  if ( $|Z_t^*(s_0) - Z_t(s_0)| \geq \epsilon$ ) then
11:    Report ( $Z_t(s_0), round_2$ ) to the central site.
12:  end if
13: end if

```

---

The algorithm for sensors is divided into two rounds. In the first round a sensor decides to report with a probability  $p$ . As a result, a set of representative sensors are selected to temporarily represent the field. Reporting sensors route their readings to the sink and also send their readings to all sensors within distance  $r$  for use in the second round. The second round is needed because the first round does not have error bounds for non-reporting sensors. In the second round, a non-reporting sensor in the first round will **interpolate** its reading assuming it does not report, using only the readings received from reporting sensors in the first round. If the estimated value deviates from the real value by more than  $\epsilon$ , then it reports.

Any kind of interpolation method can be used in the second round, e.g., a simple average. However, a better interpolation method results in fewer sensors that need to report in the second round. Basically, the probabilistic first round provides a reasonable number of sensors to report so that the estimation methods will work well in the second round, and the better the interpolation method is the better the overall result will be. We will revisit spatial interpolation methods in section [3.2](#).

The algorithm for the central site is meant to mirror what is done by the individual sensors. If a reading is sent by a sensor, then that reading will be used. For sensors where no reading was received, then the reading will be estimated using the same method, and same set of neighbors, that the sensor used to determine whether to report in the second round which gives us an error bound for each sensor location of  $\epsilon$ . Note that we assume a reliable (or at least best

---

**Algorithm 2.** Central Site Algorithm

---

```

1: Let  $S$  be the set of all sensors.
2:  $R_1 \leftarrow$  the set of values received from sensors for attribute  $A$  at time  $t$  in round 1.
3:  $R_2 \leftarrow$  the set of values received from sensors for attribute  $A$  at time  $t$  in round 2.
4: for all  $s \in S$  do
5:   if ( $s$  reported a value) then
6:      $Z_t^*(s) \leftarrow$  value of  $s$  in  $R_1 \cup R_2$ 
7:   else
8:      $R_n \leftarrow$  the subset of  $R_1$  within distance  $r$  of  $s$ .
9:      $Z_t^*(s) \leftarrow \text{interp}(R_n)$ 
10:  end if
11: end for

```

---

effort messaging) sensor network in this paper. Reliability is an important issue in sensor network and will be addressed in a more extended version of this paper in the future due to space constraint. More formally:

**Lemma 1 (Error Bounding).** *For any sensor  $s$ , let  $Z_t(s)$  be the actual value and  $Z_t^*(s)$  be the estimated value of  $s$  for attribute  $A$  at time  $t$ . The  $|Z_t^*(s) - Z_t(s)| < \epsilon$  using the proposed algorithms for each sensor and the central site.*

*Proof.* There are two cases to consider:

1. If  $s$  reported its value, then  $Z_t^*(s) = Z_t(s)$  which implies  $|Z_t^*(s) - Z_t(s)| = 0 < \epsilon$  since from the problem statement  $\epsilon > 0$ .
2. If  $s$  did not report its value, then  $Z_t^*(s) = \text{interp}(R)$  where  $R$  is the set of values that reported in the first round and that are within distance  $r$  from  $s$ . This is the same as the estimated value used in the second round for sensor  $s$ . If  $|Z_t^*(s) - Z_t(s)| \geq \epsilon$ , then  $s$  would have reported. Therefore  $|Z_t^*(s) - Z_t(s)| < \epsilon$ .

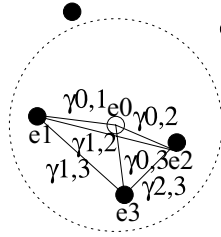
Once all the sense readings are recovered, the space is gridded based on a given spatial resolution. A raster map is created by interpolating locations/cells without any sensor readings from all the sensors ( $Z_t^*(s), s \in S$ ).

### 3.2 E2K: Choosing Interpolation Method

Although our framework is general with respect to the interpolation method used and always guarantees an error bound, which interpolation method is used has a great impact on the performance of the E2K framework. Furthermore, we need to make sure the sensor field can handle sophisticated interpolation models without incurring too much storage and computation cost.

Let  $Z(e)$  be a random function at a location  $e$  of a sensor field. The value of a location  $e_0$  is estimated by a linear estimator using the neighbors,  $e_1, e_2, \dots, e_n$ , within distance  $r$ :

$$Z^*(e_0) = \sum_{i=1}^n \lambda_i Z(e_i) \quad (1)$$



**Fig. 3.** Kriging Example

where  $Z^*(e_0)$  represents the estimated value at location  $e_0$ , and  $\lambda_i$  refers to the weight given to the  $i^{th}$  neighbor’s value. For example, in Figure 3, we wish to estimate a value at  $e_0$ , using the data values from the  $n$  neighboring sample points  $e_i, i = 1, 2, 3$ . So, we have  $Z^*(e_0) = \lambda_1 Z(e_1) + \lambda_2 Z(e_2) + \lambda_3 Z(e_3)$ .

There are a number of ways to assign the weights. It is desirable for the sum of the weights to be one so that if all neighbors have the same value, then this value will be the estimate. One way would be to assign equal weights to all neighbors, i.e., to take a simple average of the neighbors values (**Simple Average**). Another scheme would be to use the inverse of the distance as the weights, i.e.,  $\lambda_j = \frac{1/d_j}{\sum_{i=1}^n 1/d_i}$ , where  $d_i$  is the distance from  $e_i$  to  $e_0$  (**Inverse Distance**). However, these techniques do not examine the spatial structure of the data and may result in large estimation errors.

Kriging [7,28] is widely used and has a long history of popularity in many natural science domains. It is a best fit linear unbiased estimator of a spatial variable at a particular site or geographic area. The goal of kriging is to create a raster map of a given resolution to represent a surface using a set of sample readings. It estimates a value at a location of a region for which a covariance/-variogram is known, or can be estimated using data in the neighborhood of the estimation location.

We assume  $Z(e)$  is *second-order stationary*. This means the expected value  $E[Z(e)] = m$ , where  $m$  is the mean, for any point of the domain; and the covariance between any pair of locations depends only on the vector  $h$  that separates them, i.e.,  $C(h) = C(e, e + h) = E[Z(e) \times Z(e + h)] - m^2$ . We chose ordinary kriging due to its adaptivity to local conditions, i.e., it only requires local second-order stationarity as opposed to global. In ordinary kriging, the estimation of a location  $e_0$  can be expressed by equation 11

Kriging assigns weights according to a known or estimated covariance/variogram function which captures the spatial autocorrelation. The variogram  $2\gamma_Z(h)$  is defined as  $Var[Z(e + h) - Z(e)]$ . Using sampled data the semivariogram is estimated as:

$$\hat{\gamma}_z(h) = \frac{1}{2N_h} \sum_{i=1}^{N_h} (z(e_i) - z(e_i + h))^2$$

where  $N_h$  is the number of pairs of samples whose distance from each other is  $h$ .

The empirical semivariogram is then fit with a theoretical model. For example, right figure of [1](#) shows the empirical semivariogram for an Asia temperature dataset [1](#) fit with an exponential model. (the two datasets will be described further in section [4](#)).

There are two important parameters of the variogram model: the range and the sill. The range is the distance it takes for the variogram to reach the sill. The sill is an asymptotic bound the variogram reaches indicating that those values no longer have a meaningful correlation. We use range as a guidance in choosing the neighborhood parameter  $r$  for our algorithms in our E2K framework. The covariance can be expressed in terms of the variogram as  $C(h) = C(0) - \gamma(h)$ .

Once the variogram is known, the weights are chosen to minimize the error variance of the estimated values. The error variance can be calculated as:

$$\begin{aligned} \sigma_E^2 &= E[(Z^*(e) - Z(e))^2] \\ &= E[(Z^*(e))^2] - 2E[Z^*(e) \times Z(e)] + E[(Z(e))^2] \\ &= \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j E[Z(e_i) \times Z(e_j)] \\ &\quad - 2 \sum_{i=1}^n \lambda_i E[Z(e_i) \times Z(e)] + E[(Z(e))^2] \\ &= \sum_{i=1}^n \sum_{j=1}^n \lambda_i \lambda_j (C(e_i, e_j) + m^2) \\ &\quad - 2 \sum_{i=1}^n \lambda_i (C(e_i, e) + m^2) + C(0) \end{aligned}$$

Since the sum of the weights should be one, a Lagrange parameter  $\mu$  is added to get  $L = \sigma_E^2 + 2\mu\{1 - \sum_{i=1}^n \lambda_i\}$  along with the constraint  $\sum_{i=1}^n \lambda_i = 1$ . Optimal values for weights  $\lambda_i, i = 1, 2, \dots, n$ , are then obtained by the standard method of taking first derivatives of  $L$  with respect to each weight  $\lambda_i$  and setting them to zero.

$$\begin{aligned} \frac{\partial(L)}{\partial(\lambda_i)} &= 2 \sum_{j=1}^n \lambda_j (C(e_i, e_j) + m^2) - 2(C(e_i, e) + m^2) - 2\mu \\ &= 2 \sum_{j=1}^n \lambda_j C(e_i, e_j) - 2C(e_i, e) - 2\mu \\ &= 2 \sum_{j=1}^n \lambda_j (C(0) - \gamma(e_i, e_j)) - 2(C(0) - \gamma(e_i, e)) - 2\mu \\ &= -2 \sum_{j=1}^n \lambda_j \gamma(e_i, e_j) + 2\gamma(e_i, e) - 2\mu = 0 \\ &\implies \sum_{j=1}^n \lambda_j \gamma(e_i, e_j) + \mu = \gamma(e_i, e), \text{ for } i = 1, 2, \dots, n \end{aligned}$$

If the variogram function  $\gamma(e_i, e_j)$  is given or can be estimated, this system along with the constraint on the weights gives us  $n + 1$  equations,  $n$  unknown weights  $\lambda_i, i = 1, \dots, n$ , and the unknown Lagrange parameter  $\mu$  that we wish to obtain through solving the linear system. In Figure 3, to obtain the weights for estimating the value of  $e_0$ , we have the following system:

$$\begin{pmatrix} \gamma(e_1, e_1) & \gamma(e_1, e_2) & \gamma(e_1, e_3) & 1 \\ \gamma(e_2, e_1) & \gamma(e_2, e_2) & \gamma(e_2, e_3) & 1 \\ \gamma(e_3, e_1) & \gamma(e_3, e_2) & \gamma(e_3, e_3) & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \mu \end{pmatrix} = \begin{pmatrix} \gamma(e_1, e_0) \\ \gamma(e_2, e_0) \\ \gamma(e_3, e_0) \\ 1 \end{pmatrix}$$

Because kriging is preceded by an analysis of the spatial structure of the data by using a variogram model to integrate the average variability into the estimation model, the interpolation is likely to be more accurate than simple models. For the ordinary kriging that we described, the interpolation is exact, meaning if a sample value is available at the location of estimation, the kriging solution is equal to that value. Furthermore, kriging as a statistical method provides an indication of the estimation error.

The two main issues in the E2K framework using kriging are training and processing. Unlike the simple average and inverse distance schemes, kriging uses a variogram that is determined using previous data. Fortunately, all the training could be performed at the sink where resources are not constrained. After the training, only a few values used to describe the theoretical variogram need to be disseminated to the sensor network. Specifically, we use a training surface to obtain an empirical variogram at the sink. Then a theoretical variogram is fit to the empirical variogram with the goal of fitting the data with distance less than  $r$  as close as possible, again at the sink. The theoretical variogram is described by three parameters, namely a type (exponential, spherical, etc), range, and sill is provided to each sensor. As such, a fairly small amount of information needs to be sent to the sensors to allow sensor field and the sink to have the same interpolation model.

In terms of processing the main requirement is that the sensor is capable of solving the linear system once it determines which neighbors report in the first round. Due to the computation constraint of sensors, a large linear system is not desirable. Fortunately for kriging not many neighbors are needed to interpolate a value. E2K achieves a desired number of neighbors for each sensor by setting the value of the probability to report in the first round in our sensor side algorithm to  $\frac{n_{desired}}{n_{current}}$ , where  $n_{desired}$  is the number of neighboring sensors desired and  $n_{current}$  is the current number of neighboring sensors within the distance  $r$ . With a sensor deployment following a Poisson distribution, we expect  $\frac{n_{desired}}{n_{current}}$  percent of all the sensors will report in the first round. By setting a small value for the  $n_{desired}$  parameter, e.g. 5, it also results in the beneficial effect of dynamically adapting to the density of the sensor network and provides more savings for dense sensor networks with a similar level of spatial autocorrelation.

The only messages that need to be sent for coordination are the messages sent by the sensors that decide to report in the first round to their neighbors. However, since the sensors which decide to report in the first round need to send

their readings to the sink, the other sensors in the neighborhood can ear-drop the reading. So when the interpolation neighborhood is less than the radio range of the sensors, the coordination cost in terms of messaging is 0. Formally, we have:

**Lemma 2 (Conditional Zero Coordination Cost).** *In E2K, the number of messages sent in order to coordinate sensors in deciding which ones need to report and maintain an error bound is 0 when the spatial interpolation neighborhood is less than or equal to the radio range.*

## 4 Evaluation

Performance of E2K was evaluated using two datasets: (1) *Lab*: an Intel lab dataset [20] consisting of 54 sensors running for a little over a month monitoring temperature, voltage, humidity, and light. This data contains traces from a real sensor network deployment including network failure information, so it is particularly useful for examining failures. (2) *Asia Temperature*: the Asian portion of a dataset created using station records from the Global Historical Climatology Network (GHCN) and Legates and Willmott’s [18] datasets for monthly precipitation and air temperature. We used version 1.02 of this dataset, released in July of 2001, that contains data for each month from January 1950 to December 1999 [1]. The Asia subset was sampled randomly to get two densities with 25% and 75% of the points respectively. In our experiments, these sampled points were treated as sensors that formed a sensor network. This data set provides a large number of sensors in an outdoor environment where spatial correlations are stronger.

We compare our model with 6 other models on the two datasets:

- A base line model **TinyDB (T)** [22]: Every sensor will report in every epoch. The error is always 0, thus it is bounded in this scheme.
- Three models where each sensor only reports if its readings deviates from a reading agreed upon by both the sensor field and the sink by more than  $\epsilon$ :
  - (1) **Global Average (G)**: Both the sink and sensors keep a global average reading of all sensors obtained from the training data. If a sensor deviates from the global average by more than  $\epsilon$ , then it reports;
  - (2) **Approximate Caching (A)** [24]: Every sensor caches the last reported reading to the sink so the sink knows the same value as the sensor in case the sensor does not report. A sensor reports if the real reading deviates from the cached values by more than  $\epsilon$ ;
  - (3) **Periodical Approximate Caching (P)**: Same as approximate caching except that every sensor caches the last reported readings for a set of episodes and uses the previous episode to determine its value, e.g. for a 24 hour cycle the value is checked against the reading from the same hour in the previous day. The lab data uses a 24 hour period and the Asian temperature data uses a 12 month period.
- **Ordinary Kriging (O)**: E2K with *ordinary kriging* as the interpolation method. For the Asian temperature data, we removed the trend of the data

for each sensor by subtracting periodic means obtained from the training data for each location. Each sensor will subtract that mean and use E2K for reporting its residual. The sink knows the same mean and will add it back upon receiving or interpolating the residual.

- Two schemes using the E2K framework but with simple interpolation methods without considering spatial structure: (1) **Simple Average (S)**: E2K with *simple average* as the interpolation method; (2) **Inverse Distance (I)**: E2K with *inverse distance* as the interpolation method.

The abbreviations for the schemes are summarized in Table 1 for the convenience of the readers. For all schemes, we implemented a program using Java to simulate the reporting behavior of each sensor node using the readings of each sensor and its neighboring sensors for a given time from Lab and Asia datasets (we are currently implementing the algorithms on Berkeley Motes for a project to monitor soil moisture at Ray Roberts Greenbelt of North Texas). Unless otherwise specified, the radio range is 30 meters and the error threshold is  $0.5^{\circ}\text{C}$  for the lab data, and the radio range is 5 degrees with an error threshold  $1^{\circ}\text{C}$  for the Asian temperature data.

**Table 1.** Abbreviations of Comparison Schemes

T: TinyDB	G: Global Avg.	A: Appr. Caching	P: Periodical Appr. Caching
S: Simple Avg.	I: Inverse Distance	O: Ordinary Kriging	

For the lab dataset the total messaging cost was estimated using a fixed multiple of the number of reporting sensors, i.e., we assumed each sensor had to send a message to the central site using some fixed number of hops. This was done because the area and number of sensors are too small for there to be a large number of hops. For the Asia data set the central site was chosen to be the center of the map and the number of hops to send a message was estimated taking into account the distance of each reporting sensor to the sink. For the lab data the first 94 hours was used for training and the next 452 hours was used for testing. For the Asian temperature data the first 10 years was used for training and the next 40 years was used for testing. We performed extensive experiments using various values for the parameters and, due to the limitation of space, we present a representative set.

#### 4.1 Performance on Asia Temperature Dataset

**Messaging Savings in Percentage that Report.** In this section, we examine the percentage of sensors that actually reported for all the schemes. Note the communication cost needed to route the data to the sink are not factored in yet.

Figure 4 shows the performance of the schemes with respect to the density of the sensors and the error threshold  $\epsilon$ . The figures in Figure 4 show that E2K outperforms all the other schemes in terms of percentage of sensors that need



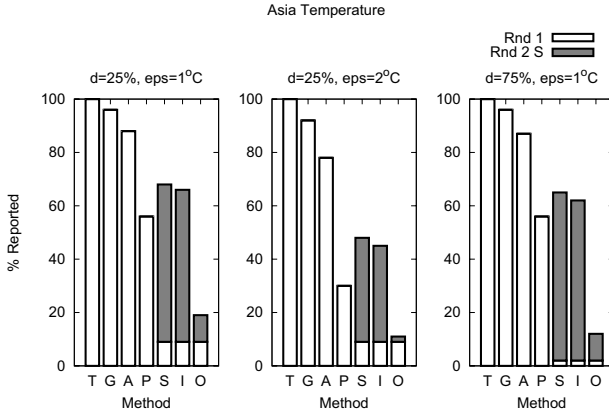


Fig. 4. Percentage Report for Asia Temperature Data

to report. Compared to the baseline method **T**, two round reporting helps E2K schemes, i.e., **O**, **S** and **I**, avoid unnecessary reporting from sensors that can be interpolated from the sink. E2K using ordinary kriging reduces the percentage of sensors that need to report to 19% with  $\epsilon = 1$  and 12% with  $\epsilon = 2$  for density of 25%. For density of 75%, the percentage of sensors that need to report are 13% with  $\epsilon = 1$ .

With increased density from 25% to 75%, the spatial related schemes including ordinary kriging (**O**), simple average (**S**), and inverse distance (**I**) show various levels of increased savings, ranging from 3% to 6%. There is no noticeable increase in savings for **G** when the density is increased. As expected, performance of temporal methods including approximate caching (**A**) and periodical approximate caching (**P**) do not change according to sensor density. In general

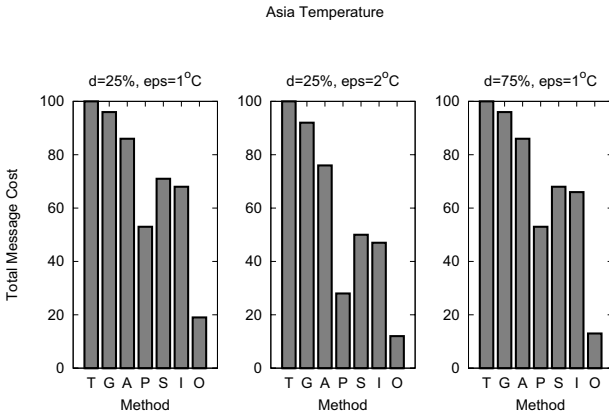


Fig. 5. Total Messaging Cost for Asia Temperature Data

approximate caching and global average do not do well resulting in more than 80% reporting sensors.

**Total Messaging Savings.** In this section, we report the performance of the schemes with routing cost taken into consideration. Figure 5 shows the percent of total messaging cost for each method. It is important to note that the trend is the same as the percentage to report discussed previously. By keeping neighbors within radio range to eliminate coordination cost the most important factor is the number of sensors to report.

### 4.2 Performance on Lab Dataset

For lab data, schemes using E2K framework, i.e. **O**, **S**, and **I**, outperforms **T**, **G**, and **P** by more than 20%. With  $\epsilon$  increasing, the savings increase. Approximate caching performs the best due to the strong temporal correlation in this dataset. Incorporating temporal compression schemes into our E2K framework to allow adaptive utilization of spatial or temporal autocorrelation based on whichever is stronger will be an interesting topic for future work. An interesting observation is

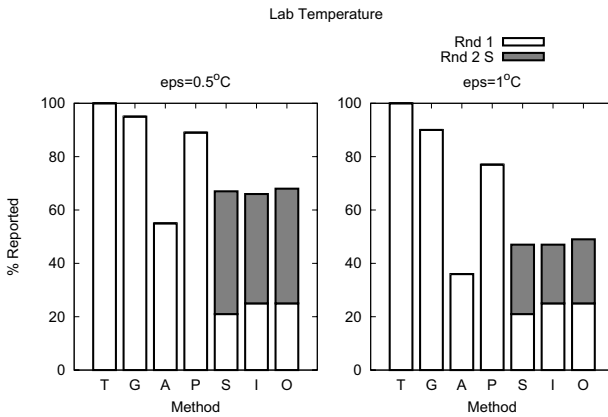


Fig. 6. Percentage Report for Lab Data

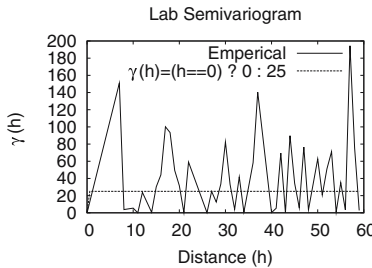


Fig. 7. Lab Data Empirical and Theoretical Variograms

that the schemes using E2K framework achieve similar performance to each other with no method being clearly superior. The reason is that the lab data is not a typical outdoor environmental monitoring setup and the spatial autocorrelation is not captured well by variograms. When using a nugget theoretical function (straight line parallel to x-axis) as shown in Figure 7 for the variogram, kriging interpolation degenerates into a simple average model. A promising topic for future research is a query optimizer for the sink that uses variograms to choose the best interpolation model. In case a simple average or reverse distance model is chosen, we will have the advantage of a simplified decision making process at sensor nodes.

## 5 Extensions and Conclusion

In this work, we proposed the E2K framework which can use any general spatial interpolation method. When the spatial interpolation method works well, i.e., the residual errors are small and it requires only localized information for spatial interpolation, our framework is likely to save more on messaging cost. Possible future extensions of our framework include incorporating temporal compression to and considering the use of regression or co-kriging to utilize auxiliary variables.

## References

1. University of delaware surface air temperature data.  
<http://climate.geog.udel.edu/~climate>
2. Ali, M.H., Aref, W.G., Nita-Rotaru, C.: Spass: Scalable and energy-efficient data acquisition in sensor databases. In: *MobiDE (2005)*
3. Bash, B.A., Byers, J.W., Considine, J.: Approximately uniform random sampling in sensor networks. In: *DMSN (2004)*
4. Bonnet, P., Gehrke, J.E., Seshadri, P.: Towards Sensor Database Systems. In: *Proc. of Second International Conference on Mobile Data Management (2001)*
5. Chu, D., Deshpande, A., Hellerstein, J., Hong, W.: Approximate data collection in sensor networks using probabilistic models. In: *ICDE (2006)*
6. Considine, J., Li, F., Kollios, G., Byers, J.: Approximate aggregation techniques for sensor databases. In: *ICDE (2004)*
7. Cressie, N.A.C.: *Statistics for Spatial Data*. Wiley and Sons, Chichester (1991)
8. Deligiannakis, A., Kotidis, Y., Roussopoulos, N.: Compressing historical information in sensor networks. In: *ACM SIGMOD*, pp. 527–538. ACM Press, New York (2004)
9. Deshpande, A., Guestrin, C., Madden, S.R., Hellerstein, J.M., Hong, W.: Model-driven data acquisition in sensor networks. In: *Proc. of VLDB*, pp. 588–599 (2004)
10. Emekci, F., Tuna, S.E., Agrawal, D., Abbadi, E.: Binocular: A system monitoring framework. In: *International Workshop on Data Management for Sensor Networks (August 2004)*
11. Fang, Q., Zhao, F., Guibas, L.: Counting targets: Building and managing aggregates in wireless sensor networks. Tech. Report, Palo Alto Research Center (2002)
12. Goel, S., Passarella, A., Imielinski, T.: Using buddies to live longer in a boring world, 2004. Rutgers Depart. of Computer Science Tech. Report DCS-TR-558 (2004)

13. Harrington, B., Huang, Y.: In-network surface simplification for sensor fields. In: ACM-GIS, ACM Press, New York (2005)
14. Jain, A., Chang, E.Y., Wang, Y.-F.: Adaptive stream resource management using kalman filters. In: SIGMOD (2004)
15. Karp, B., Kung, H.T.: Gpsr: greedy perimeter stateless routing for wireless networks. In: MobiCom (2000)
16. Kotidis, Y.: Snapshot queries: Towards data-centric sensor networks. In: ICDE, pp. 131–142 (2005)
17. Krishnamachari, B., Estrin, D., Wicker, S.B.: The impact of data aggregation in wireless sensor networks. In: Proceedings of the 22nd International Conference on Distributed Computing Systems, pp. 575–578 (2002)
18. Legates, R.D., Willmott, C. J.: Mean seasonal and spatial variability in global surface air temperature. *Theor. Appl. Climatol.* , 11–21 (1990)
19. Li, M., Ganesan, D., Shenoy, P.: Presto: Feedback-driven data management in sensor networks. In: Proceedings of the Third ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI) (May 2006)
20. Madden, S.: Intel lab data. <http://berkeley.intel-research.net/labdata/>
21. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tag: a tiny aggregation service for ad-hoc sensor networks. In: OSDI (2002)
22. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Design of an acquisitional query processor for sensor networks. In: SIGMOD (2003)
23. Madden, S.R., Szewczyk, R., Franklin, M.J., Culler, D.: Supporting aggregate queries over ad-hoc wireless sensor networks. In: Workshop on Mobile Computing and Systems Applications (2002)
24. Olston, C., Loo, B.T., Widom, J.: Adaptive precision setting for cached approximate values. In: SIGMOD Conference (2001)
25. Sharifzadeh, M., Shahabi, C.: Supporting spatial aggregation in sensor network databases. In: GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems, ACM Press, New York (2004)
26. Trigoni, N., Yao, Y., Demers, A., Gehrke, J., Rajaraman, R.: WaveScheduling: Energy-Efficient Data Dissemination for Sensor Networks. Internet Draft (2004)
27. Vuran, M.C., Akan, B., Akyildiz, I.F.: Spatio-temporal correlation: theory and applications for wireless sensor networks. *Comput. Networks* 45(3) (2004)
28. Wackernagel, H.: *Multivariate Geostatistics*. Springer, Heidelberg (1995)
29. Yao, Y., Gehrke, J.: The cougar approach to in-network query processing in sensor networks. In: Proceedings of SIGMOD (2002)
30. Yu, Y., Govindan, R., Estrin, D.: Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks, UCLA Computer Science Department Technical Report UCLA/CSD-TR-01-0023 (2001)
31. Zhao, F., Guibas, L.: *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, San Francisco (2004)

# Online Amnesic Summarization of Streaming Locations

Michalis Potamias<sup>1</sup>, Kostas Patroumpas<sup>2</sup>, and Timos Sellis<sup>2</sup>

<sup>1</sup> Computer Science Department, Boston University, MA, USA

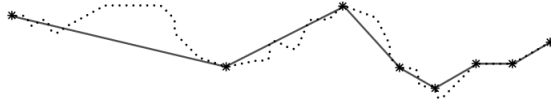
<sup>2</sup> School of Electrical and Computer Engineering  
National Technical University of Athens, Hellas  
mp@cs.bu.edu, {kpatro,timos}@dbnet.ece.ntua.gr

**Abstract.** Massive data streams of positional updates become increasingly difficult to manage under limited memory resources, especially in terms of providing near real-time response to multiple continuous queries. In this paper, we consider online maintenance for spatiotemporal summaries of streaming positions in an aging-aware fashion, by gradually evicting older observations in favor of greater precision for the most recent portions of movement. Although several *amnesic* functions have been proposed for approximation of time series, we opt for a simple, yet quite efficient scheme that achieves contiguity along all retained stream pieces. To this end, we adapt an amnesic tree structure that effectively meets the requirements of time-decaying approximation while taking advantage of the succession inherent in positional updates. We further exemplify the significance of this scheme in two important cases: the first one refers to trajectory compression of individual objects; the other offers estimated aggregates of moving object locations across time. Both techniques are validated with comprehensive experiments, confirming their suitability in maintaining online concise synopses for moving objects.

## 1 Introduction

The tremendous amount of information flowing as transient *data streams* in many modern applications that monitor the current position of people, vehicles, animals etc. or track their trajectories, clearly calls for single-pass processing. It is inevitable that not all data can be retained permanently in memory, so stale items can either be archived on disk or even discarded to make room for newly arriving tuples. Sometimes, it is indispensable to maintain a finite *window* over the stream, which provides access to the most recent stream items [1].

More importantly, the significance of each isolated positional tuple is *time-decaying*: when it first arrives, it perhaps conveys critical information, but it gets less and less important as time goes by, until it eventually becomes obsolete and practically useless. Therefore, the older a data item gets, the coarser its representation could become in a progressive fashion, implying that greater precision should be reserved for the most recent items. This is essentially reminiscent of the way human memory actually works: we can accurately describe



**Fig. 1.** Amnesic approximation of a trajectory

recent events in much detail, but we can hardly recollect facts that occurred a long time ago.

This treatment of data has been termed *amnesic* with respect to time series approximation [9], in the sense that acceptable error margin in data is allowed to increase in proportion to its age. A wide range of amnesic functions was identified for controlling the amount of error tolerated at every single point in the time series. Piecewise linear approximation (PLA) functions were superior in terms of incremental computation and availability of various distance metrics [9].

In this paper, we turn our focus on amnesic approximation of spatiotemporal data streams generated by tracking a large number of moving objects. This approach is mainly dictated by the sheer volume of data: trajectories can become rather lengthy when continuously accumulating positional updates, and thus difficult to accommodate in full precision for their entire history. In addition, different levels of abstraction are also inherent in semantics related to multi-scale representation of spatial features, given that more precision is allocated to a user-defined area of interest. In a spatiotemporal context, this specific interest can be interpreted as the current or most recent location of objects.

Although our primary interest is on spatiotemporal streams, the framework we propose can certainly be applied over typical streams comprised of items in a sequence, such as sensor readings or stock tickers across time. In particular, we present the *multiple-granularity* AmTree framework, by adapting a tree structure from [3]. AmTree accepts streaming items and maintains summaries over hierarchically organized levels of precision, realizing an amnesic treatment over stream portions. This mechanism can produce reduced data representations for approximate query answering, by efficiently handling streaming locations from moving point objects or retaining a rough outline of their trajectories. As illustrated in Fig. 1, this summary keeps more dense information for the recent past, while older segments are evidently underrepresented. As we show in this paper, not only is this specific framework proven sufficient to cope with on-line trajectory approximation, but it can further be used in spatiotemporal aggregation, in order to estimate distinct count queries over locations of moving objects.

The distinguishing characteristics of our approach as opposed to general piecewise linear approximation are twofold. First, we preserve *contiguity* among successive trajectory segments for each individual object, no matter how coarse the level of approximation gradually becomes. In fact, we do not purposely introduce fictitious points to obtain a smoother approximation, but we retain a subset of actual locations to keep the reduced representation consistent to the original data. Second, the *hierarchical* behavior of our transform fits well to the

streaming nature of incoming locations. Our amnesic tree structure gradually achieves a coarser representation for each trajectory without sacrificing its latest details.

Our contributions can be summarized as follows:

- We propose a generic structure named AmTree and we formalize its basic operations. AmTree exploits the notion of temporal timeliness and is capable to maintain a compact amnesic representation of streaming items.
- We demonstrate how this structure can be applied to address two challenging issues in spatiotemporal data compression, namely concise approximation of entire trajectories and distinct count estimation over moving objects.
- We present an extensive experimental study of our techniques, using large synthetic datasets. These experiments confirm the robustness of the proposed structure and the high-quality synopses it produces.

The remainder of this paper proceeds as follows. Section 2 discusses some preliminary concepts. In Section 3, we present the structure of AmTree along with its basic operations and characteristics. In Section 4, this scheme is utilized to maintain amnesic approximations at a single-trajectory level. In Section 5, we introduce a composite structure based on AmTree capable to estimate distinct counts of numerous moving objects. Experimental results are discussed in Section 6. Section 7 reviews related work, while Section 8 offers concluding remarks.

## 2 Aging Stream Features at Multiple Time Granularities

Time dimension is intuitively liaised to several levels of detail with respect to a time domain. A *time granule* is a unified set of discrete time instants that can be used as a time reference for data items. Consecutive and non-overlapping time granules can be merged into greater ones in an iterative fashion, thus defining several levels of *granularity* [2], like seconds, minutes, hours, days etc.

More concretely, let  $G_0, G_1, \dots, G_{k-1}$  denote  $k$  successive levels of granularity, respectively characterized by a granule unit  $\delta_0, \delta_1, \dots, \delta_{k-1}$ . Granule unit  $\delta_i$  at level  $i$  consists of a fixed set of primitive time instants, e.g., an hour granule consists of 3600 seconds (assuming seconds as primitive instants). In our approach, we require that each granule unit  $\delta_i$  at level  $i$  subsumes a fixed number of contiguous non-overlapping granules of unit  $\delta_{i-1}$  at level  $i-1$ . Assuming that granularity level  $G_i$  consists of  $n$  granules  $g_0^i, g_1^i, \dots, g_{n-1}^i$ , each such granule is equivalent to a fixed-size finite set of contiguous instants. In general, it may occur that the number of granules making up the immediately higher granule (i.e., the *granularity factor*) varies across levels, as occurs for hours, days, months etc.

The whole process implies a hierarchical composition of granules, as long as their endpoints coincide throughout different levels of granularity. Days, months and years can be defined in that order one from the other, since the intervals they span are entirely contained in a granule higher up in the hierarchy. Apparently, such a hierarchical scheme can be effectively represented by a *tree*: its lower level corresponds to the finer granules, and each successive level higher in the tree to

coarser ones, up to the root level that denotes the coarsest granule available. By simply ascending or descending that tree, we achieve varying levels of detail.

When it comes to data representation, a hierarchical scheme can be advantageous in referencing aging stream portions. A *data stream*  $S$  may be regarded as an ordered sequence of items  $\langle s, \tau \rangle$ , where  $s$  is a typical relational tuple (e.g., carrying object id's and positions) and  $\tau$  its timestamp value.

We envisage a mechanism where streaming tuples are taken in at the finest granularity  $\delta_0$ ; periodically, as soon as all items spanning the duration  $\delta_i$  of any given level  $i > 0$  have been received, they are combined into a summary assigned to a granule at level  $i$ . Although this process is similar for all levels, it is applied at time instants that denote endpoints of the respective granules. For instance, stream items may arrive every second and a synopsis is emitted each minute; when 60 such synopses have been produced, a more coarse synopsis is made up over past hour and so on, until the higher level (e.g., year) is reached. In addition, data items of a finer granule unit  $\delta_{i-1}$  could get discarded as soon as they are summed up at a coarser granule unit  $\delta_i$ . This way, less and less detailed information will be maintained for older stream portions, practically implementing a deterministic *time-decaying* policy regarding data stream summarization.

### 3 The Amnesic Tree

Next, we present the proposed AmTree structure, which maintains a time-aware, hierarchical amnesic synopsis of a data stream. As it will become evident soon, we opt for a summarization scheme that manipulates pairs of items at every granularity level. Still, the proposed framework is much more general and can be easily calibrated to work with varying number of granules at each level.

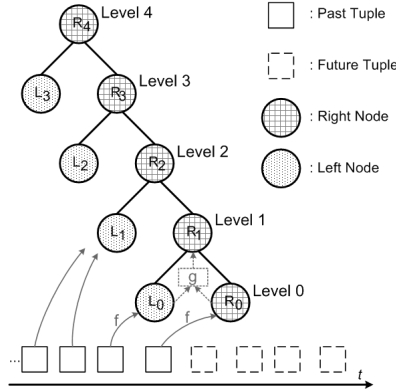
#### 3.1 Structure and Properties

The general structure of an AmTree is illustrated in Fig. 2. We enumerate granularity levels of the tree starting from the lowest one ( $0^{th}$  level). We assume that the granularity factor is 2; hence, a granule at each level spans two granules half its size at the level beneath. Except for the root, each level  $i$  of the tree consists of two nodes, the right ( $R_i$ ) and the left one ( $L_i$ ). At the  $0^{th}$  level, each node accepts data with reference to the finest granularity unit  $\delta_0$ , which characterizes every timestamp attached to incoming tuples. Each node at the  $i^{th}$  level contains information about twice as many timestamps as a node at the  $(i - 1)^{th}$  level. Hence, a node at level  $i$  contains information characterizing  $2^i$  timestamps.

Let  $N$  denote the number of stream items received thus far by AmTree, each one carrying information at the finest granularity unit  $\delta_0$ . Then, we can easily calculate the height  $H$  of the tree, i.e., the total number of levels, as  $H = \lceil \log_2 N \rceil + 1$ . Thus, the number of tree nodes is  $2 * H - 1$ .

Without loss of generality, we assume that AmTree accepts tuples with distinct successive timestamps, so we adhere to a count-based stream model where each new item is timestamped with the next available sequential integer value.





**Fig. 2.** Basic structure of AmTree

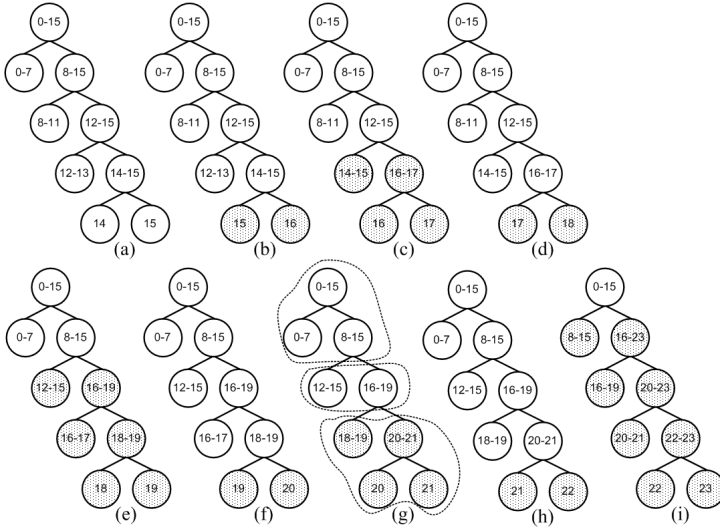
Even without this precondition, we may assume a mapping  $f$  that is applied over the batch of tuples with current timestamp value  $\tau$  and transforms them into a single tuple that can be the content of a tree node. As shown in Fig. 2, the resulting content is assigned to node  $R_0$ , while the previous content of  $R_0$  is shifted to node  $L_0$ . As time goes by and new data comes in, the contents of each level are combined using a function  $g$  and propagated higher up in the tree, retaining less detail. This process is performed using the following methods:

- $new(\tau)$  is invoked as soon as a new item of timestamp  $\tau$  has arrived and it simply assigns the result of mapping  $f$  into node  $R_0$ . Note that node  $R_0$  is the only entry point to the synopsis prepared by the AmTree.
- $shift(i)$  discards the contents of node  $L_i$  and shifts contents of  $R_i$  into  $L_i$ . If node  $L_i$  does not exist, then  $shift(i)$  allocates appropriate node space.
- $merge(i)$  invokes function  $g$  over the contents of  $L_i$  and  $R_i$  and assigns their combined result to node  $R_{i+1}$ . If  $R_{i+1}$  does not yet exist, then node space is allocated, effectively increasing the height  $H$  of the tree.

### 3.2 Streaming Operation

In stream processing, *sliding windows* are utilized to fetch the most recent items qualifying to a standard extent of interest in terms of time interval (e.g., ten minutes long) or number of stream tuples (e.g., 100 tuples). *Landmark windows* specify a fixed anchor point in time and they continue appending newly arriving tuples (e.g., get all items after 10 a.m. today). Our intention is to handle stream elements in a way that combines the characteristics of both window modes. While we need to follow stream evolution in a sliding fashion, we must still keep some reference to its long-running history after a specific “landmark” in time. This approach actually resembles a *tilt time frame* model [4] and fulfils the requirements of a time-decaying representation.

We will demonstrate the update procedure with an example depicted in Fig. 3. For clarity, nodes are shown filled with timestamps and not with the actual



**Fig. 3.** Successive snapshots of an AmTree

stream values. We illustrate 9 successive states of AmTree, starting (Fig. 3a) from a snapshot after 16 insertions (i.e., timestamps  $0, 1, 2, \dots, 14, 15$  have been processed). The case when tuple with  $\tau = 16$  is inserted into the tree is depicted in Fig. 3b, where all updated nodes are highlighted. In particular,  $shift(0)$  is invoked to transfer the current contents of  $R_0$  into  $L_0$ , and subsequently  $new(16)$  assigns a new content to  $R_0$ , as always occurs at every new insertion.

When tuple of  $\tau = 17$  arrives, another update call is invoked; this time, level 1 will be also updated as long as both nodes of level 0 have entirely new information since the last update of level 1 occurred (Fig. 3c). Level 0 is processed in exactly the same way as before, while level 1 is processed in two steps: first,  $shift(1)$  is employed to discard contents of  $L_1$  and transfer  $R_1$  to  $L_1$ ; consequently,  $merge(0)$  causes the new content of  $R_1$  to be a combination of  $R_0$  and  $L_0$ .

Observe that updates work in a bottom-up fashion. So, level 0 will be updated at every insertion, by always calling  $shift(0); new(\tau)$ . Accordingly, level 1 will be updated every second tuple, because only then contents of level-0 nodes have not yet been merged. So, whenever the update procedure reaches level 1, calls  $shift(0); new(\tau); shift(1); merge(0)$  must have been invoked. At  $\tau = 19$  (Fig. 3e), levels 0, 1 and 2 will be updated, but updates will not proceed to level 3, since  $R_3$  still temporally overlaps with  $L_2$ .

Intuitively, level 2 will be updated every fourth tuple, level 3 every eighth tuple and so on. This example indicates that each level  $i$  is updated every  $2^i$  timestamps ( $i = 0, 1, 2, \dots$ ). We can now predict that level 3 is updated at timestamps 7, 15, 23 (Fig. 3i), and so on. At each incoming timestamp, the update procedure ascends the tree up to a maximum level  $M$ , which depends on the  $N$  stream items seen so far. Clearly,  $M$  can be derived as the power of 2, if  $N$  is analyzed in its prime factors, i.e.,  $M = \max\{k : \exists \gamma \in \mathbb{N}, \gamma \cdot 2^k = N\}$ .

Updating AmTree is clearly an online technique that incurs logarithmic memory storage and can be performed quite quickly. More formally:

**Lemma 1.** *The space complexity of the AmTree structure is  $O(\log N)$ .*

**Lemma 2.** *The amortized time complexity of the AmTree update procedure is  $O(1)$  per stream tuple.*

*Proof.* Each level  $i$  of the AmTree is updated every  $2^i$  timestamps and we assume cost  $O(1)$  to update a single level. So, when  $N$  tuples (i.e., timestamps) have arrived, every level has been updated  $\lfloor \frac{N}{2^i} \rfloor$  times. Since the total number of tree levels is  $H = \lfloor \log N \rfloor + 1$ , this results to a cost of  $\sum_{i=0}^{H-1} O(1) \frac{N}{2^i} = O(N)$ , because  $\sum_{i=0}^{H-1} \frac{1}{2^i} < 2$ . Thus, the amortized update cost per tuple is  $O(1)$ .  $\square$

### 3.3 Multiple Concurrent AmTrees

The basic AmTree structure (Section 3.1) has non-tunable characteristics that determine its aging approximations. Relaxing this strict control over time decaying rate, we can generalize it into a forest of AmTrees of varying resolution.

AmTreeH is a set of  $m$  *concurrently maintained* AmTree structures. When a stream tuple arrives for processing, it gets inserted into just one AmTree. The appropriate  $\text{AmTree}^{(p)}$  is determined by a hash function  $p = h(\tau)$  computed over the current timestamp value  $\tau$ . Therefore, amortized update complexity is still  $O(1)$  per tuple, while worst-case space complexity is  $O(m \log N)$ .

AmTreeH achieves tunable amnesic features depending on the mappings performed by function  $h$ . To exemplify its usage, we consider hash function  $p = h(\tau) = \max\{i \in \mathbb{N} : \exists \lambda \in \mathbb{N}, \tau = 2^i + \lambda \cdot 2^{i+1}\}$ ,  $\tau = 1, 2, 3, \dots$ , which assigns tuples into any given  $\text{AmTree}^{(p)}$ . Figure 4 illustrates a snapshot of this multi-AmTree structure after tuple with  $\tau = 1024$  has been inserted. Assuming tuples of distinct timestamps, items with  $\tau = 1, 3, 5, 7, \dots$  are inserted into  $\text{AmTree}^{(0)}$ , items with  $\tau = 2, 6, 10, 14, \dots$  get inserted into  $\text{AmTree}^{(1)}$ , those with  $\tau = 4, 12, 20, 28, \dots$  are consumed by  $\text{AmTree}^{(2)}$ , etc. A new AmTree is initiated whenever a tuple of timestamp equal to a power of 2 is received. So, another AmTree will be created as soon as tuple of  $\tau = 2048$  arrives.

Essentially, this hash function demultiplexes the incoming stream into several substreams consumed by diverse trees. In fact, the first  $\text{AmTree}^{(0)}$  is updated every second tuple consuming half of the incoming items, the second  $\text{AmTree}^{(1)}$  is updated every fourth tuple consuming a quarter of the total items and so on. Clearly,  $\text{AmTree}^{(0)}$  evolves rapidly,  $\text{AmTree}^{(1)}$  is updated more slowly (at the half rate) and so on. Thus, any given  $\text{AmTree}^{(p)}$  has one level less compared to its predecessor  $\text{AmTree}^{(p-1)}$ . The overall space complexity of AmTreeH is  $O((\log N - 1) + (\log N - 2) + \dots + 1) = O((\log N - 1) \cdot \frac{1 + \log N - 1}{2}) = O(\log^2 N)$ . Meanwhile, amortized update complexity still remains  $O(1)$  per tuple, since each item is inserted into exactly one AmTree.

Not only does this scheme provide more dense approximations, but it may be also valuable for multi-resolution stream representations. Thus, a query may

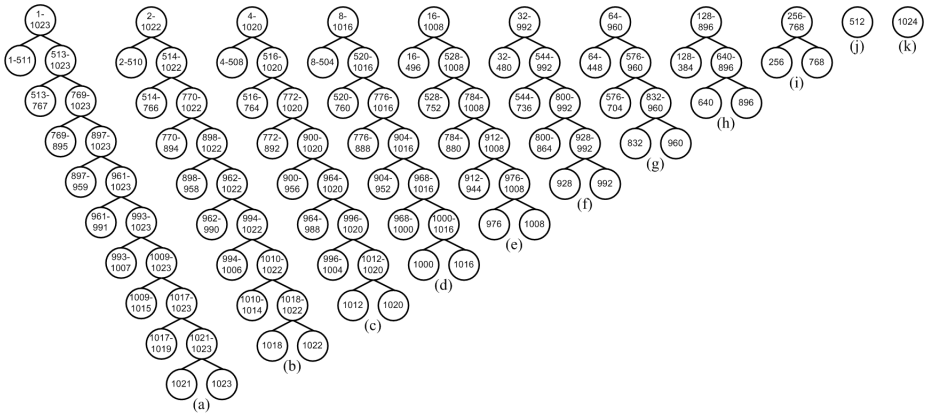


Fig. 4. Multiple concurrent AmTrees

be given a quick approximate answer using data from the last *AmTree* in the forest. Upon user request, further refinements may be retrieved from other trees with more levels, gradually enhancing result with extra details.

## 4 Amnesic Trajectory Synopses

So far, we have presented our generic summarization structure, but we believe that *AmTree* is best suited for summarizing streams of sequential features, i.e., time series that must retain contiguity among their consecutive elements. This is exactly the case of streaming locations that track the movement of many moving objects across time. In this section, we will study amnesic synopses concerning singleton trajectories, before proceeding to give a technique for computing aggregates over moving objects in Section 5.

### 4.1 Linear Representation of Trajectories

Consider a set of numerous point objects continuously moving over the Euclidean 2-dimensional plane. Obviously, the evolving sequence of locations recorded for each object constitutes its trajectory. Since it is not realistic to maintain a continuous trace of each object, only point samples can be practically collected from the respective data source at distinct time instants (e.g., every few seconds a car sends its location measured by a GPS device).

Therefore, *trajectory*  $T$  of a point object moving over the Euclidean plane is a possibly unbounded sequence of timestamped locations across time. Thus, we consider a sequence of tuples  $\langle oid, \tau_i, x_i, y_i \rangle$ ,  $i = 1, \dots, N, \dots$ , assuming that for an object with identity  $oid$ , spatial position  $(x_i, y_i)$  has been recorded at timestamp  $\tau_i$ . Note that this representation implies that a trajectory is essentially a collection of points put in order by their timestamp values.

Instead of such a “chain” of points, a linear representation of each trajectory is sometimes more adequate, since it conveys the sense of a continuous, though

approximate, trace of its movement. Such a polyline is composed of consecutive line segments, each one connecting a pair of successive point locations recorded for this object. Effectively, each line segment indicates the *displacement* of an object between two successive recordings, that is, it reveals the change of its position with respect to its previously known location. More concretely, let two successive positional updates  $\langle oid, \tau_i, x_i, y_i \rangle, \langle oid, \tau_j, x_j, y_j \rangle$  of the same moving object  $oid$ , where  $\tau_i < \tau_j$  and  $\exists \langle oid, \tau_k, x_k, y_k \rangle, \forall \tau_k \in (\tau_i.. \tau_j)$ . Then, we can calculate in constant time the displacements  $dx = x_i - x_j$  and  $dy = y_i - y_j$  during time interval  $[\tau_i.. \tau_j]$  along axes  $x, y$  respectively. This process can be applied in an incremental fashion for each object; every time a positional update arrives, this calculation requires just the (already maintained) last known location.

Therefore, as an alternative representation to a time series of points, a trajectory can be trivially transformed to a sequence of displacements of the form  $\langle oid, \tau_{start}, \tau_{end}, dx, dy \rangle$ . In fact,  $\tau_{start} = \tau_i$  and  $\tau_{end} = \tau_j$  denote the bounds of the corresponding time interval, but, for storage savings, we may even omit attribute  $\tau_{end}$  from the schema of tuples, thanks to contiguity property.

## 4.2 Updating Trajectory Synopses

For compressing a single trajectory, we suggest an instantiation of AmTree that manipulates all successive displacement tuples recorded for this object. In direct correspondence to AmTree functionality presented in Section 3.1, mapping  $f$  converts each current position  $\langle oid, \tau_{cur}, x_{cur}, y_{cur} \rangle$  into a displacement tuple  $\langle oid, \tau_{prev}, \tau_{cur}, dx, dy \rangle$ . This tuple is then inserted into node  $R_0$ , possibly triggering further updates at higher levels as mentioned in Section 3.2.

When the contents of level  $i$  must be *merged* to produce a coarser representation, a function  $g$  must be invoked in order to combine the displacements stored in nodes  $L_i$  and  $R_i$ . In the case of trajectories, this can be handled as a *concatenation* of two trajectory segments that span consecutive time intervals of identical size. More formally, let two displacement tuples  $l_1, l_2$  concerning the same object  $oid$  with  $\tau_{end}^{(1)} = \tau_{start}^{(2)}$ , that are stored in nodes  $L_i$  and  $R_i$ , respectively. Function  $g(l_1, l_2)$  is used to concatenate them into a single tuple  $l \equiv l_1 \circ l_2 = \langle oid, \tau_{start}^{(1)}, \tau_{end}^{(2)}, dx^{(1)} + dx^{(2)}, dy^{(1)} + dy^{(2)} \rangle$ . This concatenation inevitably leads to a loss of information, since the common articulation point of segments  $l_1$  and  $l_2$  is removed altogether, i.e., the point location that was recorded at timestamp  $\tau_{end}^{(1)} = \tau_{start}^{(2)}$ . In effect, the resulting displacement  $l$  is defined solely by the two non-common endpoints of  $l_1, l_2$ , which are exactly the locations recorded at timestamps  $\tau_{start}^{(1)}$  and  $\tau_{end}^{(2)}$ . This process can be easily generalized to  $k \geq 2$  consecutive displacement tuples  $l_1, l_2, \dots, l_k$ , retaining only the first and the last point location in this subsequence.

It must be stressed that endpoints of all displacements stored in AmTree nodes correspond to original positional updates, while displacements remain connected to each other at every level. However, as trajectory information propagates into higher levels of the tree, line segments of increasing temporal extent and less positional accuracy get created by progressively eliminating intermediate point

locations. Thus, when going higher in each tree snapshot, more rough segments are derived for the more distant trajectory parts, while more precise segments at the lower levels correspond to recent portions of its movement. Evidently, an amnesic behavior is realized for trajectory segments through levels of gradually less detail in such bottom-up tree maintenance.

### 4.3 Reconstructing Trajectories from Synopses

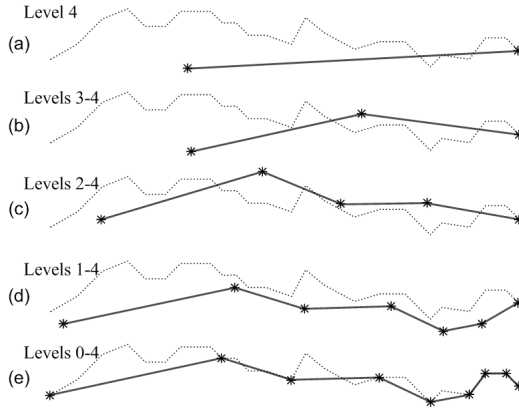
Trajectory synopses accomplished by AmTree can give a reduced representation of an object's movement utilizing several tree traversal schemes. As long as consecutive displacements are preserved, we are able to reconstruct the movement of a particular object starting from its most recent position and going steadily backwards in time by choosing points in descending temporal order. Any trajectory reconstruction process can be gradually refined by combining information from multiple levels and nodes of the tree.

A simple reconstruction scheme would be a tree traversal in a bottom-up fashion, starting from node  $R_0$  and then accessing right and left nodes at each level. Yet, many nodes that do not contribute any additional points will also be visited. In the snapshot at Fig. 3a, none of the  $L_i$  nodes will provide any additional point, since their content is already merged into their parent  $R_{i+1}$ .

By construction, all  $R_i$  nodes contain displacements concerning gradually aging positions of an object, as their time granules span more extended periods in the past. Besides, each new location does not necessarily incur changes to all tree levels, but up to a maximum level  $M$  that depends on the arrival order of this point. But how many of those  $L_i$  nodes should be used to further refine any trajectory approximation? Further, which is the minimal set of these nodes?

The important observation is that tree updates take place in waves that involve levels up to  $M$ . Each such wave synchronizes these levels to the current timestamp value, but nodes at levels higher than  $M$  remain completely untouched, meaning that the granules of  $R_M$  and  $R_{M+1}$  are time intervals that do not end up at the same timestamp value. Thus, after each insertion, consecutive tree levels are clustered into groups according to the time instant they were updated for the last time, as shown in Fig. 3g. Intuitively, if we were able to locate at which level each successive wave has finished its updates, we could get the required  $L_i$  nodes as these are the upper left nodes of each such cluster.

Let  $c$  denote the number of  $L_i$  nodes that are able to contribute to trajectory reconstruction. Not surprisingly,  $c$  can be easily computed from the number of 1's in the binary representation of the current number  $N$  of positional updates received thus far for this particular object, after excluding the most significant bit (since always MSB=1). Referring back to Fig. 3g, when point with timestamp  $\tau = 21$  arrives, it is the 22nd tuple in the sequence; hence,  $N = 22$  and its binary representation (10110) contains three 1's, so  $c = 2$  after excluding the MSB. It is precisely the position of those 1's in this binary representation that dictates the level of the  $L_i$  nodes that suffices to be visited, i.e., left nodes at levels 1 and 2. Hence, for reconstructing the trajectory that corresponds to the



**Fig. 5.** Multiple resolutions of a trajectory

snapshot of Fig. 3g, apart from all  $R_i$ , nodes  $L_1$  (18-19) and  $L_2$  (12-15) should also be taken.

In total, all  $H$  (height of the tree) of the  $R_i$  nodes and a number  $c$  of the  $L_i$  nodes can provide all knowledge contained in any AmTree state. This reconstruction process will produce  $H + c + 1$  points, because there are  $H + c$  consecutive segments in that approximation. This simple calculation gives 6 points for the state at  $\tau = 15$  (Fig. 3a), while it yields 9 points when  $\tau = 22$  (Fig. 3h).

#### 4.4 Multi-resolution Trajectory Approximation

The aforementioned bottom-up approach will retrieve all available positional data stored in an AmTree. Instead of such an exhaustive process, we will describe another reconstruction policy that can produce amnesic trajectory approximations at multiple resolutions. Clearly, trajectory segments stored in successive tree levels are connected to each other. Therefore, if we take any number of the higher levels of the tree, we can still reconstruct an uninterrupted representation of the movement for the corresponding time intervals. Essentially, we still apply the bottom-up technique of Section 4.3, but for a restricted number of levels.

If this process is repeated  $H$  times, each time getting another level lower in the hierarchy, we will be able to get multiple trajectory approximations, at progressively finer resolutions for the most recent portions, as illustrated in Fig. 5. The initial approximation is performed using only the upmost level of the tree depicted in Fig. 3h. Since the point at hand is the last seen location, we get the other point from the displacement available in node  $R_4$ . Note that, since the tree is not entirely synchronized at this timestamp, this procedure will not yield the actual location of the moving object 16 timestamps ago (Fig. 5a). The second approximation will include information from level 3 as well, and will be slightly better (Fig. 5b). The final approximation (Fig. 5e) retrieves all information currently summarized in the tree and the locations it yields are the original positions at the respective timestamps.

Multi-resolution reconstruction is application dependent, enabling fast estimations from coarse trajectory representations. Critical features (e.g., movement orientation) can be quickly conjectured by inspecting just a few upmost levels.

## 5 Computing Aggregates over Moving Objects

In this section, we present a summarization technique that provides unbiased estimates about the number of objects moving in an area of interest during a specified time interval. When each object must be counted only once, the problem is known as *distinct counting* [12]. We introduce an amnesic treatment for this problem, utilizing the AmTree structure with a powerful sketching algorithm over a static spatial decomposition.

### 5.1 Overview of Flajolet-Martin (FM) Sketches

FM sketches [6] were designed to approximate the zeroth frequency moment ( $F_0$ ), which actually provides an estimate for the number of *distinct objects* in a multiset. They have since widely used in stream summarization (e.g., [7][12]).

As described in [12], an FM sketch is a bit-vector consisting of  $r$  bits, all of them initially set to 0. The appropriate size of this bitmap is  $O(\log_2 DC)$ , where  $DC$  is an upper bound on the number of distinct objects. FM algorithm employs a hash function  $h$  that maps an object identity  $oid$  into a pseudo-random integer  $h(oid)$  with a geometric distribution, meaning that  $Pr[h(oid) = v] = 2^{-v}$ , where  $v \in [1, r]$ . For every object  $oid$  in the multiset, the algorithm sets the  $h(oid)$ -th bit of the FM sketch to 1. Regardless of how many times object  $oid$  is found in the set, the same bit will always be switched on; this fact establishes the duplicate-insensitivity property of FM sketches. No matter in which order objects will appear, the same bitmap will be eventually created. For  $n$  distinct objects, it is expected that  $n/2$  of them will map to bit 1,  $n/4$  objects go to bit 2, and so on.

It turns out that, if we locate in the sketch vector its first bit  $k$  that is still 0, we can get a good unbiased estimate of  $n$  as  $1.29 \cdot 2^k$ . To improve probabilistic confidence, Flajolet and Martin [6] suggested the use of  $m$  independent sketches, each with its own hash function, and finally averaging all results to estimate  $n$ . Furthermore, to keep processing cost per object to  $O(1)$  instead of  $O(m)$ , Probabilistic Counting with Stochastic Averaging (PCSA) was proposed, which utilizes  $m$  atomic FM sketches. Every time an update is due, only one of these sketches is chosen using another hash function  $h'(oid)$ . Thus, every sketch keeps count for about  $\frac{n}{m}$  distinct objects. Assuming that  $k_1, k_2, \dots, k_m$  are the first non-zero bits in each of the  $m$  atomic sketches, the distinct count is estimated to  $n = 1.29 \cdot m \cdot 2^{\frac{1}{m} \sum_{i=1}^m k_i}$ , with expected standard error  $O(m^{-\frac{1}{2}})$  and overall space complexity  $O(m \log_2 DC)$ .

Another important property of FM sketches is that they can be *composed* from other partial FM sketches. If a sketch  $FM_A$  is maintained over multiset  $A$  and a sketch  $FM_B$  is independently computed over multiset  $B$ , we can get the FM sketch of  $A \cup B$ , by applying a bit-wise OR over their bitmaps (and between respective sketches for the PCSA variation), i.e.,  $FM_{(A \cup B)} = FM_A \vee FM_B$ .



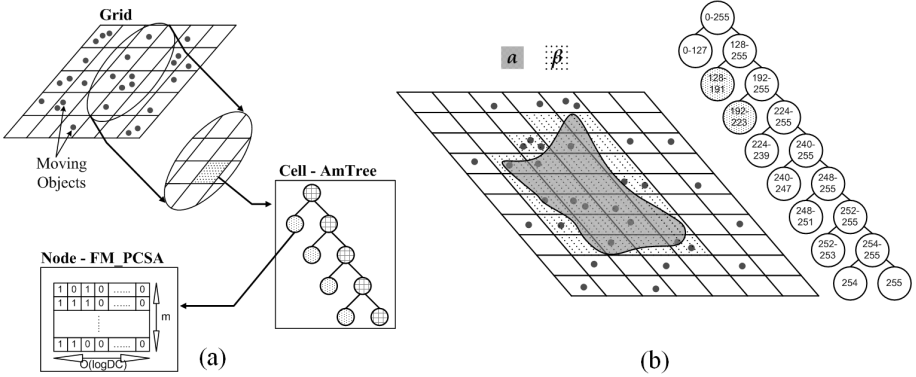


Fig. 6. (a) The 3-tier FM-AmTree structure (b) Evaluating a range query

### 5.2 Applying 3-Tier Compression

We consider queries of the form “How many distinct moving objects have been observed in area  $\alpha$  during time interval  $\Delta\tau$ ?”. In our terminology,  $\alpha$  is the *spatial extent* of the query region defined as a 2-d polygonal area, while  $\Delta\tau$  is the *temporal extent*. The distinguishing feature of our approach is that we allow an amnesic behavior to the resulting estimations: as long as the temporal extent remains close to the current time, it is probable that certain time intervals stored in the AmTree can approximate it with more precision. To this end, we introduce a 3-tier structure that accomplishes compression at three levels (Fig. 6a):

1. We utilize a regular decomposition of the 2-d spatial plane into equal-area cells. If  $HG$  and  $VG$  are respectively the number of subdivisions along axes  $x, y$  of this grid partitioning, then a set of  $HG \cdot VG$  cells is used to maintain a simplified spatial reference of moving objects instead of their actual locations.
2. To accommodate temporal extents, each cell points to an AmTree, which maintains gradually aging count of objects within that cell.
3. Query-oriented compression is achieved using FM\_PCSA sketches. Each node of an AmTree corresponds to  $m$  bitmap vectors utilized by the FM\_PCSA sketch. Hence, we avoid enumeration of objects, as we are satisfied with an acceptable estimate of their distinct count given by the sketching algorithm.

Assuming that  $N$  locations have been received for each object and that  $DC$  is an upper bound of the distinct objects, it is easy to see that the overall space complexity is  $O(HG \cdot VG \cdot \log N \cdot m \cdot \log DC)$ . Beyond its small memory footprint, this technique is well suited to a streaming context, due to its low update cost.

Indeed, when a positional update of an object arrives for processing, initially its location  $(x, y)$  is hashed against the spatial grid and the appropriate cell is identified. Then, the FM sketch linked to node  $R_0$  of the corresponding AmTree must be updated. This is carried out by hashing the object’s id over this FM

sketch. When updates must propagate to higher levels of the tree, merging function  $g$  takes advantage of the ability to compose partial FM sketches into a single sketch. Thus, the union of sketches  $FM_{R_i} \cup FM_{L_i}$  at the same level  $i$  is assigned to node  $R_{i+1}$ . For synchronization, AmTree updates are performed when all point locations for the current timestamp  $\tau$  have arrived, assuming that each object discloses its position at every  $\tau$ . In total,  $HG \cdot VG$  trees need to be updated at each timestamp, so update complexity is  $O(HG \cdot VG)$  per timestamp.

### 5.3 Estimating Count of Distinct Objects

In order to estimate the number of distinct objects moving within area  $\alpha$  during time interval  $\Delta\tau$ , we initially start by identifying which cells of the grid partitioning completely cover region  $\alpha$ . As shown in Fig. 6b, the cells overlapping with polygon  $\alpha$  constitute a greater area  $\beta$ . Next, those cells are used to determine the group of qualifying AmTree structures that maintain the aggregates. Corresponding nodes at every AmTree actually refer to identical time intervals, since all trees are concurrently updated. Thus, we need to locate the set of nodes that overlap time period  $\Delta\tau$  specified by the query; these nodes are the same for each qualifying tree. As illustrated in Fig. 6b, when the AmTree snapshot is traversed, nodes  $L_5$  of interval  $[192..223]$  and  $L_6$  of  $[128..191]$  are found to make up the minimal time period  $[128..223]$  that contains the query interval  $\Delta\tau = [135..220]$ . This means that we should access only the FM sketches linked to these two nodes in each of the AmTrees qualifying for area  $\beta$ . By taking the union of all those sketches (i.e., an OR operation over the respective bitmaps), we finally get an approximate answer to our query.

Clearly, the response given by this algorithm is an overestimation of the actual distinct count, as it is influenced by the approximations performed in each tier. First, the degree of grid partitioning into cells apparently controls the spatial resolution of aggregates. Second, the temporal granularity determines the aging rate inherent in the AmTree mechanism, and consequently the extent of time intervals maintained. Last but not least, the number of bitmaps in each sketch affect the precision of the estimated counts. It goes without saying that there is a trade-off between processing time and each of these three parameters that guide response accuracy, a fact empirically confirmed in the experiments.

## 6 Experimental Evaluation

Next, we report on an experimental validation of AmTree when utilized (i) to approximate trajectories of moving objects (Section 4) and (ii) to provide estimates for distinct count queries (Section 5), always working in main memory.

All experiments were performed on an Intel Pentium-4 2.5GHz CPU running WindowsXP with 512 MB of main memory. We generated synthetic datasets for trajectories of objects moving at various speeds along the road network of greater Athens (an area of about 250 km<sup>2</sup>). Geometric nodes of the road segments were randomly chosen as origin and destination points for a shortest path algorithm. Finally, 500 sample points were taken for each calculated route.



**Fig. 7.** Approximation quality for trajectories

We run simulations for several time intervals (up to 500 timestamps) and number of objects (up to 20,000). The interarrival time of streaming tuples was constant for all trajectories, assuming that all objects reported their positional updates concurrently at regular time intervals. This is actually the most intensive situation, since agility of objects was set to 100%.

### 6.1 Quality of Trajectory Approximation

In this set of experiments, we maintained an AmTree structure for each trajectory from a set of 1000 moving objects, each recorded for 500 timestamps. To assess approximation quality, we utilized spatiotemporal range queries that continuously return objects contained in a given spatiotemporal hyper-rectangle. We defined 10 static spatial rectangles, each covering about 1% of the total space, and at every 8 timestamps we observed which approximate trajectories fell within the spatial area. Hence, temporal extents proceed by 8 timestamps each time, so that each window spans the entire movement history thus far.

We counted qualifying trajectories in each query region and results were averaged for each temporal interval. Then, we compared these estimations against exact answers based on original trajectories. We measured approximation quality as  $\rho = \frac{TP}{P+FN}$ , where  $TP$  denotes the number of *true positives*,  $P$  is the number of (false and true) *positives*, whereas  $FN$  signifies the number of *false negatives*.

We calculated values of  $\rho$  as plotted in Fig. 7a, where the  $x$ -axis of the diagram shows the query intervals used, i.e., the size of landmark windows. As expected, the quality of approximation deteriorates as the query interval becomes gradually larger due to the time-decaying positional accuracy of the trajectory segments. As stated in Section 4.3, the most remote trajectory segments progressively obtain an increasingly rough outline maintaining a reduced number of the original points. Although recent segments are always more accurate, overall error heavily depends on the temporal extent in the past, since we steadily get away from the “landmark” time instant the query was issued.

Notice that abrupt drops in approximation quality occur at timestamp values that correspond to major updates in AmTree, i.e., changes across the majority of tree levels. When all nodes get synchronized, segments from the right nodes suffice to provide all information contained in the tree. Subsequently, changes

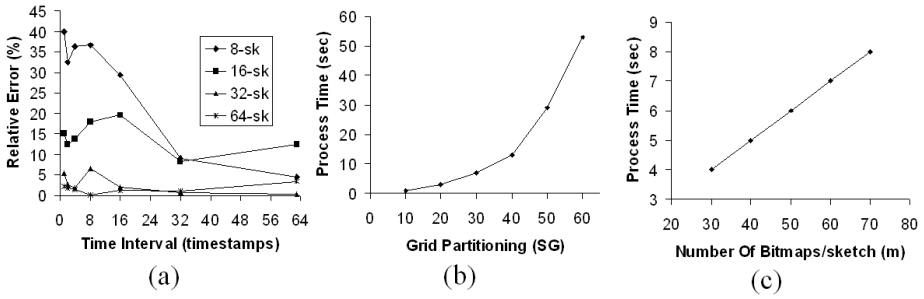


Fig. 8. Quality of aggregate estimation

occur at waves that always propagate to a higher level until a complete update happens again. Recall from Section 4.3 that, at snapshots where more tree levels have been updated at diverse waves, the number  $H + c$  of reconstructed segments has a local maximum, so we can get more additional points by taking advantage of information stored in left nodes. This is reflected in the sporadic “spikes” in Fig. 7a that signify trajectory representations with maximal number of points.

Fig. 7b shows approximation quality versus the compression rate achieved. Compression rate denotes the ratio in the size of an approximated trajectory with respect to the original, i.e., the percentage of original points retained in the coarsened trajectory. Not surprisingly, approximation quality decreases for more compressed trajectories, since more points are discarded. As the temporal extent gradually increases, the amnesic behavior of AmTree gets more pronounced and a trajectory obtains exponentially more compressed representation.

Still, even for heavily compressed trajectories (rates close to 2%), accuracy index  $\rho$  is quite satisfactory (about 93.5%). For small compression rates, the additional deviation is attributed to trajectory segments far away from the current time, i.e., those that have been consistently reduced over and over again, eventually devoid of much positional information. In contrast, answers reconstructed from most recent time intervals are remarkably more accurate, since the respective segments have not been altered much yet.

## 6.2 Quality of Distinct Count Estimates

To validate the algorithm for estimating aggregates, we utilized a dataset containing locations of 20,000 objects, each recorded for 100 timestamps. We assumed square-shaped grid cells, so the degree of spatial decomposition  $SG$  is a single parameter for both axes  $x, y$ , i.e.,  $SG = HG = VG$ .

In Fig. 8a we illustrate the effect of the number  $m$  of bitmaps per FM\_PCSA sketch on relative error, with respect to distinct count queries for various temporal extents, fixing  $SG = 10$ . Quite predictably, the more bitmaps allocated to each sketch, the more confidence is attained for the estimates. While for  $m = 8$  estimation quality was off even by 40%, for  $m = 64$  the relative error was less than 4%. Further increase in  $m$  did not seem to yield better results.

In Fig. 8b we plot the total update cost against the spatial decomposition  $SG$  at each dimension, for  $m = 32$  sketches. Note that the cost is quadratic to the number of cells, i.e., a finer grid partitioning incurs more processing time at the expense of increased accuracy. Besides, computation over the entire stream of locations (and the update cost per tuple as well) is linear to the number  $m$  of sketch bitmaps, as depicted in Fig. 8c, when  $SG = 10$ .

## 7 Related Work

The notion of time-decaying significance in data items is widespread in streaming applications, such as telecom usage patterns or connections through Internet gateways. Aging-aware computation of aggregates and statistics over a stream was first formalized in [5]. Several types of time-decaying functions were defined and theoretically analyzed in terms of their storage requirements to fine-tune the rate of decay, considering exponential, polynomial and sliding-window variants.

*Amnesic* approximation of streaming time series was set forth in [9], offering the intuitive idea that data can be approximated with a precision proportional to its age. A taxonomy of amnesic functions was provided, along with online algorithms that can incrementally compute reduced representations of scalar time series with time complexity independent of the total stream size. Our approach falls naturally under this generic framework, as an amnesic structure with exponential decay especially tailored for streaming positional updates.

AmTree scheme extends the time-varying SWAT mechanism proposed in [3], which produces multi-resolution approximations of a data stream. Despite its external similarity to SWAT, AmTree is more generic in certain aspects. First, SWAT is intrinsically bound to wavelet transform of scalar stream values, whereas AmTree can even handle multi-dimensional points with user-specified approximation functions. Further, in functionality terms, at each tree level we eliminate the intermediate node, which SWAT utilizes for maintenance of transient values. Finally, we can effectively tune approximation accomplished at each level, by properly controlling the number of granules (i.e. tree nodes).

A multi-resolution tree structure was introduced in [8] for computing aggregates (SUM, COUNT, etc.) over multi-dimensional points, by employing typical spatial indices (e.g., quad-trees, R-trees). Still, this technique cannot cope with streaming data that arrive at high rates, while it cannot easily accommodate the sequential nature of positional updates. Temporal aggregation over streams was also studied in [13], but the emphasis was mainly on indexing schemes that could be dynamically maintained using multiple time granularities. Efficient exact computation of aggregates over sliding windows is tackled in [1] with a particular interest in sharing resources for numerous continuous queries over scalar stream values. In contrast, our concern is to maintain age-biased synopses to get reduced representations, rather than just computing aggregate statistics.

The distinct-count problem over moving objects was tackled in [12]. Their solution applies to spatiotemporal databases and builds a sketch index [6] equipped with R-trees to maintain spatial regions and B-trees to host temporal intervals.

This scheme maintains all successive stages of aggregation without any notion of time-decay; instead, our 3-tier approach is geared towards a streaming model.

In [10] we proposed two single-pass sampling techniques to achieve effective trajectory compression, exploiting spatial locality and temporal timeliness inherent in trajectory streams. The underlying principle of those heuristics was that speed and orientation of each object play an important role in deciding which samples to retain. Departing from this data-driven approach, in this paper we elaborate on our ideas briefly outlined in [11]. Thus, we focus on aging-aware schemes where all stream tuples have a lifespan, instead of discarding data upon admission to the system as carried out with a sampling technique.

## 8 Conclusion

In this paper, we presented a hierarchical structure called AmTree, capable of maintaining multiple-granularity approximations over streaming locations of numerous moving objects. This method works in an online amnesic fashion, by reserving more precision for the most recent items while accepting increasing error for gradually aging stream portions. We empirically confirmed that this time-decaying approach can effectively cope with online trajectory approximation, also providing affordable estimates for distinct count spatiotemporal queries.

In the future, we intend to study other types of amnesic behavior adjusting this structure to deal with user-specified aging patterns. In terms of answering queries over positional streams, we plan to explore spatiotemporal associations, such as the aging rate of query intervals or topological relations between moving objects and query regions, in order to better assess approximation quality.

## References

1. Arasu, A., Widom, J.: Resource Sharing in Continuous Sliding-Window Aggregates. In: VLDB, pp. 336–347 (2004)
2. Bettini, C., Dyreson, C.E., Evans, W.S., Snodgrass, R.T., Wang, X.S.: A Glossary of Time Granularity Concepts. In: Etzion, O., Jajodia, S., Sripada, S. (eds.) Temporal Databases: Research and Practice. LNCS, vol. 1399, pp. 406–413. Springer, Heidelberg (1998)
3. Bulut, A., Singh, A.K.: SWAT: Hierarchical Stream Summarization in Large Networks. In: ICDE, pp. 303–314 (2003)
4. Chen, Y., Dong, G., Han, J., Wah, B.W., Wang, J.: Multi-Dimensional Regression Analysis of Time-Series Data Streams. In: VLDB, pp. 323–334 (2002)
5. Cohen, E., Strauss, M.: Maintaining Time-Decaying Stream Aggregates. In: PODS, pp. 223–233 (2003)
6. Flajolet, P., Martin, G.N.: Probabilistic Counting Algorithms for Database Applications. *Journal of Computer and Systems Sciences* 31(2), 182–209 (1985)
7. Ganguly, S., Garofalakis, M., Rastogi, R.: Processing Set Expressions over Continuous Update Streams. In: SIGMOD, pp. 265–276 (June 2003)
8. Lazaridis, I., Mehrotra, S.: Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure. In: SIGMOD, pp. 401–412 (May 2001)

9. Palpanas, T., Vlachos, M., Keogh, E., Gunopulos, D., Truppel, W.: Online Amnesic Approximation of Streaming Time Series. In: ICDE, pp. 338–349 (2004)
10. Potamias, M., Patroumpas, K., Sellis, T.: Sampling Trajectory Streams with Spatiotemporal Criteria. In: SSDBM, pp. 275–284 (2006)
11. Potamias, M., Patroumpas, K., Sellis, T.: Amnesic Online Synopses for Moving Objects. In: CIKM, pp. 784–785 (2006)
12. Tao, Y., Kollios, G., Considine, J., Li, F., Papadias, D.: Spatio-Temporal Aggregation Using Sketches. In: ICDE, pp. 214–226 (2004)
13. Zhang, D., Gunopulos, D., Tsotras, V.J., Seeger, B.: Temporal Aggregation over Data Streams using Multiple Granularities. In: Chaudhri, A.B., Unland, R., Djeraba, C., Lindner, W. (eds.) EDBT 2002. LNCS, vol. 2490, pp. 646–663. Springer, Heidelberg (2002)

# Spatial Partition Graphs: A Graph Theoretic Model of Maps

Mark McKenney and Markus Schneider\*

University of Florida, Department of Computer and Information Sciences  
{mm7,mschneid}@cise.ufl.edu

**Abstract.** The notion of a *map* is a fundamental metaphor in spatial disciplines. However, there currently exist no adequate data models for maps that define a precise spatial data type for *map geometries* for use in spatial systems. In this paper, we consider a subclass of map geometries known as *spatial partitions* that are able to model maps containing region features. However, spatial partitions are defined using concepts such as infinite point sets that cannot be directly represented in computers. We define a graph theoretic model of spatial partitions, called *spatial partition graphs*, based on discrete concepts that can be directly implemented in spatial systems.

## 1 Introduction

In spatially oriented disciplines such as geographic information systems (GIS), spatial database systems, computer graphics, computational geometry, computer vision, and computer aided design, a *map* is a fundamental metaphor. Many of these systems fundamentally represent space through more primitive concepts, such as points, lines, or regions, which we denote the *traditional spatial types*. These more basic representations of space are often combined to form maps in many applications. Furthermore, in applications such as GIS, global positioning systems, and navigation systems, the map itself tends to play a central role in that the map is the primary user interface tool. Thus, users tend to manipulate data in terms of maps instead of the underlying types of point, line, and region.

Despite the intimate ties between maps and a wide variety of spatial applications, spatial systems tend to represent space as collections of individual points, lines, and regions, and utilize maps as a visualization tool for these more simple types. For example, a system may display a collection of regions to a user as a map, but the system itself can only process space based on points, lines, and regions. Thus, a map in this sense is not being used as a representation of space, but as a *visualization* of more basic spatial entities. However, treating a map simply as a visualization gives rise to problems at both the conceptual and implementation levels. At the conceptual level, maps must conform to some set of properties in order to handle certain configurations of map components.

---

\* This work was partially supported by the National Science Foundation under grant number NSF-CAREER-IIS-0347574.



Consider two regions  $r$  and  $s$  such that  $s$  fits completely in the interior of  $r$ . If a user attempts to construct a map visualization of these regions, then it is possible that  $s$  will be completely hidden by  $r$ , indicating that some set of properties is required to govern the construction of maps. Therefore, even when using maps only as visualizations, a type definition of maps is required that poses constraints on the map contents and indicates how situations such as this one should be handled. Furthermore, map operations are not purely geometric. For example, components of maps are typically labeled with thematic information (e.g., names of states, cities, rivers, etc.). When considering the intersection of two maps, not only must a spatial intersection operation be defined, but some method of labelling the resulting map must be defined too that takes into account the labels of the argument maps.

At the implementation level, systems that utilize maps as visualization tools cannot take advantage of map properties in algorithms. For example, maps implicitly model topological relationships such as the adjacency between regions. Thus querying a map for neighborhoods of adjacent regions or regions that are disjoint from a query region is intuitive and simple in the map context. Given a spatial system that cannot take advantage of map properties, maps must be simulated as collections of other spatial objects and such topological information must be calculated explicitly. Additionally, when implementing operations such as map intersection in such systems, a Cartesian product of both collections of spatial objects must be computed.

In this paper, we take the view that maps themselves are a *spatial data type*, and not merely a visualization tool. We view a map as a *map geometry*, i.e., as a spatial data type with its own type definition and operations. The idea of using map geometries to represent space in spatial systems aligns nicely with the current practice of having users interact with spatial systems through maps. As was mentioned before, maps implicitly model topological relationships between the spatial components, such as regions, within the map. Furthermore, the use of map geometries in spatial systems unifies the user interface model and the data model such that user queries can be computed over maps.

Despite the advantages that processing space in map form provides for spatial systems, research into modeling maps has not resulted in an adequate data model for maps (see Section 2). Current models define maps as sets of traditional spatial types. These models lack a mechanism to enforce constraints over the traditional types, and tend to lack closure properties. Conceptual models exist that define abstract mathematical models of map geometries that are able to address problems other models have with constraint enforcement and closure concerns; however, such models rely on concepts such as infinite point sets and mappings that cannot be directly implemented in computer systems.

In this paper, we only consider a particular subclass of map geometries that we call *spatial partitions*. A spatial partition is a subdivision of the plane into regions such that each region has a *label* that identifies it and its characteristic thematic properties, and regions either *meet* or are *disjoint* with one another. We do not consider map geometries that contain features such as line networks or spatial

point entities, but leave this to future work. By considering map geometries as spatial partitions, we are able to model a map as a single entity containing regional features satisfying particular topological constraints and define operations and predicates over it. We provide the formal definition of spatial partitions in Section 4. However, this definition is not sufficient for modeling map geometries in spatial systems because it defines an abstract model of spatial partitions.

The main contribution of this paper is the unique characterization of a spatial partition as a spatially embedded graph called a *spatial partition graph*. We define the type of spatial partition graphs along with constraints for these graphs that are enforced implicitly by the data type. Furthermore, we define spatial partition graphs at the discrete level, meaning that the definition avoids concepts such as infinite point sets that cannot be directly represented in computers. The result is a precise conceptual data type for map geometries that can be directly implemented in computer systems. Thus, our model resolves the existing conceptual and implementation level problems associated with current uses of map visualizations in spatial systems. Because we define a graph theoretic model, it is possible to directly utilize the many known graph algorithms for implementing operations such as reachability and connectivity. Finally, a discrete definition of map geometries provides a foundation upon which new algorithms for map operations can be specified.

In Section 2, we discuss related research into spatial data types for map geometries. The abstract model of spatial partitions, upon which we base our model of spatial partition graphs, is presented formally in Section 3. In Section 4, we define and discuss the properties of spatial partition graphs. Finally, in Section 5, we draw some conclusions.

## 2 Related Work

Research into spatial data types has focused on the development of simple and complex points, lines, and regions. Simple lines are continuous, one-dimensional features embedded in the plane with two endpoints; simple regions are two dimensional point sets that are topologically equivalent to a closed disc. Increased application requirements and a lack of closure properties of the simple spatial types lead to the development of the complex spatial types. In [1], the authors formally define complex data types, such as complex points (a single point object consisting of a collection of simple points), complex lines (which can represent networks such as river systems), and complex regions that are made up of multiple faces and holes (i.e., a region representing Italy, its islands, and the hole representing the Vatican). These types are defined based on concepts from point set topology, which allow the identification of the interior, exterior, and boundary of spatial the types. The notations  $S^\circ$ ,  $\partial S$ , and  $S^-$  respectively indicate the interior, boundary, and exterior of a point, line, or region spatial data type.

The idea of a *map* as a spatial data type has received significant attention in the literature. In [2,3,4,5], a map is not defined as a data type itself, but as a collection of spatial regions that satisfy some topological constraints.

Because these map types are essentially collections of more basic spatial types, it is unclear how the topological constraints can be enforced, and how thematic information can be effectively modeled. Furthermore, these models focus on an *implementation model* that can be directly incorporated into spatial systems while neglecting spatial data type considerations such as closure of maps under map operations. Other approaches [6,7] to defining map types have focused on raster or tessellation models. However, such approaches are not general enough for our purposes in the sense that the geometries of maps in these models are restricted to the tessellation scheme in use. In [8], the authors consider a map to be a planar subdivision; however, they do not discuss how a planar subdivision should be modeled except to say that data structures such as winged edge or quad edge structures should be used.

The work that comes closest to ours is [9,10] in which the authors consider modeling maps as special types of plane graphs. However, the authors of these works define such graphs based on modeling a map as a collection type consisting of spatial point, line, and region objects. Problems in the proposed methods arise when different spatial objects in the map share coordinates. For example, given the method of deriving a plane graph from a collection of points, lines, and regions, it is unclear if a spatial point object that has the same coordinates as the endpoint of a spatial line object in the plane graph can be distinguished. Furthermore, the authors require a separate structure to model what they term the *combinatorial structure* of a plane graph, which includes the topological relationships between different spatial components of the graph. Finally, the plane graph, as defined, is not able to model thematic properties of the map.

We base our work on the model of maps presented in [11]. The authors of this paper define an abstract, mathematical data model that formally describes the type of *spatial partitions*. A spatial partition is the partitioning of the plane into regular, open point sets such that each point set is associated with a label. The use of labels to identify point sets allows thematic information to be modeled explicitly in spatial partitions. Furthermore, operations are defined over spatial partitions, and it is shown that the operations are closed over the type of spatial partitions. A detailed description of the type of spatial partitions is provided in Section 3. The main drawback to this model is that it is based on the concepts of infinite point sets and mappings that are not able to be represented discretely.

### 3 The Spatial Partition Model

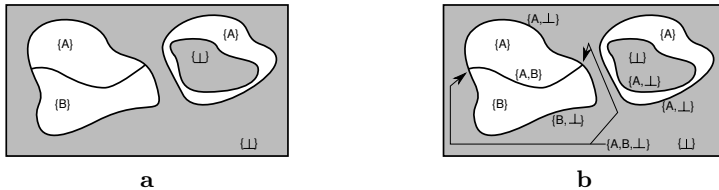
In this section, we review the definition of spatial partitions upon which we base our graph model of spatial partitions. We begin by providing a high level description of the type of spatial partitions that indicates their properties and introduces their terminology. We then introduce the mathematical notation and definitions required to formally define spatial partitions. Finally, the formal mathematical type definition of spatial partitions is presented.

### 3.1 Description of Spatial Partitions

A spatial partition, in two dimensions, is a subdivision of the plane into pairwise disjoint *regions* such that each region is associated with a *label* or *attribute* having simple or complex structure, and these regions are separated from each other by *boundaries*. The label of a region describes the thematic data associated with the region. All points within the spatial partition that have an identical label are part of the same region. Topological relationships are implicitly modeled among the regions in a spatial partition. For instance, neglecting common boundaries, the regions of a partition are always disjoint; this property causes maps to have a rather simple structure. Note that the *exterior* of a spatial partition (i.e., the unbounded face) is always labeled with the  $\perp$  symbol. Figure 1a depicts an example spatial partition consisting of two regions.

We stated above that each region in a spatial partition is associated with a single attribute or label. A spatial partition is modeled by mapping Euclidean space to such labels. Labels themselves are modeled as sets of attributes. The regions of the spatial partition are then defined as consisting of all points which contain an identical label. Adjacent regions each have different labels in their interior, but their common boundary is assigned the label containing the labels of both adjacent regions. Figure 1b shows an example spatial partition complete with boundary labels.

In [11], operations over spatial partitions are defined based on known map operations in the literature. It is shown that all known operations over spatial partitions can be expressed in terms of three fundamental operations: intersection, relabel, and refine. Furthermore, the type of spatial partitions is shown to be closed under these operations, indicating that the type of spatial partitions is closed under all known operations over them. In this paper, we only require the use of the refine operation, which is discussed later. We direct the reader to [11] for the definitions of intersection and relabel due to space constraints.



**Fig. 1.** Figure *a* shows a spatial partition with two regions and annotated with region labels. Figure *b* shows the same spatial partition with its region and boundary labels. Note that labels are modeled as sets of attributes in spatial partitions.

### 3.2 Notation

We now briefly summarize the mathematical notation used throughout the following sections. The application of a function  $f : A \rightarrow B$  to a set of values

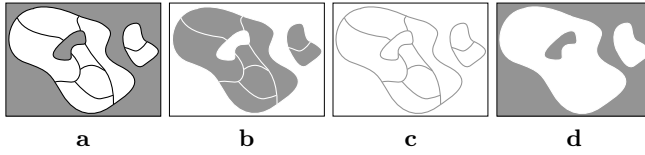
$S \subseteq A$  is defined as  $f(S) := \{f(x)|x \in S\} \subseteq B$ . In some cases we know that  $f(S)$  returns a singleton set, in which case we write  $f[S]$  to denote the single element, i.e.  $f(S) = \{y\} \implies f[S] = y$ . The inverse function  $f^{-1} : B \rightarrow 2^A$  of  $f$  is defined as  $f^{-1}(y) := \{x \in A|f(x) = y\}$ . It is important to note that  $f^{-1}$  is a total function and that  $f^{-1}$  applied to a set yields a set of sets. We define the range function of a function  $f : A \rightarrow B$  that returns the set of all elements that  $f$  returns for an input set  $A$  as  $rng(f) := f(A)$ .

Let  $(X, T)$  be a topological space [12] with topology  $T \subseteq 2^X$ , and let  $S \subseteq X$ . The *interior* of  $S$ , denoted by  $S^\circ$ , is defined as the union of all open sets that are contained in  $S$ . The *closure* of  $S$ , denoted by  $\overline{S}$  is defined as the intersection of all closed sets that contain  $S$ . The *exterior* of  $S$  is given by  $S^- := (X - S)^\circ$ , and the *boundary* or *frontier* of  $S$  is defined as  $\partial S := \overline{S} \cap \overline{X - S}$ . An open set is *regular* if  $A = \overline{A}^\circ$  [13]. In this paper, we deal with the topological space  $\mathbb{R}^2$ .

A *partition* of a set  $S$ , in set theory, is a complete decomposition of the set  $S$  into non-empty, disjoint subsets  $\{S_i|i \in I\}$ , called blocks: (i)  $\forall i \in I : S_i \neq \emptyset$ , (ii)  $\bigcup_{i \in I} S_i = S$ , and (iii)  $\forall i, j \in I, i \neq j : S_i \cap S_j = \emptyset$ , where  $I$  is an index set used to name different blocks. A partition can equivalently be regarded as a total and surjective function  $f : S \rightarrow I$ . However, a spatial partition cannot be defined simply as a set-theoretic partition of the plane, that is, as a partition of  $\mathbb{R}^2$  or as a function  $f : \mathbb{R}^2 \rightarrow I$ , for two reasons: first,  $f$  cannot be assumed to be total in general, and second,  $f$  cannot be uniquely defined on the borders between adjacent subsets of  $\mathbb{R}^2$ .

### 3.3 The Definition of Spatial Partitions

In [11], spatial partitions have been defined in several steps. First a *spatial mapping* of type  $A$  is a total function  $\pi : \mathbb{R}^2 \rightarrow 2^A$ . The existence of an undefined element  $\perp_A$  is required to represent undefined labels (i.e., the exterior of a partition). Definition 1 identifies the different components of a partition within a spatial mapping. The labels on the borders of regions are modeled using the power set  $2^A$ ; a *border* of  $\pi$  (Definition 1(ii)) is a block that is mapped to a subset of  $A$  containing two or more elements, as opposed to a *region* of  $\pi$  (Definition 1(i)) which is a block mapped to a singleton set. The *interior* of  $\pi$  (Definition 1(iii)) is defined as the union of  $\pi$ 's regions. The *boundary* of  $\pi$  (Definition 1(iv)) is defined as the union of  $\pi$ 's borders. The *exterior* of  $\pi$  (Definition 1(v)) is the block mapped  $\perp_A$ . As an example, let  $\pi$  be the spatial partition in Figure 1 of type  $X = \{A, B, \perp\}$ . In this case,  $rng(\pi) = \{\{A\}, \{B\}, \{\perp\}, \{A, B\}, \{A, \perp\}, \{B, \perp\}, \{A, B, \perp\}\}$ . Therefore, the regions of  $\pi$  are the blocks labeled  $\{A\}$ ,  $\{B\}$ , and  $\{\perp\}$  and the boundaries are the blocks labeled  $\{A, B\}$ ,  $\{A, \perp\}$ ,  $\{B, \perp\}$ , and  $\{A, B, \perp\}$ . Figure 2 provides a pictorial example of the interior, exterior, and boundary of a more complex example map (note that the borders and boundary consist of the same points, but the boundary is a single point set whereas the borders are a set of point sets).



**Fig. 2.** Figure *a* shows a spatial partition  $\pi$  with two disconnected faces, one containing a hole. The interior ( $\pi^\circ$ ), boundary ( $\partial\pi$ ), and exterior ( $\pi^-$ ) of the partition are shown in Figures *b*, *c*, and *d*, respectively. Note that the labels have been omitted in order to emphasize the components of the spatial partition.

**Definition 1.** Let  $\pi$  be a spatial mapping of type  $A$

- (i)  $\rho(\pi) := \pi^{-1}(\text{rng}(\pi) \cap \{X \in 2^A \mid |X| = 1\})$  (regions)
- (ii)  $\omega(\pi) := \pi^{-1}(\text{rng}(\pi) \cap \{X \in 2^A \mid |X| > 1\})$  (borders)
- (iii)  $\pi^\circ := \bigcup_{r \in \rho(\pi) \mid \pi[r] \neq \perp_A} r$  (interior)
- (iv)  $\partial\pi := \bigcup_{b \in \omega(\pi)} b$  (boundary)
- (v)  $\pi^- := \pi^{-1}(\{\perp_A\})$  (exterior)

A *spatial partition* of type  $A$  is then defined as a spatial mapping of type  $A$  whose regions are regular open sets [13] and whose borders are labeled with the union of labels of all adjacent regions. From this point forward, we use the term *partition* to refer to a spatial partition.

**Definition 2.** A *spatial partition* of type  $A$  is a spatial mapping  $\pi$  of type  $A$  with:

- (i)  $\forall r \in \rho(\pi) : r = \overline{r^\circ}$
- (ii)  $\forall b \in \omega(\pi) : \pi[b] = \{\pi[r] \mid r \in \rho(\pi) \wedge b \subseteq \partial r\}$

As was mentioned before, the type of spatial partitions is closed under all known partition operations. In this paper, we require the use of the *refine* operation. We provide a high-level description of the operation, and direct the reader to [11] for the formal definition. The *refine* operation uniquely identifies the connected components of a partition. Recall that two regions in a partition can share the same label if they are disjoint or meet at a point. Given a partition  $\pi$  containing multiple regions with the same label, the operation  $\text{refine}(\pi)$  returns a partition with identical structure to  $\pi$ , but with every region having a unique label. This is achieved by appending an integer to the label of each region that shares a label with another region. Figure 3 shows an example partition and the same partition after performing a *refine* operation. Note that the notation  $(A, 1)$  indicates that the integer 1 has been appended to label  $A$ .

The boundary of a spatial partition implicitly imposes a graph on the plane. Specifically, the boundaries form an undirected planar graph. The edges of the graph are the points mapped to the boundaries between two regions. The vertices of the graph are the points mapped to boundaries between three or more regions. We identify edges and vertices based on the cardinality of their labels. However, due to degenerate cases, we must use the refinement of a partition to identify



**Fig. 3.** Figure *a* shows a spatial partition with two regions and its boundary and region labels. Figure *b* shows the result of the refine operation on Figure *a*.

these features. We define the set of edges and vertices imposed on the plane by a spatial partition as follows:

**Definition 3.** *Boundary points of a spatial partition  $\pi$  are classified as being a vertex or as being part of an edge by examining the refinement  $\sigma = \text{refine}(\pi)$  as follows:*

- (i)  $\epsilon(\pi) = \{b \in \omega : |\sigma(b)| = 2\}$
- (ii)  $\nu(\pi) = \{b \in \omega : |\sigma(b)| > 2\}$

## 4 A Discrete Model of Maps

The abstract model of spatial partitions maps each point in the plane to a specific label. However, computers provide only a finite resolution for the representation of data which is not adequate for the explicit representation of abstract spatial partitions. In order to represent maps in computers, a *discrete map model* is required that preserves the properties of spatial partitions while providing a representation that is suitable for storage and manipulation in computers. In this section, we provide a *graph model of spatial partitions*, which is a graph theoretic, discrete model of spatial partitions. Note that there is some ambiguity among graph terms in the literature, especially concerning terms indicating graphs that are allowed to contain loops and multiple edges between pairs of vertices. We begin this section by first providing an overview of graph terms and definitions that we use to develop our model.

### 4.1 Definitions from Graph Theory

In graph theory, a graph is a pair  $G = (V, E)$  of disjoint sets such that  $V$  is a set of vertices and  $E \subseteq V \times V$  is a set of vertex pairs indicating edges between vertices. We denote the sets of vertices and edges for a given graph  $g$  as  $V(g)$  and  $E(g)$ , respectively. A *multigraph* is a pair  $G_M = (V, E)$  of disjoint sets with a mapping  $E \rightarrow V \times V$  allowing a multigraph to have multiple edges between a given pair of vertices. A *loop* in a graph is an edge that has a single vertex as both of its endpoints. A multigraph with loops is a *pseudograph*, and is defined as a pair  $G_P = (V, E)$  with a mapping  $E \rightarrow V \times V$ . Finally, a *nodeless pseudograph* is a pseudograph that (possibly) contains edges that form loops that connect no vertices and that intersect no other edges or vertices. We define a nodeless pseudograph as a triple  $G_N = (V, E, N)$  (where  $N$  is the set of nodeless edges) with mapping  $E \rightarrow V \times V$ .

A *path* is a non-empty graph  $P = (V, E)$  such that  $V = \{v_1, \dots, v_n\}$ ,  $E = \{v_1v_2, \dots, v_{n-1}v_n\}$  and all  $v_i$  are distinct. Given a graph  $g = (V, E)$ , a path of  $g$  is a graph  $p = (V_p, E_p)$  where  $V_p \subseteq V(g) \wedge E_p \subseteq E(g)$  where  $p$  satisfies the definition of a path. A *cycle* is a path whose first and last vertices are identical, defined by a non-empty graph  $C = (V, E)$  such that  $V = \{v_1, \dots, v_n\}$ ,  $E = \{v_1v_2, \dots, v_nv_1\}$  where all  $v_i$  are distinct and  $n \geq 3$ . Given a graph  $g = (V, E)$ , a cycle of  $g$  is a graph  $c = (V_c, E_c)$  where  $V_c \subseteq V(g) \wedge E_c \subseteq E(G)$  where  $c$  satisfies the definition of a cycle. The *length* of a cycle is the number of its edges. A *polygonal arc* is the union of finitely many straight line segments embedded into the plane and is homeomorphic to the closed unit interval  $[0, 1]$ . We use the terms *arc* and *polygonal arc* interchangeably.

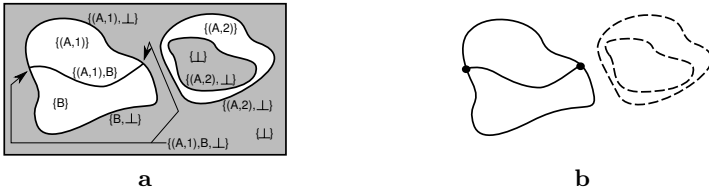
A graph is *planar* if it can be embedded in the plane such that no two edges intersect. A particular drawing of a graph is an *embedding* of the graph. A particular planar graph can have multiple embeddings in the plane such that edges in each embedding are drawn differently. A particular embedding of a planar graph is a *plane graph*. Formally, a plane graph is a pair  $(V, E)$  such that  $V \subseteq \mathbb{R}^2$ , every edge is an arc between two vertices, the interior of an edge contains no vertex, and no two edges intersect except at their vertices. A plane graph  $g$  may contain cycles. We say a cycle  $c$  in plane graph  $g$  is *minimal* if there does not exist a path in  $g$  that splits the polygon induced by  $c$  into two pieces. We use the notation  $C(g)$  to indicate the set of minimal cycles in the graph  $g$ .

## 4.2 Representing Spatial Partitions as Graphs

In this section, we define the type of *spatial partition graphs* that is able to model both the structural and labelling properties of spatial partitions in a graph. We first attempt to define a graph based on the vertex and edge structure of a partition, but show that this is not sufficient because the labels of the partition are not explicitly represented in such a graph. We then define a new type of graph that is capable of representing both the structural and labelling properties of a spatial partition and that is defined based on discrete concepts. We then show how such a graph can be obtained from a given spatial partition.

Recall that in the abstract model of spatial partitions, we can identify a graph structure based on the boundary of a partition  $\pi$ . Specifically, we observe that we can identify  $\nu(\pi)$ , the set of points that are vertices, and  $\epsilon(\pi)$ , the set of point sets belonging to edges of a partition. However, we cannot simply assign the vertices and edges to a pair  $(V, E)$  in order to achieve a graph representation of  $\pi$  since it is possible for nodeless edges to be present in  $\epsilon(\pi)$ . For example, the boundary of the rightmost face of region  $A$  in Figure 1 is composed of two nodeless edges; these edges are nodeless because the label of every point they contain is a set containing two attributes (i.e., no point in the edge satisfies the definition of a vertex in a partition). Therefore, we must use a triple  $(V, E, N)$  consisting of the sets of vertices, edges, and nodeless edges to represent a partition as a graph. Deriving the set  $V$  from a partition  $\pi$  is trivial because we can directly identify the set of vertex points  $\nu(\pi)$ . Definition 3 defines the set of all edges of  $\pi$  as  $\epsilon(\pi)$ ; thus, it does not differentiate between edges and nodeless edges. It follows that





**Fig. 4.** Figure *a* shows the refinement of the partition in Figure 1a. Figure *b* shows the SSPG of *a*. Nodeless edges are dashed.

the set of nodeless edges of a partition,  $N$ , is a subset of  $\epsilon(\pi)$ , but we cannot identify  $N$  by simply examining labels. Intuitively, the label of a nodeless edge should not form a subset of the label of any vertex. However, in Figure 1, the label of the boundary of the upper border of the left face of region  $A$  and the labels of both borders of the right face of region  $A$  are the same. Therefore, we cannot necessarily differentiate nodeless edges from edges by comparing labels. We can circumvent this problem if we can differentiate identically labeled edges from different faces of the same region. This can be achieved through the *refine* operation. Each nodeless edge in  $\pi$  can be identified as an edge in  $\sigma = refine(\pi)$  whose label does not form a subset of any vertex label (a vertex lies on an edge in the refinement of a partition iff the edge label is a subset of the vertex label). Note that the refine operation does not alter the edge structure of a partition, only its labels. Therefore, we can use  $\sigma = refine(\pi)$  to identify the set of nodeless edges  $N$  in  $\pi$  by saying that each edge in  $\sigma$  whose label does not form a subset of any vertex label in  $\sigma$  is in the set  $N$ . The edges of  $\pi$  can then be calculated as  $E = \epsilon(\pi) - N$ . Figure 4a shows the refinement of the partition in Figure 1. Figure 4b shows the graph representation of the partition in Figure 4a obtained by the method described above (nodeless edges are dashed). Because deriving a graph from a partition in this fashion results in a graph that exactly represents the edge structure of the partition from which it is derived, we call this type of graph a *structural spatial partition graph* (SSPG), and define it formally as follows:

**Definition 4.** Given a spatial partition  $\pi$  of type  $A$  and its refinement  $\sigma = refine(\pi)$ , we construct a structural spatial partition graph  $SSPG = (V, E, N)$  with:

$$\begin{aligned}
 V &= \nu(\pi) \\
 E &= \epsilon(\sigma) - N \\
 N &= \{n \in \epsilon(\sigma) \mid \nexists v \in \nu(\sigma) : \sigma(n) \subseteq \sigma(v)\}
 \end{aligned}$$

The SSPG is able to represent the structural aspects of a spatial partition, but it does not maintain the labelling information of the partition. Because a SSPG is defined based on a given partition, the spatial mapping for the partition is known. Therefore, the label for any edge or vertex in a SSPG can be determined through the associated spatial mapping, but this is insufficient for our purposes as we require an explicit representation of labels. However, the SSPG has the property that it is a *plane nodeless pseudograph* (PNP):

**Theorem 1.** *Given a partition  $\pi$  of type A, its corresponding SSPG is a plane nodeless pseudograph.*

*Proof.* The definition of a nodeless graph states that a nodeless graph may contain nodeless edges; therefore, an SSPG is nodeless by definition. Similarly, the definition of a pseudograph states that the graph may contain multiple edges between the same vertices and loops. An SSPG is therefore a pseudograph by definition because it does not exclude such features. Now we must prove that a SSPG is a plane graph. The edges of an SSPG are taken directly from a spatial partition, which is embedded in  $\mathbb{R}^2$ , indicating that the SSPG is an embedded graph. By the definition of spatial partitions, an edge in a partition is a border defined by a one-dimensional point set consisting of points mapped to a single label. Furthermore, this label is derived from the regions which the border separates. Assume that there exists two borders in a spatial partition that cross, which implies that the SSPG for this partition will contain two edges that intersect. In order for this to occur, these edges must separate regions that overlap, which violates the definition of spatial partitions. Therefore, a spatial partition cannot contain two borders that intersect, except at endpoints. Because the edges of an SSPG are taken directly from a spatial partition, then no two edges of an SSPG can intersect. Thus, the SSPG is a plane nodeless pseudograph.  $\square$

Although an SSPG can be easily obtained from a spatial partition, using a SSPG to model spatial partitions is inadequate for two reasons. First, spatial partitions depend on the concept of labels, so the graph representation of a partition must include a label representation. The SSPG does not implicitly model the labels of regions, edges, or vertices; rather, it depends on the existence of a spatial mapping. Second, the edges in the SSPG are taken directly from a spatial partition, which is defined on the concept of infinite point sets that we cannot directly represent discretely. Despite these drawbacks, the SSPG does have the nice property that because a SSPG is defined based on a given spatial partition, we know that any given SSPG is *valid* in the sense that it represents a valid spatial partition. Therefore, we proceed in two phases: we first define a type of graph that is capable of discretely representing the structural properties, and explicitly representing the labeling properties, of spatial partitions. We then show how we can derive graphs of this type from spatial partitions. This allows us to define a valid graph representation of any given spatial partition.

It follows from Theorem [1](#) that an embedded graph that models a spatial partition such that its edges and vertices correspond to the partition's edges and vertices, respectively, must be a PNP. However, a PNP does not model the labeling of spatial partitions. In order to model labels in a graph, we must associate labels with some feature in a graph. In spatial partitions, the labels of boundaries can be derived from the labels of the regions they represent. Thus, it is possible to derive all edge and vertex labels in a partition from the region labels. Therefore, we choose to associate labels in a graph with features that are analogous to regions in spatial partitions. We are tempted to associate labels with minimum cycles in a graph representing a spatial partition; however, this is not able to accurately model situations in which a region in the spatial partition

contains another region such that the boundaries of the regions are disjoint (e.g., the region labeled  $A_2$  and the hole it contains in Figure 4a). Instead, we associate labels with *minimum polycycles* (MPCs) in graphs representing partitions. A minimum polycycle in a PNP  $G$  is a set of minimum cycles consisting of a minimum cycle  $c_o$  (the outer cycle), and all other minimum cycles in  $G$  that lie within  $c_o$ , and not within any other minimum cycle. Note that a minimum cycle of a plane graph induces a region in the plane defined by a Jordan curve. Therefore, we can differentiate between the interior, boundary, and exterior of such a region. We denote the region induced in the plane by the minimum cycle  $C$  of plane graph  $G$  as  $R(C)$ . We now formally define minimum polycycles:

**Definition 5.** *Given a plane nodeless pseudograph  $G$ , a minimum polycycle of  $G$  is a graph  $MPC = (V, E, N)$  where  $V \subseteq V(G)$ ,  $E \subseteq E(G)$ ,  $N \subseteq N(G)$ , and  $C(MPC) \subseteq C(G)$ , containing an outer cycle  $c_o \in C(MPC)$  and zero or more inner cycles  $c_1, \dots, c_n \in C(MPC)$  such that:*

- (i)  $\forall v \in V : (\exists d \in C(MPC) | v \in V(d))$
- (ii)  $\forall e \in E : (\exists d \in C(MPC) | e \in E(d))$
- (iii)  $\forall n \in N : (\exists d \in C(MPC) | n \in N(d))$
- (iv)  $\forall c_i \neq c_o \in C(MPC) : \partial R(c_i) \subseteq (\partial R(c_o) \cup R(c_o)^\circ)$
- (v)  $\nexists c_j, c_k \in C(MPC) | c_j \neq c_o \wedge c_k \neq c_o \wedge c_j \neq c_k$   
 $\wedge \partial R(c_j) \subseteq (\partial R(c_k) \cup R(c_k)^\circ)$
- (vi)  $\nexists d \in C(G) | (\partial R(d) \subseteq (\partial R(c_o) \cup R(c_o)^\circ)) \wedge \neg(d \in C(MPC))$   
 $\wedge (\forall c_i \neq c_o \in C(MPC) : \neg(\partial R(d) \subseteq (\partial R(c_i) \cup R(c_i)^\circ))$

Thus, a MPC induces a region in the plane that may contain holes defined by the minimum cycles that lie in the interior of the outer cycle. Furthermore, by associating labels with MPCs in a graph representing a spatial partition, we do not need to explicitly represent the labels of edges and vertices in the graph since they can be derived by simply finding all MPCs that an edge or vertex participates in. We denote the set of all MPCs for a PNP  $G$  as  $MC(G)$ .

The second problem with SSPGs is that the edges are defined as infinite point sets. We require a discrete representation of edges. Therefore, in addition to assigning labels to MPCs, we define our new type of graph such that its edges are arcs consisting of a finite number of straight line segments. We define the *labeled plane nodeless pseudograph* (LPNP) as a PNP with labeled MPCs, denoted *faces*, and edges modeled as arcs as follows:

**Definition 6.** *Given an alphabet of labels  $\Sigma_L$ , a labeled plane nodeless pseudograph is defined by the four-tuple  $LPNP = (V, E, N, F)$  consisting of a set of vertices, a set of arcs forming edges between vertices, a set of arc loops forming nodeless edges, and a set of faces, with:*

- $V \subseteq \mathbb{R}^2$
- $E \subseteq V \times V \times (\mathbb{R}^2)^n$  where  $n$  is finite and each edge is an arc (arcs with endpoints in  $V$  and segment endpoints in  $\mathbb{R}^2$ )
- $N \subseteq (\mathbb{R}^2)^n$  where  $n$  is finite (nodeless arc loops)
- $F \subseteq \{(l \in \Sigma_L, m \in MC((V, E, N)))\}$

Recall that in the definition of spatial partitions, an unbounded face is explicitly represented with an empty label corresponding to the exterior of the partition. We do not explicitly model this unbounded face in the LPNP for two reasons: (i) we cannot guarantee that the edges incident to the unbounded face of a LPNP will form a connected graph, and (ii) if the edges incident to the unbounded face do form a connected graph, we cannot guarantee that it will be a cycle. However, we can determine if an edge in a LPNP is incident to the unbounded face if it participates in only a single MPC. This follows from the fact that each edge separates two regions in a partition. Because all bounded faces are modeled as MPCs in a LPNP, an edge that participates in only a single face must separate a MPC and the unbounded face.

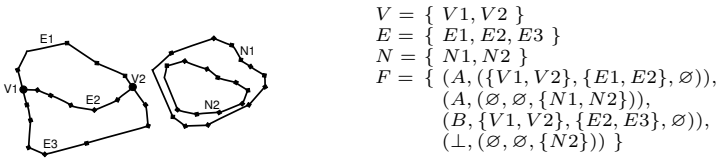
The LPNP allows us to discretely model a labeled graph structure, but we have not yet discussed how we can obtain a LPNP for a given spatial partition. Note that because the edges of a LPNP are defined as arcs, they cannot directly represent edges from a partition. Instead, each edge in a LPNP is an approximation of an edge in its corresponding spatial partition. Therefore, we define the approximation function  $\alpha$  that takes an edge from a partition and returns an arc which approximates that edge. Given a spatial partition  $\pi$  of type  $A$  and an edge approximation function  $\alpha$ , we derive the corresponding LPNP in a similar manner as we derived the SSPG from a partition. The set of vertices in the LPNP is equivalent to  $\nu(\pi)$ . In order to calculate the nodeless edges, we consider the refinement  $\sigma = refine(\pi)$ . Nodeless edges are then identified as all edges in  $\sigma$  whose label is not a subset of any vertex label. The set of edges is then difference of  $\epsilon(\pi)$  and the set of nodeless edges. Finally, each labeled MPC consists of the set of approximations of the edges surrounding each region in  $\sigma$  along with the label of the corresponding face in  $\pi$ . Given an edge  $e$ , we use the notation  $V_e(e)$  to indicate the set of vertices that  $e$  connects. Figure 5 depicts the LPNP for the partition shown in Figure 1.

**Definition 7.** *Given a spatial partition  $\pi$  of type  $A$ , edge approximation function  $\alpha$ , and  $\sigma = refine(\pi)$ , we derive a LPNP =  $(V, E, N, F)$  from  $\pi$  as follows:*

$$\begin{aligned}
 V &= \nu(\pi) \\
 E &= \{\alpha(e \in \epsilon(\sigma) | \neg(\alpha(e) \in N))\} \\
 N &= \{\alpha(n \in \epsilon(\sigma) | \nexists v \in \nu(\sigma) : \sigma(n) \subseteq \sigma(v)\} \\
 F &:= \forall r \in \rho(\sigma) | r \subseteq s \in \rho(\pi) : (l, (V_m, E_m, N_m)) \in F \text{ where :} \\
 &\quad l = \pi^{-1}(r) \\
 &\quad E_m = \{\alpha(e) | e \in \omega(\sigma) \wedge e \subseteq \partial r \wedge \alpha(e) \in E\} \\
 &\quad N_m = \{\alpha(n) | n \in \omega(\sigma) \wedge n \subseteq \partial r \wedge \alpha(n) \in N\} \\
 &\quad V_m = \bigcup_{e \in E_m} V_e(e)
 \end{aligned}$$

Note that it is possible to approximate edges in a partition in multiple ways. Thus, a single spatial partition may have multiple LPNPs that represent it.

In the definition of LPNPs, no restrictions are placed on the labels of MPCs. Therefore, it is possible for an labeled graph to fit the definition of an LPNP, but be labeled in such a way that violates the definition of spatial partitions. In other words, a LPNP may be labeled such that no spatial partition exists from which



**Fig. 5.** A labeled plane nodeless pseudograph for the partition in Figure 1. The edges and vertices are marked so that the sets of vertices, edges, nodeless edges, and faces can be expressed more easily.

the LPNP can be derived. A simple example of this is a LPNP containing two MPCs that have the same label and share an edge. Because edges only separate regions with different labels in a partition, this LPNP can not be derived from any valid spatial partition. Thus, the set of all valid LPNPs is larger than the set of LPNPs that can be derived from some spatial partition. We define a LPNP that can be derived some spatial partition as a *spatial partition graph* (SPG).

**Definition 8.** A spatial partition graph  $G$  is a labeled plane nodeless pseudograph such that there exists some valid partition from which  $G$  can be derived.

### 4.3 Properties of Spatial Partition Graphs

In the previous section, we defined the type of spatial partition graphs and showed how a SPG can be derived from a given spatial partition. However, we currently define a SPG as being valid only if it can be derived from a valid spatial partition. Given a labeled graph in the absence of a spatial partition, we currently cannot determine if the graph is a SPG. In this section, we discuss the properties of SPGs such that we can determine if a SPG is valid by examining its structure and labels.

Given a LPNP in the absence of a source partition from which it can be derived, we must ensure that structural and labelling properties of the LPNP are consistent with the properties of spatial partitions. To achieve this, we examine the properties of spatial partitions, defined by Definition 2 and Definition 3, and show how these properties are expressed in SPGs derived from spatial partitions. We then define the properties of SPGs and show that any LPNP that satisfies these properties is a SPG.

Recall that a SSPG is a PNP. It follows from Theorem 1 that any graph that models the edges of a partition as graph edges and vertices of a partition as graph vertices must be a PNP. Therefore, a graph cannot be a SPG if it is not a PNP. This is already expressed indirectly by the definition of SPGs as LPNPs.

Definition 2 formally defines constraints on spatial mappings that specify the type of partitions. These constraints indicate that (i) the regions in a partition are regular open point sets, and (ii) the borders separate uniquely labeled regions and carry the labels of all adjacent regions. From these properties of partitions, we can derive properties of SPGs. By (i), we infer that all edges and vertices in a SPG must be part of a MPC. If an edge is not part of a MPC, then the edge does not separate two regions. Instead, it extends either into the interior of a MPC or

into the unbounded face of the SPG, forming a cut in the polygon induced by the MPC or the unbounded face. If a vertex exists that is not part of a MPC, then it is either a lone vertex with no edges emanating from it, or it is part of a sequence of edges that are not part of a MPC. In the first case, the vertex either exists within the region induced by a MPC or the unbounded face of the LPNP, forming a puncture. In the second case, the vertex exists in a sequence of edges that is not part of a MPC, which we have already determined to be invalid.

By the second property of spatial partitions (ii), we infer that edges must separate uniquely labeled MPCs. Therefore, there cannot be an edge in an LPNP that participates in two MPCs with the same label. Furthermore, every region in a partition must be labeled. It follows that every MPC in a SPG must be labeled. Because the unbounded face is not explicitly labeled in a SPG, one special case exists: a MPC forming a hole in a SPG (i.e., labeled with  $\perp$ ) cannot share an edge with the unbounded face, as this would result in an edge separating two regions with the same label.

Definition 3 further identifies properties of spatial partitions. According to this definition, edges in spatial partitions always have two labels, and vertices always have three or more labels. Recall that in LPNPs, the unbounded face of the graph is not explicitly labeled. Therefore, in a SPG, all edges must participate in either one or two MPCs. Edges incident to the unbounded face of the graph will participate in only one MPC. The requirement that vertices have three or more labels in a spatial partition indicates that at a vertex, at least three regions meet. It follows that in a SPG, each vertex has at least a degree of three. Furthermore, because the unbounded face is not explicitly labeled, vertices must have at least two labels in a SPG (i.e., a vertex must participate in at least two MPCs). We summarize the properties of SPGs and show that any LPNP that satisfies these properties is a SPG:

**Definition 9.** *An SPG  $G$  has the following properties:*

- (i)  $G$  is a plane nodeless pseudograph (Theorem 1)
- (ii)  $\forall e \in E(G) \cup N(G), \exists (l, X) \in F(G) | e \in E(X) \cup N(X)$  (Definition 2(i))
- (iii)  $\forall v \in V(G), \exists (l, X) \in F(G) | v \in V(X)$  (Definition 2(i))
- (iv)  $\forall v \in V(G) : \text{degree}(v) \geq 3$  (Definition 3)
- (v)  $\forall e \in E(G) \cup N(G) :$   
 $1 \leq |\{(l, X) \in F(G) | e \in E(X) \cup N(X)\}| \leq 2$  (Definition 3)
- (vi)  $\forall (l_1, X_1), (l_2, X_2) \in F(G) | l_1 = l_2 :$   
 $(\nexists e_1 \in E(X_1) \cup N(X_1), e_2 \in E(X_2) \cup N(X_2) | e_1 = e_2)$  (Definition 2(ii))
- (vii)  $\forall m \in MC(G), \exists f = (l, X) \in F(G) | m = X$  (Definition 2(ii))
- (viii)  $\forall e \in E(G) \cup N(G)$   
 $|\{(l, X) \in F(G) | e \in E(X) \cup N(X)\}| = 1 : l \neq \{\perp\}$  (Definition 2(ii))

**Theorem 2.** *Any LPNP that satisfies the properties in Definition 9 is a SPG.*

*Proof.* The properties listed in Definition 9 indicate how the properties of spatial partitions are expressed in SPGs. From Theorem 1, we know that a valid SPG must be a PNP. Definition 2(i) states that all regions in a partition must be

regular open sets. Because the faces in a SPG are analogous to regions in a spatial partition, this means that all edges and vertices must belong to some face; otherwise, they form a puncture or cut in some face of the SPG. Definition 9(ii) and Definition 9(iii) express this requirement. Definition 2(ii) states that borders in a partition between regions carry the labels of both regions. This implies that an edge in a spatial partition separates regions with different labels, and that every region in a spatial partition has a label. Definition 9(vi) and Definition 9(vii) express this by stating that if an edge participates in two faces of a SPG, those faces have different labels, and that every MPC in a SPG is a labeled face of the SPG. Because the unbounded face is not labeled, we must explicitly state that no edge that participates in a cycle forming a hole can have only a single label, as this implies that an edge is separating two regions with the  $\perp$  label (Definition 9(viii)). Definition 3 states that edges in a partition carry two region labels, and that vertices carry three or more region labels. Because the unbounded face is not labeled in a SPG, we cannot directly impose these properties on a SPG. Instead, we observe that the number of region labels on a vertex in a partition indicates a minimum number of regions that meet at that vertex. Therefore, we can express this property in SPG terms by stating that vertices in a SPG must have degree of at least three (Definition 9(iv)). The edge constraint from Definition 3 can be specified in terms of a SPG by the property that an edge must participate in exactly one or two faces, indicating that the edge will have exactly one or two labels (Definition 9(v)). Therefore, all properties of partitions are expressed in terms of SPGs in Definition 9, and any LPNP that satisfies these properties is a SPG.  $\square$

We now have the ability to either derive a valid SPG from a spatial partition, or verify that a SPG is valid in the absence of a spatial partition from which it can be derived. Finally, we show that given a valid SPG, we can directly construct a valid spatial partition that exactly models the SPG's spatial structure and labels. Recall that a spatial partition is defined by a spatial mapping that maps points to labels and satisfies certain properties. Therefore, to construct a partition from a SPG, we must be able to derive a spatial mapping from a SPG. We can construct such a mapping based on the labeled MCPs of a SPG. Each MCP of a SPG induces a polygon in the plane that is associated with a label. Each of these polygons is a spatial region, and is defined by its boundary, which separates the interior of the polygon from its exterior. Therefore, we can identify the interior, boundary, and exterior of such a polygon. We use the notation  $R(X)$  to denote the polygon induced in the plane by MCP  $X$ . A point that falls into the interior a polygon can therefore be mapped directly to that polygon's label. The labels of points belonging to edges in the SPG are slightly more difficult to handle. Each point belonging to an edge is mapped to the labels of each face in which the edge participates. If an edge happens to be incident to the unbounded face (it is an edge participating in a single minimum cycle), it also is mapped to the  $\perp$  label. Similarly, each vertex is mapped to the labels of each cycle in which it participates. If a vertex is incident to the unbounded face, it is also mapped to the  $\perp$  label. A vertex is incident to the unbounded face if and only if it is the endpoint of an edge that is incident to the unbounded face.

In order to define this mapping, we first provide a notation to distinguish between edges and vertices that are incident to the unbounded face, and those that are not. We then show how to derive a spatial partition from a SPG:

**Definition 10.** *Given a SPG  $G$ , we distinguish two sets of edges: the set containing edges incident to the unbounded face, denoted  $E_{\perp}$ , and the set containing edges not incident to the unbounded face, denoted  $E_b$ . Likewise, we distinguish two sets of vertices: the set containing vertices incident to the unbounded face, denoted  $V_{\perp}$ , and the set containing vertices not incident to the unbounded face, denoted  $V_b$ . These sets are defined as follows:*

$$\begin{aligned} E_{\perp}(G) &= \{e \in E(G) \cup N(G) \mid |\{(l, X) \in F(G) \mid e \in E(X) \cup N(X)\}| = 1\} \\ E_b(G) &= (E(G) \cup N(G)) - E_{\perp}(G) \\ V_{\perp}(G) &= \{v \in V(G) \mid (\exists e \in E_{\perp} \mid v \in V_e(e))\} \\ V_b(G) &= V(G) - V_{\perp}(G) \end{aligned}$$

**Definition 11.** *Given a SPG  $G$ , we can directly construct a spatial partition  $\pi$  of type  $A$  as follows:*

$$\begin{aligned} A &= \{l \mid (l, X) \in F(G)\} \cup \{\perp\} \\ \pi(p) &= \begin{cases} \left\{ \begin{array}{ll} \{l \mid (l, X) \in F(G) \wedge p \in R(X)^{\circ}\} & \text{if } \exists (l, X) \in F(G) \mid p \in R(X)^{\circ} & (1) \\ \{\perp\} & \text{if } \nexists (l, X) \in F(G) \mid \\ & p \in R(X)^{\circ} \cup \partial R(X) & (2) \end{array} \right. \\ \left\{ \begin{array}{ll} \{l \mid (l, X) \in F(G) \wedge p \in \partial R(X)\} & \text{if } (\exists (l, X) \in F(G) \mid p \in \partial R(X)) \\ & \wedge (\nexists e \in E_{\perp}(G) \mid p \in e) & (3) \\ \{l \mid (l, X) \in F(G) \wedge p \in \partial R(X)\} \cup \{\perp\} & \text{if } (\exists (l, X) \in F(G) \mid p \in \partial R(X)) \\ & \wedge (\exists e \in E_{\perp}(G) \mid p \in e) & (4) \end{array} \right. \end{cases} \end{aligned}$$

In Definition 11, the type of a spatial partition derived from a SPG consists of the set of labels of faces in the SPG. The mapping is then defined by finding the labels of all faces a point participates in. If a point  $p$  lies in the interior of a face (1), then the label of that point will be the label of the face. If  $p$  does not lie in the interior or boundary of any face (2), it maps to the label  $\perp$ . If  $p$  lies on a boundary that is not incident to the unbounded face (3), it is mapped to the set of labels of all faces that include  $p$  in its boundary. Note that because face boundaries include vertex points, this case handles the mapping of both edge and vertex points. Similarly, if  $p$  lies on a face boundary that is incident to the unbounded face (4), then  $p$  is mapped to the set of labels from all faces which include  $p$ , as well as the label  $\perp$ .

## 5 Conclusions and Future Work

The notion of a map geometry as a data type in spatial systems has received much attention in the literature. The type of spatial partitions is, so far, the only representation of map geometries that is able to implicitly model both the spatial and thematic properties of maps and guarantee closure of the spatial partition type under all known operations over spatial partitions. In this paper, we have provided a discrete, graph theoretic model of spatial partitions suitable



for implementation in spatial systems. This contribution overcomes the main drawback of the type of spatial partitions, i.e., that they are defined on abstract concepts such as infinite point sets that are not implementable in computers. We have defined the type of spatial partition graphs, and identified their properties. Furthermore, we have shown how spatial partition graphs can be derived from spatial partitions, and vice versa. By defining a precise, discrete, mathematical model of spatial partition graphs, we have provided a basis which can be used to implement map geometries in spatial systems, and to carry out research into algorithms for map geometries.

Future work includes extending the spatial partition model, and the spatial partition graph model, to allow the inclusion of point and line features in map geometries. Additionally, map geometries have shown promise in being used for spatial query processing. We plan to implement the spatial partition graph model at various levels of a spatial database system to test its functionality in different roles. Finally, we plan to investigate new operations over map geometries.

## References

1. Schneider, M., Behr, T.: Topological Relationships between Complex Spatial Objects. *ACM Trans. on Database Systems (TODS)* 31(1), 39–81 (2006)
2. Güting, R.H.: Geo-relational algebra: A model and query language for geometric database systems. In: Schmidt, J.W., Missikoff, M., Ceri, S. (eds.) *EDBT 1988*. LNCS, vol. 303, pp. 506–527. Springer, Heidelberg (1988)
3. Güting, R.H., Schneider, M.: Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal* 4, 100–143 (1995)
4. Huang, Z., Svensson, P., Hauska, H.: Solving spatial analysis problems with geosal, a spatial query language. In: *Proceedings of the 6th Int. Working Conf. on Scientific and Statistical Database Management*, Institut f. Wissenschaftliches Rechnen Eidgenössische Technische Hochschule Zürich, pp. 1–17 (1992)
5. Voisard, A., David, B.: Mapping conceptual geographic models onto DBMS data models. Technical Report TR-97-005, Berkeley, CA (1997)
6. Ledoux, H., Gold, C.: A Voronoi-Based Map Algebra. In: *Int. Symp. on Spatial Data Handling (July 2006)*
7. Tomlin, C.D.: *Geographic Information Systems and Cartographic Modelling*. Prentice-Hall, Englewood Cliffs (1990)
8. Filho, W.C., de Figueiredo, L.H., Gattass, M., Carvalho, P.C.: A topological data structure for hierarchical planar subdivisions. In: *4th SIAM Conference on Geometric Design (1995)*
9. De Floriani, L., Marzano, P., Puppo, E.: Spatial queries and data models. In: Frank, I.C.A.U., Formentini, U. (eds.) *Information Theory: a Theoretical Basis for GIS*. LNCS, vol. 716, pp. 113–138. Springer, Heidelberg (1992)
10. Viana, R., Magillo, P., Puppo, E., Ramos, P.A.: Multi-vmap: A multi-scale model for vector maps. *Geoinformatica* 10(3), 359–394 (2006)
11. Erwig, M., Schneider, M.: Partition and Conquer. In: Frank, A.U. (ed.) *COSIT 1997*. LNCS, vol. 1329, pp. 389–408. Springer, Heidelberg (1997)
12. Dugundi, J.: *Topology*. Allyn and Bacon (1966)
13. Tilove, R.B.: Set Membership Classification: A Unified Approach to Geometric Intersection Problems. *IEEE Trans. on Computers* C-29, 874–883 (1980)

# Geographic Ontology Matching with IG-MATCH

Guillermo Nudelman Hess<sup>1,2</sup>, Cirano Iochpe<sup>1,3</sup>, and Silvana Castano<sup>2</sup>

<sup>1</sup> Universidade Federal do Rio Grande do Sul

Instituto de Informática - Av. Bento Gonçalves, 9500, 15064 Porto Alegre - Brazil

<sup>2</sup> Università degli Studi di Milano

DICO - Via Comelico, 39, 20135 Milano - Italy

<sup>3</sup> Procempa - Empresa da Tecnologia da Informação e Comunicação de Porto Alegre

Av. Ipiranga, 1200, Porto Alegre - Brazil

{hess,ciochpe}@inf.ufrgs.br, castano@dico.unimi.it

**Abstract.** To achieve accurate results when matching geographic ontologies, it is important to have clear what has to be compared, and just then start comparing them. In this paper we define a geographic ontology reference model and, from it, the set of heterogeneities that may occur when comparing two geographic ontologies is elaborated, at both the concept and instance-level. Based on the heterogeneities set we then present the IG-MATCH, which consists in a software architecture that implements a methodology to perform geographic ontology matching, using some metrics specially developed for the geographic domain.

## 1 Introduction

Since the creation of Geographic Information Systems (GIS), new fields of research are emerging due to the peculiarities of the geographic data, which is much more specific than conventional (alphanumeric) data. In fact, besides the descriptive components, geographic data is featured by at least other two characteristics, namely geometry and location [1,2]. Geographic data may also have the temporal component [3], even if this cannot be pointed as a specific feature for geographic data. In general, geographic data is stored in a Geographic Database (GDB). A GDB has all the functionalities and capabilities of a conventional database; in addition it can handle spatial relationships among two or more geographic data as well as their spatial component (geometry and location).

Actually GIS are used every day. Some examples are the Global Positioning Systems (GPS) used in cars, the Google Earth tool, maps generators on the web, and so on. Producing geographic data is time consuming and expensive. Furthermore, in many cases the data needed is already available in some other systems or organizations.

At the same time, the diffusion of the Internet allowed the interchange of information all around the world. If, on one hand, this interchange offers a lot of benefits, such as the reuse of information and knowledge sharing, on the other hand it generates the need to deal with the heterogeneities among the

information obtained from distinct geographic sources. This problem is difficult to solve due to poor documentation as well as implicit semantics of the data and diversity of data sets.

The scenario above encouraged the present research, in which propose one solution for the problem of geographic information integration. Our solution, called iG-MATCH, is conceived for both concept and instance-level matching, and consists in a methodology with similarity metrics tailored specially for the particularities of the geographic information.

### 1.1 Motivating Example

Figures 1 and 2 presents two geographic ontologies to be compared. The rectangles with continuous lines represent concepts, the ellipses the properties representing attributes associated with a concept and the dashed rectangles the instances belonging to a concept. The arcs linking two concepts correspond to the properties which represent relationships holding between them, while the *isa* labeled arrows are the taxonomic relationships (axioms) between two concepts, in which one is the specialization of the other.

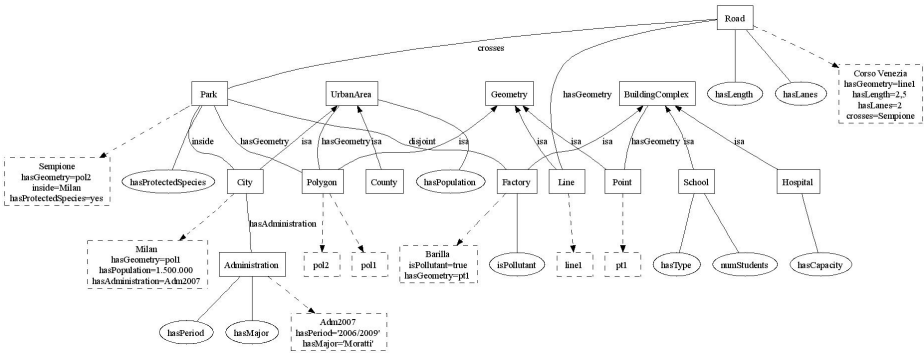


Fig. 1. Geographic ontology *O*

The examples are rather simple, but complete for our purposes. They have both spatial concepts, such as *Park*, *City*, *Factory* and non-geographic concepts (*Administration*). There are spatial relationships in both ontologies (*crosses*, *inside*, *overlaps*) as well as conventional relationships (*hasAdministration* held between *City* and *Administration*). The geometries of the concepts are of various types. We use these two ontologies throughout this paper and in the following sections we analyze how these elements may affect the matching of geographic ontologies.

The rest of the paper is organized as follows. In Section 2 we define a geographic ontology model, which guide the heterogeneities identification and the matching process as well. Section 3 formalizes the types of heterogeneities that

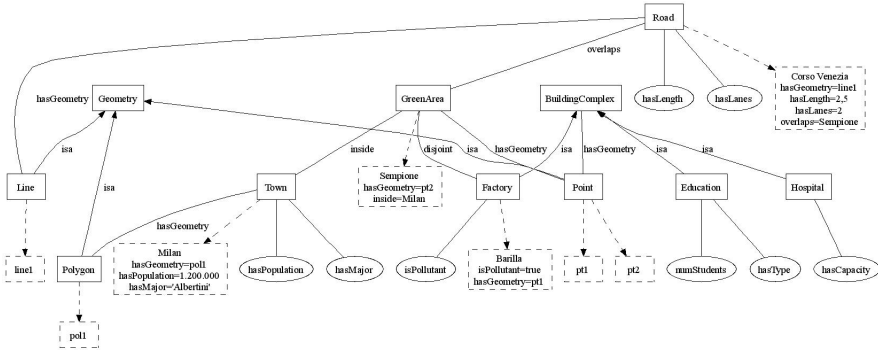


Fig. 2. Geographic ontology  $O'$

may occur when comparing two geographic ontologies. In Section 4 we present the 1G-MATCH, which consists in a software architecture and similarity metrics to assess the degree of similarity between two geographic ontologies. Some related works are presented in Section 5. Finally, the conclusions and future directions are discussed in section 6.

## 2 Definition of a Reference Model

A geographic ontology (or geo ontology) can be defined as a 4-tuple  $O = (C, P, I, A)$ , where  $C$  is the set of concepts,  $P$  is the set of properties,  $I$  is the set of instances, and  $A$  is the set of axioms. A concept  $c \in C$  is classified into domain concept, such as a *River*, a *Park* or a *Building*, or geometry concept, such as *Point*, *Line* or *Polygon*. Furthermore, a geographic domain concept  $gc$  is a specialization of a domain concept which represents a geographic phenomena. By definition, a geographic domain concept  $gc$  must have at least one associated geometric property, which is explained in the following. In this paper the terms geographic class, geographic concept and geographic domain concept are synonyms and may be interchanged.

In an ontology a property can be defined by itself, i.e., outside the context of a concept. However, for matching purposes a property is relevant when associated to a concept, directly in its domain or through a restriction. Thus, we opted to associate the concept to a property in the definition of the latter.

In a geographic ontology, each property  $p \in P$  can be of one of four possible types: conventional, spatial, geometric or positional. A conventional property may be even a data type property or an object type property. In the first case it represents an attribute of a domain concept. In the second case it represents an association between a domain concept (geographic or not) with a non-geographic domain concept.

An attribute  $a \in P$  is a ternary relation of type  $a(c, an, dtp)$ , where  $an$  is the name of the attribute and  $dtp$  is a data type (such as string, integer, etc.).

A conventional property  $cr$  is defined as:

$$cr = \{r(c, rn, c_x, minCard, maxCard) \in P | ((c : \neg gc) \vee (c_x : \neg gc)) \vee ((r : \neg ge) \wedge (r : \neg sr))\}$$

where  $c$  is the domain concept,  $rn$  is the property name,  $c_x$  is the related concept and  $minCard$  and  $maxCard$  are, respectively, the property minimum and maximum cardinality.  $ge$  is a geometric property and  $sr$  represents a spatial relation. Both will be detailed in the following.

A spatial property  $sr$  can be of three types: topological, directional or metric. As the metric property is, in general, processed by a GIS system, we focus on the topological and directional ones. Thus,  $sr$  is always an object type property, and represents an association between two geographic domain concepts.

$$sr = \{r(c, rn, c_x, minCard, maxCard) \in P | (c : gc) \wedge (c_x : gc)\}$$

A geometric property (always an object type property)  $ge$  is an association between a geographic domain concept with a geometry concept.

$$ge = \{r(c, rn, c_x, minCard, maxCard) \in P | (c : gc) \wedge (c_x : geo)\}$$

Finally, a positional property is a data type property that must be associated to a geometry concept, to give its location (set of coordinates).

The set of axioms  $A$  describes the hierarchical (IS-A) relationships between concepts and provides associations among properties and concepts. A hierarchy  $h \in A$  is a binary relation of type  $h(c, c_x)$ , where  $c_x$  is superclass of the concept  $c$ .

A geographic instance  $i \in I$  is defined as  $i = (type, id, vP, coord)$ , where  $type$  is the concept  $c$  it instantiates,  $id$  is the unique identifier of the instance,  $vP$  is the set of values defined for the properties associated to the concept  $c$  that  $i$  instantiates, and  $coord$  is the set of geographic coordinates of the instance.

On the basis of this reference model, it is possible to point out at least three differences between geographic information and conventional (ontology or schema) matching:

- The spatial relationships have a pre-defined semantics and are standardized, while conventional relationships may assume different semantics depending on the associated concepts.
- Every geographic concept has, at least, one associated geometry representing it. The geometry plays a fundamental role on defining the possible spatial relationships the concept may have.
- A geographic instance has a number of pair of coordinates (x,y) representing its spatial position over the earth surface. These coordinates may be expressed in a given coordinate system.

### 3 A Clarification of Geographic Information Heterogeneities

In order to know what to consider when matching two geographic ontologies, in this section we define the kinds of heterogeneities that should be taken into account at both the concept-level and the instance-level.

### 3.1 Concept-Level Heterogeneities

In this section the possible heterogeneities are classified regarding the comparison of a concept  $c \in$  ontology  $O$  against a concept  $c' \in$  ontology  $O'$ . Considering the definition of ontology, concepts, instances, properties and axioms presented in the preview section, the possible heterogeneities are defined as follows.

**Name heterogeneity:** The concept name heterogeneity NH occurs when, given two concepts names  $t(c)$  and  $t(c')$  they are neither equal nor synonyms. The synonym relation  $SYN(t(c), t(c'))$  is obtained by searching an external thesaurus.

$$NH(c, c') = \{(t(c) \neq t(c')) \wedge (SYN(t(c), t(c')) = false)\}$$

Considering the ontologies  $O$  and  $O'$ , the concepts *Park* from  $O$  and *GreenArea* from  $O'$  are examples of name heterogeneity. On the other hand, *City* and *Town* do not have name heterogeneity, because even if the terms are not the same, the function  $SYN(t(c), t(c'))$  returns true when searching a external dictionary.

**Property heterogeneity:** The concept property heterogeneity PH occurs when there is an attribute heterogeneity AH or a relationship heterogeneity RH.

The AH heterogeneity between  $c \in O$  and  $c' \in O'$  occurs when at least one of the attributes  $a(c, an, dtp) \in P$  in ontology  $O$  does not match with any of the attributes  $a(c', an', dtp') \in P'$  in ontology  $O'$ . The heterogeneity can be generated due to different attribute names or different attribute datatypes.

$$AH(c, c') = \{\exists a(c, an, dtp) \in P | \forall a(c', an', dtp') \in P', (an \neq an') \vee (dtp \neq dtp')\}$$

As an example of attribute heterogeneity, lets consider the concepts *City* from  $O$  and *Town* from  $O'$ . The attribute *hasMajor* is a property of *Town*, but is not associated to *City*.

The RH heterogeneity between  $c \in O$  and  $c' \in O'$  is defined over the conventional relationships (not geometric nor spatial). It applies to both geographic as to non-geographic concepts. It occurs when at least one of the relationships  $cr(c, rn, c_x, minCard, maxCard) \in P$  in ontology  $O$  does not have a correspondent  $cr(c', rn', c'_x, minCard', maxCard') \in P'$  in ontology  $O'$ . The heterogeneity may occur due to a different associated concept  $c_x$  as well as due to the relationship cardinalities *minCard* and *maxCard*. As in many times the conventional relationships names are not significant as to identify the relationship, the component *rn* can be ignored.

$$RH(c, c') = \{\exists cr(c, rn, c_x, minCard, maxCard) \in P | \forall r(c', rn', c'_x, minCard', maxCard') \in P', (c_x \neq c'_x) \vee (minCard \neq minCard') \vee (maxCard \neq maxCard')\}$$

The concepts *City* from  $O$  and *Town* from  $O'$  present an example of relationship heterogeneity. The property *hasAdministration*, which relates *City* with the concept *Administration* is not in the context of the concept *Town*.

Regarding the geographic domain concepts, two additional types of heterogeneity exist, one for each type of relationship (geometry and spatial relation).

The **geometric concept heterogeneity** GH between  $gc \in O$  and  $gc' \in O'$  happens when the two geographic concepts  $gc$  and  $gc'$  have different geometries, i.e., the *hasGeometry* property relates the geographic domain concept to concepts representing different geometries.

$$GH(gc, gc') = \{\exists ge(gc, hasGeometry, geo, minCard, maxCard) \in P \mid \forall ge(gc', hasGeometry', geo', minCard', maxCard') \in P', geo \neq geo'\}$$

In this case only the associated geometry concept  $geo$  counts, because it is the one which defines the geometry (point, line, polygon) of the geographic concept. Due to the possibility of the multi-representation of a geographic concept, i.e., multiple geometries, if at least one of the geometries of  $gc$  matches with a geometry of  $gc'$ , there is no heterogeneity.

As an example of geometric heterogeneity, let's consider the concepts *Park* from  $O$  and *GreenArea* from  $O'$ . While the former is associated to a geometry concept *Polygon*, the latter is associated to a geometry concept *Point*, for the same *hasGeometry* property.

The **spatial relationship heterogeneity** SH between two geographic concepts  $gc \in O$  and  $gc' \in O'$  happens when there is at least one spatial relationship  $sr(gc, rn, gc_x, minCard, maxCard) \in P$  in ontology  $O$  without a matching  $sr(gc', rn', gc'_x, minCard', maxCard') \in P'$  in ontology  $O'$ .

$$SH(gc, gc') = \{\exists sr(gc, rn, gc_x, minCard, maxCard) \in P \mid \forall sr(gc', rn', gc'_x, minCard', maxCard') \in P', (gc_x \neq gc'_x) \vee (minCard \neq minCard') \vee (maxCard \neq maxCard') \vee (rn \neq rn')\}$$

The names of the spatial relationships are, in general, standardized in the literature. Hence, the component  $rn$ , which holds the relationship name, has to be considered. The topological and directional relationships are considered. The metrics one do not count because in general they are calculated by a GIS and not defined as properties or restrictions of a concept.

Example of a spatial relationship heterogeneity is the association *Road crosses Park* in ontology  $O$  and *Road overlaps GreenArea* in ontology  $O'$ . Even if we consider that *GreenArea* and *Park* could be synonyms, in ontology  $O$  the relationship name is *crosses*, while in  $O'$   $rn' = overlaps$ . As will be discussed in the paper, these relationships can be equivalent, but in a first analysis it seems that there is a spatial relationship heterogeneity.

**Hierarchy heterogeneity:** The hierarchy heterogeneity HH between two concepts  $c \in O$  and  $c' \in O'$  occurs when the set of superclasses  $SUP(c)$  of the concept  $c \in O$  is different from the set of superclasses  $SUP(c')$  of the compared concept  $c' \in O'$ . This means that at least one of the superclasses present in  $SUP(c)$  is not found in  $SUP(c')$ .

$$HH(c, c') = \{\exists c_x \in h(c, c_x) \mid \forall c'_x \in h(c', c'_x), c_x \neq c'_x\}$$

The concepts *City* and *Town* from  $O$  and  $O'$ , respectively, are examples of hierarchy heterogeneity. The former has as superclass the concept *UrbanArea*, while the latter do not have a superclass (actually, in an ontology, all concepts are subclasses of *thing*, but for easiness of comprehension we omitted it from the ontology).

### 3.2 Instance-Level Heterogeneities

As important as the matching of the concepts is the matching of their instances. Especially in the geographic field there are many features that can influence the similarity measurement process which are not present when dealing with non-geographic ontologies. These features are, for example, the scale, spatial position, time when the instances were obtained, and so on. However, the non-spatial properties, such as the attributes (property) values, cannot be neglected. In this section we define the heterogeneities that may occur at the instance-level when comparing two geographic ontologies.

**Identifier heterogeneity:** When a concept in an ontology is instantiated, in general the unique identifier has a really significant value. It is not like the `objectId` of an instance of a class which is automatically generated. In the case of an ontology is the main way to both the user and the computer identify the instance. When two instances  $i \in O$  and  $i' \in O'$  do not have the same identifier (in OWL, the ID parameter) there is an identifier heterogeneity IHH.

$$IHH(i, i') = \{\exists i \in O \mid \forall i' \in O', id(i) \neq id(i')\}$$

The concepts  $c$  and  $c'$  the instances belong are not considered because maybe they are not known.

**Coordinate heterogeneity:** As already stated, one of the main characteristics of the geographic data is that it has a position over the earth surface. The set of coordinates of a given instance  $i \in O$  can be defined as a unary function  $coord(i)$ .

If two instances  $i \in O$  and  $i' \in O'$  do not have the same spatial position, there is a coordinate heterogeneity ICH.

$$ICH(i, i') = \{i \in O, i' \in O' \mid coord(i) \neq coord(i')\}$$

**Attribute heterogeneity:** When a property of a concept is a data type property it represents an attribute, i.e., a property to which the allowed values are string, float, integer, etc. In this case the relation  $at(i, p, val)$  can be identified as  $at(i, p, v)$ .

When two instances  $i \in O$  and  $i' \in O'$  have different values  $v$  for the same data type property  $p$  there is an attribute heterogeneity IAH.

$$IAH(i, i') = \{\exists at(i, p, v) \in O \mid \forall at(i', p', v') \in O', (p \equiv p') \wedge (v \neq v')\}$$

**Relationship heterogeneity:** When a property of a concept is an object type property it represents a relationship, i.e., a property which allowed values are



instances of other concepts. In this case the relation  $vp(i, p, val)$  can be identified as  $rl(i, p, i_x)$ .

When two instances  $i \in O$  and  $i' \in O'$  have associated, respectively, the instances  $i_x$  and  $i'_x$  which represent different concepts, there is a relationship heterogeneity IRH.

$$IRH(i, i') = \{\exists rl(i, p, i_x) \in O \mid \forall rl(i', p', i'_x) \in O', (i_x \neq i'_x)\}$$

## 4 The IG-MATCH Matchmaker

In this section we detail our proposal for a geographic matchmaker. We start from the IG-MATCH architecture and then explain the procedure to measure the degree of similarity of two compared geographic ontologies, at both the concept and the instance-level. The procedure starts by determining the similarity among the concepts of the two ontologies and based on the results the instance similarity is measured.

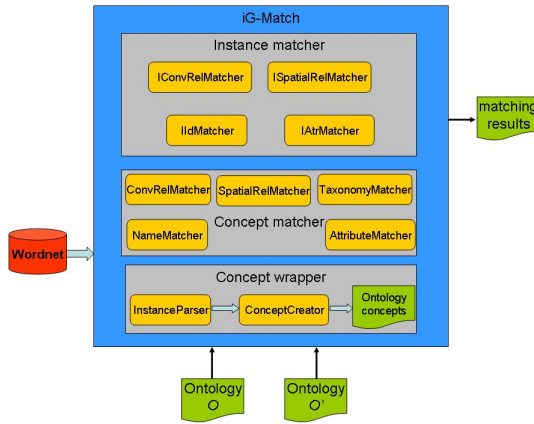
### 4.1 IG-MATCH Architecture

The IG-MATCH architecture is depicted in Figure 3. It receives as input two ontologies  $O$  and  $O'$  to be compared and produces as an output the mappings among the concepts and instances of these two ontologies in a tabular format as an XML file. Furthermore, it makes use of an external dictionary or thesaurus, such as WordNet [4], for the linguistic affinity measure, during the process of matching.

The IG-MATCH architecture is composed by three different layers, namely *ConceptWrapper*, *Concept matcher* and *instance matcher*. Each layer works as an independent module, and depending on the input ontologies it is executed or not. In the following we provide the details for each one of the layers.

**ConceptWrapper:** It may happen that one (or both) of the input ontologies  $O$  and  $O'$  contain only the data (instances), without the explicit definition of the concepts, properties and restrictions. As we are dealing with semi-structured information (in our case, OWL - Ontology Web Language), even if the structure of a concept is not explicitly defined, it may be inferred from its instances. This can be done by parsing the document XML's tags in a kind of reverse engineering process. For example, if we consider the following piece of OWL document (taken from the ontology  $O$  example)

```
<City rdf:ID="Milan">
  <Population>1.568.920</Population>
  <hasAdm rdf:resource="#Adm2006"/>
  <hasGeometry>
    <Polygon rdf:ID="pol1"/>
  </hasGeometry>
</City>
```



**Fig. 3.** IG-MATCH architecture

it is possible to infer that *Milan* is an instance of a concept called *City* which has a data type property *population* of type integer. Furthermore it has an object type property *hasAdm* associating it to an instance identified as *Adm2006* and another object type property *hasGeometry* which points to an instance of *Polygon*. Once one discovers to which concept the instance *Adm2006* belongs, it is possible to define the allowed values for the property *hasAdm* at the concept-level. In other words, it is possible to know to which concept  $c_x$  the concept *City* is associated.

For the case when there are instances  $i \in O$  and/or  $i' \in O'$  to which there is not a concept  $c$  or  $c'$  explicitly defined, before of matching them (the instances) the concepts to which they belong must be determined. This is mandatory because the instance similarity measurement depends on the similarity of the concepts they instantiate, i.e., the instances are compared only if they belong to concepts already identified as equivalents or very similar. As the input ontologies must be described in a semi-structured language, the *ConceptWrapper* first parses the document and extracts the structure of the concept the instance belongs, from its tags. This is done by the *InstanceParser* module. Then, the inferred concepts are actually defined, by the *ConceptCreator* module. The output of this layer is a file, called *Ontology Concepts* describing concepts not explicitly defined in the incoming ontologies  $O$  and  $O'$ .

**ConceptMatcher:** When all the concepts which have associated instances are known, the *ConceptMatcher* layer performs the process of similarity measurement at the concept-level. This process is performed in two phases. Firstly, the name (module *NameMatcher*) and attribute similarities (data type properties - module *AttributeMatcher*) are measured. The second phase uses the results obtained in the first as parameters to calculate the similarity regarding the taxonomies (module *TaxonomyMatcher*), conventional relationships (object type properties - module *ConvRelMatcher*) and spatial relations (object type properties - module *SptRelMatcher*). We decided to perform the similarity measurement

in two phases because to infer if two relations are equivalent it is mandatory to know how similar are the associated concepts. At last the results of each module are combined and a concept mapping document is produced.

**InstanceMatcher:** Once the process of identifying similarities between concepts is finished, the *Instance Matcher* layer is executed. Using the results produced by the *ConceptMatcher* layer, the *InstanceMatcher* performs the instance similarity measurement. Only instances belonging to equivalent (or very similar) concepts  $c \in O$  and  $c' \in O'$  are compared. The output produced by this layer is a document containing the instance mapping results.

The modules *IIdMatcher*, *IAttrMatcher*, *ICovRelMatcher* and *ISpatialRelMatcher* perform the similarity measurement between two instances. Again we chose a two phases processing. In the first phase the *id* similarity (module *IIdMatcher*), the attribute value similarity (data type properties - module *IAttrMatcher*) are analyzed. Based on the partial results obtained, the second phase performs the conventional relationship (module *ICovRelMatcher*) and spatial relationship (module *ISpatialRelMatcher*) similarity measurement.

## 4.2 Concept Matching

**Name Similarity:** To measure the similarity between the terms which nominate a concept, IG-MATCH is conceived to execute a three-step procedure.

- I. Verify if the term  $t(c)$  for the concept  $c \in O$  is exactly equal to the term  $t(c')$  for the concept  $c' \in O'$
- II. Search in an external dictionary or thesaurus, such as the WordNet [4] the level of linguistic affinity between the terms  $t(c)$  for the concept  $c \in O$  and the term  $t(c')$  for the concept  $c' \in O'$ . We call this the  $SYN(c, c')$  function. This function returns a value within  $[0,1]$ , where 1 means the terms are synonyms and 0 means they are not related at all.
- III. Using a string comparison metric, confront  $t(c)$  for the concept  $c \in O$  against  $t(c')$  for the concept  $c' \in O'$ . In this work we adapt the Stoilos et al. [5] metric, which considers all the common substrings the two compared strings share and also the JaroWinkler metric.

$$SimName(c, c') = \frac{2 * \frac{length(max(ComSubstring(t(c), t(c'))))}{length(t(c)) + length(t(c'))} + JaroWinkler(t(c), t(c'))}{2} \quad (1)$$

Step II is executed only if step I returns 0. Step III is executed only if the step II does not return a satisfactory value, which is defined by the user.

The name similarity measure returns a value within  $[0,1]$  where 0 means the terms are completely different and 1 that they are exactly equals (or synonyms).

**Property Similarity:** As the property heterogeneity is classified in attribute (AH), relationship (RH), geometric (GH) and spatial relationship (SH), IG-MATCH is designed to separately measure the similarity for each one of these aspects.

**Attribute similarity:** To measure the similarity between an attribute  $a(c, an, dtp) \in P$  in an ontology  $O$  and an attribute  $a(c', an', dtp') \in P'$  in ontology  $O'$  the two components to be analyzed are the attributes' names and data types.

The similarity regarding the attributes' name  $an$  is measured in a similar way to the one used to concept names. The main difference is that only the steps I and II are performed. This means that first is checked if the attributes' names  $an$  and  $an'$  are equal and in case they are not the linguistic affinity is calculated.

$$SimNAt(a(c, an, dtp), a(c', an', dtp')) = \begin{cases} 1 & \text{if } an = an' \\ SYN(an, an') & \text{otherwise} \end{cases}$$

The similarity of data types is measured by checking if the data types are the same ( $dtp = dtp'$ , such as both integer or string) or if one is a subclass of the other ( $dtp \subseteq dtp'$  or  $dtp \supseteq dtp'$ , such as float and integer).

$$SimDAt(a(c, an, dtp), a(c', an', dtp')) = \begin{cases} 1 & \text{if } (dtp \subseteq dtp') \vee (dtp \supseteq dtp') \\ 0 & \text{otherwise} \end{cases}$$

Each attribute has an associated weight  $\varepsilon$ , corresponding to its relevance to the concept.  $\varepsilon$  is given by

$$\varepsilon = 1 - (\min((\frac{Ca - 1}{C})(\frac{Ca' - 1}{C'}))) \quad (2)$$

which means that the less concepts have an attribute, the more relevant it is. In the equation,  $Ca$  is the number of concepts having the attribute  $a$  and  $C$  is the total number of concepts of the ontology.

The final final measure of the attribute similarity is given by

$$SimAt(c, c') = \frac{\sum \max((\delta * SimNAt(a_i, a_j) + (1 - \delta) * SimDAt(a_i, a_j)) * \varepsilon)}{|A \cup A'|} \quad (3)$$

where  $\delta$  is the weight for the attribute name similarity.  $A$  is the subset of  $P$  which contains only attributes (data type properties).

**Conventional relationships similarity:** To measure the similarity regarding the conventional relationships, two components that determine the relationship heterogeneity  $RH(c, c')$  have to be considered: (1) the concepts  $c_x$  and  $c'_x$  associated, respectively, to  $c$  and  $c'$  and, (2) the relationship cardinalities. The name of the property that defines the association can be ignored because many times

it is not semantic relevant. The conventional relationship similarity between two concepts  $c \in O$  and  $c' \in O'$  is given by:

$$\begin{aligned} \text{SimRel}(c, c') = \\ \frac{\sum cr(c, rn, c_x, \text{minCard}, \text{maxCard}) \cap cr(c', rn', c'_x, \text{minCard}', \text{maxCard}')}{|CR(c) \cup CR(c')|} \end{aligned} \quad (4)$$

where  $CR$  and  $CR'$  are, respectively, the subset of properties from  $P$  and  $P'$  which correspond to conventional relationships involving, respectively,  $c$  and  $c'$ .

The computation of the conventional relationship similarity is based on the results obtained by the similarity name ( $\text{SimName}(c, c')$ ) measurement. This is due the necessity of determining if the concepts  $c_x$  and  $c'_x$  are equivalent. If  $c$  is associated to a concept  $c_x$  and  $c'$  is associated to a concept  $c'_x$  and the name similarity  $\text{SimName}(c_x, c'_x)$  is higher than a certain threshold the relationships are considered as equivalent, if the cardinalities are also equal.

**Geometric similarity:** Because of the possibility of having the same phenomenon described using different spatial representations, in IG-MATCH we do not compare directly the geometry of the compared concepts  $c$  and  $c'$ . Instead, the geometry is used in the spatial relationship similarity measure.

**Spatial relationships similarity:** To measure the similarity between two concepts regarding the spatial relationships, the three components which cause the spatial heterogeneity must be considered: the concepts  $c_x$  and  $c'_x$  associated, respectively, to  $c$  and  $c'$ , the cardinalities of the relationships, and the names  $rn$  and  $rn'$  of the relationships. The name of the association cannot be ignored because for the spatial relations the names are, in general, standardized and semantically relevant. For example, although *River crosses City* and *River inside City* involve the same concepts, they do not mean the same.

The spatial relations considered in IG-MATCH are the directional and the topological. We do not measure the similarity of the metric relationships because in general they are calculated by a GIS and not stored in the ontology. The similarity measurement of the spatial relationships between two concepts  $c \in O$  and  $c' \in O'$  is given by:

$$\begin{aligned} \text{SimSpt}(c, c') = \\ \frac{\sum sr(gc, rn, gc_x, \text{minCard}, \text{minCard}) \cap sr(gc', rn', gc'_x, \text{minCard}', \text{maxCard}')}{|SR(c) \cup SR(c')|} \end{aligned} \quad (5)$$

where  $SR$  and  $SR'$  are, respectively, the subsets of  $P$  and  $P'$  that contain the spatial relationships.

Due to the possibility of multi-spatial representation for a geographic concept, the similarity measurement regarding to topological relationships must take into consideration the geometries of the concepts. Depending on the geometry of the involved concepts  $c$  and  $c'$  relationships with different names are equivalent. An exhaustive study of the topological equivalences based on the geometries is presented in [6] and is used in the present work.

For the directional relationships the geometry is not relevant, because the relationships do not depend on the geometric shapes but on the spatial coordinates. As at the concept-level the coordinates are not defined, the directional relationships is measured in terms of the restrictions of the concepts. For example, on the definition of a concept *Bridge* there may be a restriction that says that it must be *above* a concept *River*.

The computation of the spatial relationship similarity is based on the results obtained by the similarity name ( $SimName(c, c')$ ) measurement. This is due the necessity of determining if the concepts  $c_x$  and  $c'_x$  are equivalent. If  $c$  is associated to a concept  $c_x$  and  $c'$  is associated to a concept  $c'_x$  and the name similarity  $SimName(c_x, c'_x)$  is higher than a certain threshold the relationships are considered as equivalent, if the other components (cardinalities and relationship names) are also equivalent.

**Hierarchy (Axiom) Similarity:** As defined in section 3.1, two concepts have hierarchy heterogeneity when there are differences in the concepts present in the superclasses tree. The superclass similarity is then given by the number of common superclasses of the concepts  $c$  and  $c'$  divided by the total number of superclasses of both concepts, as follows.

$$SimHier(c, c') = \frac{\sum(h(c, c_x) \cap h(c', c'_x)) * \psi}{|H(c) \cup H(c')|} \quad (6)$$

where  $\psi$  is the difference of the superclasses level. If both classes  $c_x$  and  $c'_x$  are at the same distance from the concepts  $c$  and  $c'$ , respectively,  $\psi$  is equal to 1. Otherwise,  $\psi$  is decreased.  $SUP(c)$  is the number of superclasses of the concept  $c$ , direct or indirect. The similarity measure is a value within  $[0,1]$ .

**Overall Similarity:** The final value for the similarity between two concepts  $c \in O$  and  $c' \in O$  is a weighted sum which considers all the similarities detailed previously.

$$Sim(c, c') = WN * SimName(c, c') + WA * SimAt(c, c') + WH * SimHier(c, c') + WR * SimRel(c, c') + WS * SimSpt(c, c') \quad (7)$$

where WN, WA, WH, WR and WS are, respectively, the weights for the name, attributes, hierarchy, conventional relationships and spatial relationships similarities. The sum of these weights must be 1, and thus the value of  $Sim(c, c')$  is within  $[0,1]$ . The combination of these parameters to achieve bests results is yet an open issue.

### 4.3 Instance Matching

Two instances  $i \in O$  and  $i' \in O'$  are compared only if the concepts  $c$  and  $c'$  they instantiate were already identified as equivalents. The instance similarity measurement is based on five main components, which may cause the instance

heterogeneity: (1) the instance identifier, (2) the value of the descriptive attributes (data type properties), (3) the value of the descriptive relationships (object type properties), (4) the value of the spatial relations (spatial object type properties) and, (5) the spatial position of the instances (coordinates).

When a concept is instantiated, each associated property has a value  $vp = (i, p, val)$ . When two instances are compared, only the equivalent properties are verified, i.e., if  $p \equiv p'$ , which is determined previously, in the concept similarity measure phase. Thus, the only component to be confronted is the value ( $val$ ) from the triple. The similarity among the property values  $vp = (i, p, val)$  of two instances depends on the type of the property.

**Identifier:** When measuring the similarity between two instances  $i$  and  $i'$ , the instance identifier has to be considered. As mentioned in section 3.1 the  $id$  is the instance component property which represents the unique identifier of an instance, i.e., the  $id$  component of the 4-tuple cannot be the same for two instances in the same ontology  $O$ . The measurement of the similarity between the identifiers  $SimIID(i, i')$  for two instances  $i$  and  $i'$  is given by

$$SimIAS(i(type, id, vP, coord), i(type', id', vP', coord')) = \frac{(2 * \sum_i \frac{length(max(ComSubstring_i))}{length(id)+length(id')}) + (JaroWinkler(id, id'))}{2} \quad (8)$$

**Properties Similarity: Attributes:** In the case of a data type property, i.e., an attribute, to which the allowed values are numeric, a simple equality comparison is performed:

$$SimIAtN(vp(i, p, val), vp(i', p', val')) = \begin{cases} 1 & \text{if } val = val' \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

If the numeric types are different, for example *integer* x *float*, only the common part is compared. This means that if one property value is 10 and the other is 10.5 only the integer part of the numbers is compared.

In the case of a data type property to which the allowed values are text (string) the similarity measurement is performed according to the string metric similarity defined for the concept-level.

$$SimIAS(vp(i, p, val), vp(i', p', val')) = \frac{(2 * \sum_i \frac{length(max(ComSubstring_i))}{length(val)+length(val')}) + (JaroWinkler(val, val'))}{2} \quad (10)$$

**Relationships:** If the property is an object type property, it represents a relationship. At the instance-level, as we are concerned if the associated instances are equivalent, for both conventional and spatial relationships the similarity measure is the same. We simply compare if the instances  $i_x$  and  $i'_x$  associated, respec-

tively, to  $i$  and  $i'$  are equivalent. The instances  $i_x$  and  $i'_x$  are the *val* component of the triple  $vp(i, p, val)$ .

$$SimIR(vp(i, p, val), vp(i', p', val')) = \begin{cases} 1 & \text{if } val = val' \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

If a property is present in only one of the compared instances, i.e., it is associated to only one of the concepts, the similarity regarding that property is considered zero. Hence, the final equation for measuring the similarity between two instances  $i$  and  $i'$  is

$$SimPrp(i, i') = \frac{\sum SimIAtN + \sum SimIAtS + \sum SimIR}{|vP(i) \cup |vP(i')|} \quad (12)$$

**Geographic Coordinates:** The geographic coordinates are aspects which play a crucial role in the integration of geographic instances. Although the spatial position of an instance may vary along time, the coordinates may be of great use in most of the cases.

To compare the similarity regarding the geographic coordinates from two geographic instances  $i \in gc$  and  $i' \in gc'$ , first it is necessary to reduce them to the same geometry. Thus, all the instances are transformed to points. In the case of a line string, this is done by the coordinates  $(x, y)$  of the middle of the line string. In the case of a polygon the  $(x, y)$  coordinates of the centroid are used. Then, given two pair of coordinates  $(x, y) \in i$  and  $(x', y') \in i'$  the similarity is measured by the inverse of the euclidian distance between these two pairs of coordinates, as follows:

$$SimCoord(i, i') = \frac{1}{dist(i, i')} \quad (13)$$

The coordinates similarity may not be used for the final similarity measure, but this can exclude some pairs of instances which are located too far from each other. Thus, if the coordinates similarity does not reach a certain threshold, the pair  $(i, i')$  is excluded from the list of possible matches.

**Overall Similarity:** The final similarity value when comparing two instances  $i$  and  $i'$  is then given by:

$$SimInst(i, i') = \rho * SimIID(i, i') + (1 - \rho) * SimPrp(i, i') \quad (14)$$

where  $\rho$  is the weight for the identifier similarity and can be a value within  $[0,1]$ . In the case the instances are not geographic, i.e., they instantiate conventional concepts, the coordinate similarity is not considered. Furthermore, the spatial relationships similarity is also ignored.

## 5 Related Works

There are a number of proposals trying to address the problem of geographic information integration and mapping. However, none of them cope all the issues involved in the process.



At the concept-level, Stoimenov and Djordjevic-Kajan’s [7] and Rodriguez and Egenhofer’s [8] proposals deal with the name, hierarchy and property heterogeneities, especially attributes and part-of relationships. Kokla, Kavouras and Tomai [9,10] consider only the description of the concepts, from which they extract the information used for solving the name, attribute and some spatial relationships heterogeneities. The same idea of extracting the needed information is adopted in [11], but considering only the name heterogeneity and two context features, the spatial relations and geometry [12]. Quix et al. [13] developed a matcher specific for geographic features, to be used together with an existing conventional matcher for non-geographic features (attributes and part-of relationships and hierarchies). Dobre, Hakimpour and Dittrich [14] proposal consider the hierarchies as well as attributes and relationships in the matching process. The former is also the only aspect considered in Cruz et al.’s proposal [15]. Besides attributes, hierarchy and relationships, the work of Sotnykova et al. [3] also considers spatiality (spatial relations) and temporality in the integration process.

At the instance level, most of the works consider that if two instances belonging to matching concepts are in the same spatial position, i.e., have the same spatial coordinates, probably they would refer to the same real world phenomena [16,17,18,19]. The approaches of Sehgal, Getoor and Viechnicki [16], Worboys and Duckham [18] and Beeri et al. [17], however, are limited to instances with point geometries. Besides the point geometry, Volz [19] considers also the line geometry. Furthermore, in that proposal also the topological relationships are considered. In all proposals the scales and reference system must be the same for the two ontologies. In addition to the spatial position, in [16] the name of the instances in the matching process is also considered, and for that they use some string-distance based metrics.

## 6 Conclusions and Future Directions

When integrating geographic information, the matching plays a central role in order to achieve correct results. However, the matching process is not trivial and due to the particular characteristics of the geographic data (geometry, spatial location and spatial relationships) it cannot be performed just as the conventional, non-geographic, matching. Furthermore, considering only the spatial features of the geographic data is also a naive approach, because two elements (concepts or instances) may be described using different spatial representations, and then these features may be not enough to determine if the elements are equivalent or not.

In this paper we presented an ontology reference model and the set of heterogeneities that may occur when comparing two ontologies, both at the concept and at the instance-level. Based on them we proposed the IG-MATCH, a software architecture that implements a methodology for geographic ontology matching. IG-MATCH’s methodology is based on a number of metrics, one for each type of heterogeneity. Some of the metrics are for ontologies in general, which means

that can be applied for matching non-geographic ontologies as well, and some of them are specially tailored for geographic ontology matching.

At the moment, IG-MATCH depends on the user to determine the parameters for the combination of the different metrics, i.e., the user has to set the values for WN, WA, WH, WR and WS at the concept-level and for  $\rho$  at the instance-level. For the future we plan to develop a methodology for automatic tuning this parameters according to the input ontologies. The weight for each parameter may vary depending on the granularity of the ontologies.

Another important issue not studied so far and not addressed in the related works is the role of the metadata in the matching process. For example, if two instances are described using different reference systems, their spatial coordinates will not be the same, even if they refer to the same spatial position. Furthermore, the period (date) of acquisition of the data is also important, because many properties may change along time, such as the population of a city, the coverage (location) of a forest, and so on.

## References

1. Aronoff, S.: *Geographic Information Systems: A Management Perspective*. WDL Publications (1991)
2. Fonseca, F.T., Davis, C.A., Camara, G.: Bridging ontologies and conceptual schemas in geographic information integration. *GeoInformatica* 7(4), 355–378 (2003)
3. Sotnykova, A., Vangenot, C., Cullot, N., Bennacer, N., Aufaure, M.A.: Semantic mappings in description logics for spatio-temporal database schema integration. [20] 143–167
4. Miller, G.A.: Wordnet: A lexical database for english. *Commun. ACM* 38(11), 39–41 (1995)
5. Stoilos, G., Stamou, G., Kollias, S.: A string metric for ontology alignment. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) *ISWC 2005*. LNCS, vol. 3729, Springer, Heidelberg (2005)
6. Belussi, A., Catania, B., Podestá, P.: Towards topological consistency and similarity of multiresolution geographical maps. In: *GIS'05: Proceedings of the 13th annual ACM international workshop on Geographic information systems*, Bremen, Germany, pp. 220–229. ACM Press, New York (2005), doi:10.1145/1097064.1097096
7. Stoimenov, L., Djordjevic-Kajan, S.: An architecture for interoperable gis use in a local community environment. *Computers and Geosciences* 31, 211–220 (2005)
8. Rodriguez, M.A., Egenhofer, M.J.: Determining semantic similarity among entity classes from different ontologies. *IEEE Trans. Knowl. Data Eng* 15(2), 442–456 (2003)
9. Kavouras, M., Kokla, M., Tomai, E.: Comparing categories among geographic ontologies. *Computers & Geosciences* 31(2), 145–154 (2005)
10. Kokla, M., Kavouras, M.: Semantic information in geo-ontologies: Extraction, comparison, and reconciliation. [20] 125–142
11. Kuhn, W.: Modeling the semantics of geographic categories through conceptual integration. [21] 108–118

12. Schwering, A., Raubal, M.: Spatial relations for semantic similarity measurement. In: Akoka, J., Liddle, S.W., Song, I.-Y., Bertolotto, M., Comyn-Wattiau, I., van den Heuvel, W.-J., Kolp, M., Trujillo, J., Kop, C., Mayr, H.C. (eds.) *Perspectives in Conceptual Modeling*. LNCS, vol. 3770, pp. 259–269. Springer, Heidelberg (2005)
13. Quix, C., Ragia, L., Cai, L., Gan, T.: Matching schemas for geographical information systems using semantic information. In: Meersman, R., Tari, Z., Herrero, P. (eds.) *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*. LNCS, vol. 4278, pp. 1566–1575. Springer, Heidelberg (2006)
14. Dobre, A., Hakimpour, F., Dittrich, K.R.: Operators and classification for data mapping in semantic integration. In: Song, I.-Y., Liddle, S.W., Ling, T.-W., Scheuermann, P. (eds.) *ER 2003*. LNCS, vol. 2813, pp. 534–547. Springer, Heidelberg (2003)
15. Cruz, I.F., Sunna, W., Chaudhry, A.: Semi-automatic ontology alignment for geospatial data integration. In: Egenhofer, M.J., Freksa, C., Miller, H.J. (eds.) *GIScience 2004*. LNCS, vol. 3234, pp. 51–66. Springer, Heidelberg (2004)
16. Sehgal, V., Getoor, L., Viechnicki, P.D.: Entity resolution in geospatial data integration. In: *ACM-GIS'06*, ACM Press, New York (2006)
17. Beeri, C., Doytsher, Y., Kanza, Y., Safra, E., Sagiv, Y.: Finding corresponding objects when integrating several geo-spatial datasets. In: *GIS '05: Proceedings of the 13th annual ACM international workshop on Geographic information systems*, Bremen, Germany, pp. 87–96. ACM Press, New York, NY, USA (2005), doi:10.1145/1097064.1097078
18. Worboys, M.F., Duckham, M.: Integrating spatio-thematic information [21] 346–362
19. Volz, S.: Data-driven matching of geospatial schemas. In: Cohn, A.G., Mark, D.M. (eds.) *COSIT 2005*. LNCS, vol. 3693, pp. 115–132. Springer, Heidelberg (2005)
20. Spaccapietra, S., Zimányi, E.: Journal on Data Semantics III. In: Spaccapietra, S., Zimányi, E. (eds.) *Journal on Data Semantics III*. LNCS, vol. 3534, Springer, Heidelberg (2005)
21. Egenhofer, M.J., Mark, D.M.: Geographic Information Science. In: Egenhofer, M.J., Mark, D.M. (eds.) *GIScience 2002*. LNCS, vol. 2478, pp. 25–28. Springer, Heidelberg (2002)

# Local Topological Relationships for Complex Regions

Mark McKenney, Alejandro Pauly, Reasey Praing, and Markus Schneider\*

Department of Computer & Information Science & Engineering  
University of Florida  
Gainesville, FL 32611, USA  
{mm7,apauly,rpraing,mschneid}@cise.ufl.edu

**Abstract.** Topological relationships between spatial objects are important for querying, reasoning, and indexing of data within spatial databases. These relationships are qualitative and respond to questions about the relative positions (e.g., disjointedness or containment) of spatial objects. Several models have been proposed that effectively define formal sets of topological relationships between *simple* spatial data types. The generalization of topological relationship models to *complex* spatial data types, which are roughly defined as multi-component versions of their simple counterparts, has raised awareness of the fact that these models only provide a *global* view of topological relationships whereas details of the topological relationships between individual components of the spatial objects involved are often ignored. In this paper, we introduce a fine-grained view on topological relationships between complex regions. Our model focuses on leveraging information about the *local* topological relationships that hold between the components of two spatial objects, thereby providing a *localized* view of the overall global topological relationship.

## 1 Introduction

The exploration of relationships between spatial objects is an important topic in fields such as robotics, VLSI design, linguistics, CAD, and GIS. Object relationships can be used not only to learn information about the objects involved but also for inferring new non-explicit information as well as creating fast access and indexing structures in spatial databases. Specifically, topological relationships have been the focus of extensive research for a long time. This research includes the design of models of topological relationships between all types of spatial objects as well as related topics like the exploration of topological relationships as a reasoning tool.

Models for topological relationships have predominantly considered *simple* spatial data types. A *simple point* object is defined as a single pair of coordinates, a *simple line* object is given as a non self-intersecting connected curve, and a *simple region* object is represented as an areal object topologically equivalent to a closed disc. A well-known model that defines the topological relationships between simple spatial objects is the 9-*intersection* model (9IM). The commonly known set of eight topological relationships originally defined by the 9IM between simple regions includes the relationships *overlap*, *meet*, *inside*, *contains*, *coveredBy*, *covers*, *equal*, and *disjoint*.

---

\* This work was partially supported by the National Science Foundation under grant number NSF-CAREER-IIS-0347574.

More recently, the 9IM has effectively been applied to the latest generation of *complex* spatial data types that, for example, have been propagated by the Open GeoSpatial Consortium. Roughly, a *complex point* is defined by a set of disjoint simple points. A *complex line* is composed of a set of blocks of connected simple lines. A *complex region* is defined as a set of one or more faces, each possibly containing holes. The application of the 9IM to complex spatial data types has raised awareness of the *global* nature of the 9IM. That is, the 9IM considers the interior, exterior, and boundary point sets of the whole objects, and ignores the fact that complex spatial objects are composed of individual and separate components. As a result, *local* topological information regarding the relationship between individual components from each object is lost. Consider a scenario where an oil spill has occurred on a coral reef system. If the oil spill overlaps a section of the reef, then the global relationship would be overlap. However, there may be isolated portions of the reef which are not yet affected by the oil spill, but will be soon if no action is taken. For example, a portion of the reef may meet the edge of the oil spill. The overall topological relationship according to the 9IM will be considered as an *overlap* and the local *meet* relationship will be ignored. If we can identify the local meet interaction, then a barrier can be constructed to save this portion of the reef. This example illustrates the domination of a local overlap over a local meet. But it turns out that this *dominance* problem is not the only way in which local information is lost within the 9IM. For example, an overlap can exist globally when one face of a region *B* is *coveredBy* a face of another region *A* and vice versa a face of *A* is contained in the other face of *B*. Even without the existence of a local overlap, a global overlap appears due to the *composition* of other local relationships.

In this paper, we propose a model for topological relationships between complex regions that is at least as expressive as the 9IM with respect to global information, and at the same time, is able to retain information regarding local relationships between individual components of these regions. We achieve this by defining global topological relationships based on the existence of local topological relationships between the components of the objects involved. This work provides a more fine-grained tool for querying spatial objects and increases the expressive power available to a spatial database user. Furthermore, we provide a notation to specify spatial queries that utilize local topological information.

Section 2 introduces previous related work. Section 3 formally defines the topological relationships between simple regions with holes which are necessary for the formal description of our model. In Section 4, the dominance and composition problems are detailed. Our *locality aware* model of topological relationships between complex regions is presented in Section 5. We consider the expressiveness of our locality aware model compared to the global model in Section 6. In Section 7, we connect our concept with the user by providing the notation that allows our model to be embedded into a common database query language. Finally, Section 8 gives conclusions and future work.

## 2 Related Work

This section briefly introduces the most important background concepts that are necessary to fully understand and to solve the dominance and composition problems



**Fig. 1.** (a) Illustrates the interior, boundary, and exterior point sets of a complex region composed of three faces. (b) Illustrates a single face, also denoted as a simple region with holes.

introduced earlier. We concentrate on introducing the definition of complex regions upon which this paper is based. We also describe the 9IM and cover a previous approach that deals with preserving local topological relationships.

We are interested in distinguishing between the interior, boundary and exterior point sets of complex regions, as defined in [11]. Based on this definition, Figure 1(a) illustrates a complex region with its interior, exterior, and boundary.

In order to be able to describe the topological predicates between components of complex regions, we must define their structural components. Complex regions are composed of *faces*. Each face is regarded as a *simple region with holes*, an example of which is illustrated in Figure 1(b). A simple region with holes has a connected interior and (possibly) disconnected exterior and boundary due to the existence of holes. Informally, a simple region with holes is made up of an outer polygon denoting its outer boundary and zero or more hole polygons representing its holes. All holes must be completely contained within the outer polygon and can share a finite number of boundary points with the outer cycle and with other holes. We denote the set of all simple regions with holes as SRH.

Topological relationships between spatial objects can be defined by the 9-intersection model by evaluating the non-emptiness of the intersection between all combinations of the interior ( $^{\circ}$ ), boundary ( $\partial$ ) and exterior ( $^{-}$ ) of the objects involved. A unique  $3 \times 3$  matrix with Boolean values filled as illustrated in Figure 2 describes the topological relationship between each pair of spatial objects.

Originally defined for simple regions, the 9IM has been extended to handle simple regions with holes [2], complex spatial objects [11], and *composite* regions (i.e., complex regions without holes) [3]. The model in [2] characterizes the topological relationships between two simple regions with holes as the conjunction of topological relationships between their underlying simple regions (the holes are considered simple regions). For two simple regions with holes  $A$  and  $B$  with  $n$  and  $m$  holes respectively, a matrix of  $(n+1)(m+1)$  elements represents the topological relationship between  $A$  and  $B$ . This means that under this model, the number of topological relationships between two simple regions with holes is dependent on the number of holes in each region, resulting in an arbitrary number of predicates. To avoid this, we identify the finite set of topological relationships between simple regions with holes based on the 9IM (Section 3) that is independent of the number of holes and that we use as a basis for the rest of this paper.

In [4], the authors introduce a model that preserves local topological relationships between composite regions while still maintaining global information. The approach used to define that model is a precursor to the approach used in this paper, where we

$$\begin{pmatrix} A^\circ \cap B^\circ \neq \emptyset & A^\circ \cap \partial B \neq \emptyset & A^\circ \cap B^- \neq \emptyset \\ \partial A \cap B^\circ \neq \emptyset & \partial A \cap \partial B \neq \emptyset & \partial A \cap B^- \neq \emptyset \\ A^- \cap B^\circ \neq \emptyset & A^- \cap \partial B \neq \emptyset & A^- \cap B^- \neq \emptyset \end{pmatrix}$$

**Fig. 2.** The 9-intersection matrix for topological relationships

see that the more general problem of modeling local and global topological relationships between complex regions is significantly more complex. Furthermore, in this paper we introduce and define the global topological predicates between simple regions with holes. We also define a local topological relationship specification mechanism for querying spatial objects with local spatial information. This mechanism allows the user to fully exploit locality aware models of topological relationships, and it can be applied to the model presented in this paper as well as the model in [4].

### 3 Topological Predicates Between Components of Complex Regions

Given that the set of *simple regions with holes* is a subset of the set of complex regions, it follows that the set  $TP_{srh}$  of topological relationships that can hold between two simple regions with holes is a subset of the set  $TP_{cr}$  of topological relationships that can hold between two complex regions. This is due to the fact that a SRH object represents a specific instance of a complex region; thus, any topological relationship defined for such an object must also be defined for complex regions. This means that we must find the elements (if any) from  $TP_{cr}$  that are not members of  $TP_{srh}$ . To do so, we impose a new constraint to the relationships denoted by the 9-intersection matrices as originally defined in [1].



















**Lemma 1.** *Let  $R$  and  $S$  be two simple regions with holes. If the interior of  $R$  intersects both the interior and exterior of  $S$ , then the interior of  $R$  must also intersect the boundary of  $S$ , and vice versa, i.e.,*

$$\begin{aligned} \forall R, S \in SRH, \\ (R^\circ \cap S^\circ \neq \emptyset \wedge R^\circ \cap S^- \neq \emptyset \Rightarrow R^\circ \cap \partial S \neq \emptyset) \wedge \\ (R^\circ \cap S^\circ \neq \emptyset \wedge R^- \cap S^\circ \neq \emptyset \Rightarrow \partial R \cap S^\circ \neq \emptyset) \end{aligned}$$

*Proof.* It follows from the Jordan-Curve theorem that any point set which intersects both the interior and the exterior of a region but not its boundary must be disconnected. Based on its definition, any SRH must have a connected interior; therefore, it cannot intersect the interior and exterior of another region without intersecting its boundary.  $\square$

This new constraint eliminates 15 of the 33 originally identified topological relationships between complex regions. The remaining 18 relationships represent the members of  $TP_{srh}$ . We must now prove that all 18 can actually hold between two SRH objects and that no further constraints are necessary. We follow the method of proof by drawing and show in Table 1 sample configurations of each relationship.

**Table 1.** The 18 topological predicates between simple regions with holes. One object is shaded dark and the other light as in the *disjoint*, whereas the shared areas have the darkest shade.

 $\begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ disjoint	 $\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ fillingHole	 $\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$ holeFilled	 $\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ meet	 $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ equal	 $\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ inside
 $\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$ holeCoveredBy	 $\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ coveredBy	 $\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ contains	 $\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ holeCovers	 $\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$ covers	 $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ insideOverfill
 $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$ containsOverfill	 $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ insideContains	 $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$ coversCoveredBy	 $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$ coversOverfill	 $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$ coveredByOverfill	 $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ overlap

## 4 Locality of Models for Topological Relationships

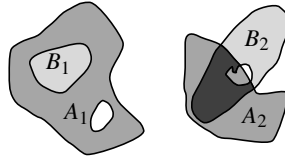
In this section, we introduce the problem of locality (or globality) of models for topological predicates that apply to multi-component spatial objects. The degree of locality of a topological predicate model refers to the ability of the model to distinguish between an overall view of a relationship between spatial objects and the relationships between the individual components of those objects. Section 4.1 identifies three degrees of locality and classifies current models of topological relationships according to these degrees. This classification motivates the need to develop a more comprehensive model in terms of locality. This need is illustrated by two main drawbacks found in the 9IM: the *dominance problem*, which we present in Section 4.2, and the *composition problem*, presented in Section 4.3. Once these drawbacks are identified, we sketch the requirements and procedure to derive a locally and globally aware model for topological predicates between complex regions in Section 4.4.

### 4.1 Views of Topological Predicates

Many models of topological predicates have been proposed in the literature, some of which have been derived from the 9IM model for simple objects; however, there has been little attempt to classify the models in any meaningful way. Here we use an example to motivate a classification of topological predicate models into three *views* of predicates where each view defines a different degree of locality. We provide a single scene of complex region objects (Figure 3) and use it to demonstrate the various views.

The first view of topological predicates we consider is the *global view*. Topological predicates that take into account the global view of a scene equate any given scene to a single topological predicate. For instance, the 9IM for simple regions is a global view model, as is the 9IM for complex regions, which is an extension of the 9IM for simple





**Fig. 3.** Two complex regions, one darker and one lighter. Their common area is shaded darkest.

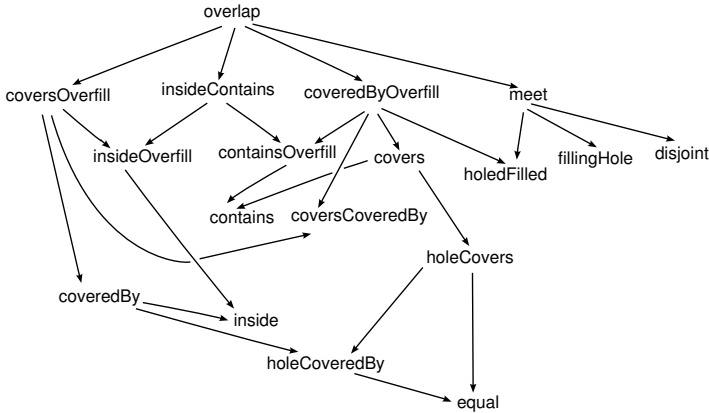
regions. The 9IM for complex regions classifies the objects in Figure 3 as satisfying the *overlap* predicate. However, such a global view does not necessarily indicate complete information about a scene. In this case, regions  $A$  and  $B$  do indeed *overlap*, but a *holeFilled* situation (between  $A_1$  and  $B_1$ ) also exists, as well as a *disjoint*. While the global view of the 9IM does assign a unique predicate to this scene, it effectively hides other interactions between components of the complex regions.

A second category is the *local view* of topological relationships, which can also be characterized as the *existence view*. This view focuses solely on the topological relationships between the individual components of multi-component objects. Queries that take advantage of the local view can typically be posed in the form of an existence question; for example, a user may wish to ask “does there exist a *meet* interaction between object  $A$  and object  $B$ ?”. The local view is fundamentally different from the other views in that it does not attempt to assign a unique predicate to a given configuration of objects. Instead, it assigns a unique predicate to the interaction of any two components between spatial objects. In the case of complex regions, these components are simple regions with holes.

Finally, the *hybrid view* combines the information provided by global and local views to allow multiple local interactions in a given scene to be expressed in a unique global predicate. One example of such a view is given in [3], where the authors present a model for topological predicates which describes a scene as a matrix of predicates where each entry in the matrix represents a 9IM predicate between components of each object. Such a view provides a single, although complex, predicate for the entire scene while maintaining the information about all interactions between all faces of the objects. The advantage of the hybrid view is that a global predicate can be assigned to a scene such that local information is not hidden.

## 4.2 The Dominance Problem

A major drawback to global views of modeling topological relationships is the *dominance problem*. The dominance problem is characterized by a global topological configuration overshadowing the existence of other local topological configurations between a pair of given objects. Figure 3 depicts an arrangement of two complex region objects  $A$  and  $B$  in the plane. As described above, the regions are globally in an *overlap* configuration due to the overlapping of components  $A_2$  and  $B_2$ . However, if the interaction of each face of object  $A$  with object  $B$  is examined independently i.e., *locally*, then in addition to the *overlap* configuration, a *holeFilled* configuration is observed between  $A_1$  and  $B_1$ . Because the global configuration for regions is *overlap*, we say that the *overlap* configuration dominates the *holeFilled* configuration. Another dominated relation in the figure is the *disjoint* configuration between  $A_1$  and  $B_2$ .



**Fig. 4.** The dominance hierarchy for topological predicates between simple regions with holes. An arrow  $p \rightarrow q$  means that  $p$  dominates  $q$ .

In order to cope with the dominance problem, it is necessary to determine which topological predicates dominate which other topological predicates. In other words, we must define a dominance order among the topological predicates. We can determine a dominance order based on the 9-intersection matrices of each topological predicate. For a 9-intersection matrix  $(p_{ij})_{0 \leq i, j \leq 2}$  representing a topological predicate, we define the *not-empty-entry* set  $NE_p$  to be the set of matrix coordinates corresponding to matrix entries that are equal to 1 (*true*). Conversely, we define the *empty-entry* set  $E_p$  to be the set of all matrix coordinates corresponding to entries in matrix  $(p_{ij})_{0 \leq i, j \leq 2}$  that are equal to 0 (*false*). Formally, we define:

$$NE_p = \{(i, j) | 0 \leq i, j \leq 2 \wedge p_{ij} = 1\}$$

$$E_p = \{(i, j) | 0 \leq i, j \leq 2 \wedge p_{ij} = 0\}$$

Let  $TP_{srh}$  be the set of topological predicates between simple regions with holes. For two topological predicates  $p, q \in TP_{srh}$ , we define the *dominates* operator to return *true* if, and only if, the not-empty-entry set of  $q$  is a subset of the not-empty-entry set of  $p$ :

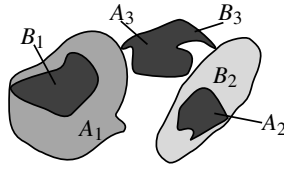
$$\text{dominates} : TP_{srh} \times TP_{srh} \longrightarrow \mathbb{B}$$

$$\text{dominates}(p, q) := \text{if } NE_q \subset NE_p \text{ then } true \text{ else } false$$

Based on the *dominates* operator, for illustration purposes we construct the dominance hierarchy for the elements of the set  $TP_{srh}$  of topological predicates between simple regions with holes as shown in Figure 4. It becomes apparent that the dominance problem only affects the topological predicates between multi-component spatial objects; thus, the dominance hierarchy can be determined for the topological predicates between complex points, lines, and regions, as well as any combination of those.

### 4.3 The Composition Problem

The composition problem arises due to the observation that global topological relationships can be present without actually existing locally. For example in Figure 5, locally



**Fig. 5.** A scene exposing the composition problem of the 9IM

we distinguish the existence of a *disjoint* between  $A_1$  and  $B_2$ , a *meet* ( $A_3, B_2$ ), an *equal* ( $A_3, B_3$ ), a *coveredBy* ( $A_2, B_2$ ) and a *covers* ( $A_1, B_1$ ). If we look at the global 9IM describing the topological relationship between  $A$  and  $B$ , we notice that it represents the *overlap* predicate even though no such *overlap* exists locally. Whereas the dominance problem relates to the fact that the global view hides information about local relationships that do exist, the composition problem hides information about global relationships that do not exist locally.

#### 4.4 Deriving a Hybrid Model for Topological Predicates

We now focus on defining the requirements of a hybrid model for topological predicates between complex regions and developing a procedure to define such a model. The first and obvious requirement is that the model must be able to describe global topological relationships, and at the same time, preserve the necessary information that may be required about the local relationships between components of the objects involved. This is achieved in principle, by defining a model that does not suffer from the dominance or composition problems as described in the previous sections.

The new hybrid model must also be at least as expressive as the 9IM. That is, the new model should be able to distinguish between all (and possibly more) of the relationships that the 9IM between complex regions can distinguish. We define two topological predicate models as equally expressive if there is exactly a one-to-one correspondence between their predicates. Ideally, the new model will have a many-to-one correspondence with the 9IM, yielding one or more predicates for each predicate derived by the 9IM.

The first step in deriving our model is to find a formal characterization of topological predicates between complex regions. This characterization must allow us to describe a topological relationship between two complex regions in such a way that the dominance and composition problems are avoided. Once such a characterization is reached, we must discover all the characterizations that are realizable based on the definition of complex regions. These valid characterizations are now considered topological predicates. Next, we must be able to compare the expressiveness of the 9IM and the new characterizations. This is done in order to ensure that the third requirement set above is fulfilled. The comparison is performed by finding the two-way correspondence between elements in the new characterization and elements (topological predicates) in the global 9IM.

Once we have ensured that the new model is as expressive as the 9IM globally, we must provide a mechanism that will allow the user to exploit the local information retained by the new characterization. This ensures that the hybrid model can be used both from a global, and from a local perspective.

## 5 A Model for Preserving Local Interactions

In this section, we present a novel model of *localized topological relationships* for multi-component spatial objects that does not suffer from the dominance and composition problems. The *localized topological predicate* (LTP) model is a hybrid view of topological relationships in that it utilizes local information to provide a global relationship between two objects. We begin by exploring the characterization of LTP, and then discovering all LTPs that are possible between complex regions. Finally we compare this new model with the 9IM between complex regions and attempt to reach a completely hybrid (local and global) model based on this comparison.

### 5.1 Characterization of Localized Topological Predicates

In order to discover the local topological relationships between complex regions, we define *existence topological predicates* (ETP). Let  $cr$  be the set of all complex regions, and  $ET_{cr}$  be the set of all existence topological predicates between complex regions. For  $p \in TP_{srh}$ , the ETP  $p_e \in ET_{cr}$  is defined as a function  $p_e : cr \times cr \rightarrow boolean$ . For any  $A, B \in cr$ :

$$p_e(A, B) = \begin{cases} true, & \exists A_i \in A, \exists B_j \in B : p(A_i, B_j) = true \\ false, & \text{otherwise} \end{cases}$$

Based on this definition, for complex regions we identify a total of 18 existence topological predicates that exactly correspond to the 18 topological predicates between simple regions with holes, i.e.,  $ET_{cr} = \{disjoint_e, fillingHole_e, holeFilled_e, meet_e, equal_e, inside_e, holeCoveredBy_e, coveredBy_e, contains_e, holeCovers_e, covers_e, insideOverfill_e, containsOverfill_e, insideContains_e, coversCoveredBy_e, coversOverfill_e, coveredByOverfill_e, overlap_e\}$ .

For complex regions  $A$  and  $B$  we define the localized topological predicate that describes their relationship as a conjunctive boolean expression with exactly eighteen clauses, each with a single element that corresponds to a predicate from  $ET_{cr}$  or its negation. That is, an LTP characterizes the topological predicate between two complex regions by asserting which ETPs hold and which do not hold between the complex regions. Let  $E(A, B)$  and  $F(A, B)$  be the sets of ETPs that yield *true* and *false* respectively, for  $A$  and  $B$ , i.e.

$$E(A, B) = \{e \in ET_{cr} \mid e(A, B) = true\}$$

$$F(A, B) = \{f \in ET_{cr} \mid f(A, B) = false\}$$

We define  $l \in LT_{cr}$  (where  $LT_{cr}$  is the set of LTPs between complex regions) as:

$$l(A, B) = \left( \bigwedge_{e \in E(A, B)} e(A, B) \right) \wedge \left( \bigwedge_{f \in F(A, B)} \neg f(A, B) \right)$$

For notation purposes, we represent each LTP as a 18-bit vector where each bit corresponds to an existence predicate in the order shown in Table 1. Based on this definition, we denote the set of all valid 18-bit vectors as  $LTBV_{cr}$ . The element from this set that represents the scenario in Figure 5 is  $[1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$ ,

because the local topological relationships that exist between components in the scene are *disjoint*, *meet*, *equal*, *coveredBy*, and *covers*.

## 5.2 Identifying the Valid Characterizations

In order to identify the elements that belong to  $LT_{cr}$ , we first consider the complete set of 18-bit vectors that can exist (i.e.,  $2^{18}$  such vectors). By considering the semantics of the bits, we can see that not all combinations are possible. A trivial example is represented by the *all-zeros* TBV that we can determine to be topologically invalid due to the fact that every pair of (non-empty) simple regions with holes must satisfy at least one of the eighteen topological predicates. We follow a procedure of *constraint* and *validation* in order to successfully identify the complete set of valid TBVs for complex regions. The first step of the procedure is to eliminate, by way of constraint rules, TBVs that are topologically invalid. The second step entails validating the remaining TBVs. If all remaining TBVs are successfully validated, then we have reached a complete set, otherwise we must identify a new constraint rule. This sequence is repeated until the validation is complete.

The constraint rules for the first step are expressed by way of lemmas that provide formal proof that certain situations within bit vectors cannot concurrently occur. For example, take the complex regions  $A$  and  $B$  such that a component (simple region with hole)  $A_i$  of  $A$  and a component  $B_j$  of  $B$  are *equal*. Besides this local relationship, there exists a local *overlap* relationship between some other component  $A_k$  of  $A$  and  $B_l$  of  $B$ . For such a configuration, we know that so far the *equal* and *overlap* bits are set to 1. But we can also determine that either the *disjoint* or the *meet* bits must be set to account for the relationships between  $A_i$  and  $B_l$  and between  $A_k$  and  $B_j$ . It must be true that  $(A_i \neq A_k) \wedge (B_j \neq B_l)$ , otherwise two components from the same complex region object would have intersecting interiors which are disallowed by the definition of complex regions. Formally,

**Lemma 2.** *Let  $A, B \in SRH$ , let  $A_i, A_k, B_j$ , and  $B_l$  be faces of  $A$  and  $B$ , respectively, and let  $q, r \in TP_{srh}$ :*

$$\begin{aligned} equal(A_i, B_j) \wedge q(A_k, B_l) &\Rightarrow r(A_i, B_l) \text{ where} \\ &(q(A_k, B_l) \Rightarrow A_k^\circ \cap B_l^\circ \neq \emptyset) \wedge r \in \{disjoint, meet\} \end{aligned}$$

$$\begin{aligned} \text{Proof: } equal(A_i, B_j) &\Rightarrow A_i^\circ = B_j^\circ \\ q(A_k, B_l) &\Rightarrow A_k^\circ \cap B_l^\circ \neq \emptyset \\ A_i^\circ = B_j^\circ \wedge A_k^\circ \cap B_l^\circ \neq \emptyset &\Rightarrow A_i^\circ \cap B_l^\circ = \emptyset \\ &\Rightarrow disjoint(A_i, B_l) \vee meet(A_i, B_l) \quad \square \end{aligned}$$

We have identified a total of 17 lemmas, including Lemma 2 above and the trivial case of an all zeros bit vector (Lemma 0). All lemmas other than the trivial all zero case are of the form  $p(A_i, B_j) \wedge q(A_k, B_l) \Rightarrow r(A_i, B_l)$ . Table 2 includes the definitions of  $p$ ,  $q$ , and  $r$  for the remaining 15 lemmas. The proofs for all these follow a structure similar to the proof of Lemma 2; thus, we choose to omit such proofs due to space constraints.

The completeness of these lemmas is validated by a mechanism designed to ensure that each 18-bit vector that is not covered by the constraints, is realizable and valid. The mechanism is based on concepts of topological reasoning that are applied through a theory of *Binary Constraint Networks* (BCN). In the mechanism we are able to reason about topological relationships by computing the composition operation [5]. The

**Table 2.** The remaining lemmas acting as constraints for the invalidation of topological bit vectors

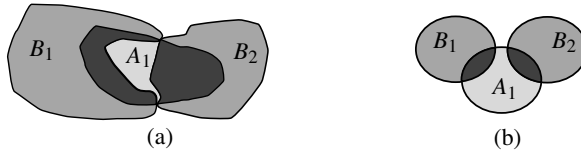
Lemma	$p$	$q$	$r$
3	$p \in \{inside\}$	$q(A_k, B_l) \Rightarrow \overline{A_k} \supseteq \overline{B_l}$	$r \in \{disjoint\}$
4	$p \in \{contains\}$	$q(A_k, B_l) \Rightarrow \overline{A_k} \subseteq \overline{B_l}$	$r \in \{disjoint\}$
5	$p \in \{coveredBy\}$	$q(A_k, B_l) \Rightarrow A_k^\circ \supseteq \partial B_l$	$r \in \{disjoint, meet\}$
6	$p \in \{covers\}$	$q(A_k, B_l) \Rightarrow \partial A_k \subseteq B_l^\circ$	$r \in \{disjoint, meet\}$
7	$p \in \{overlap\}$	$q(A_k, B_l) \Rightarrow \partial A_k \cap B_l^- = \emptyset \wedge A_k^- \cap \partial B_l = \emptyset$	$r \in \{disjoint, meet\}$
8	$p \in \{fillingHole\}$	$q(A_k, B_l) \Rightarrow A_k^- \cap \partial B_l = \emptyset$	$r \in \{disjoint, meet\}$
9	$p \in \{holeFilled\}$	$q(A_k, B_l) \Rightarrow \partial A_k \cap B_l^- = \emptyset$	$r \in \{disjoint, meet\}$
10	$p \in \{holeCoveredBy\}$	$q(A_k, B_l) \Rightarrow \partial A_k \cap \partial B_l \neq \emptyset \vee \partial A_k \cap B_l^\circ \neq \emptyset$	$r \in \{disjoint, meet\}$
11	$p \in \{holeCovers\}$	$q(A_k, B_l) \Rightarrow \partial A_k \cap \partial B_l \neq \emptyset \vee A_k^\circ \cap \partial B_l \neq \emptyset$	$r \in \{disjoint, meet\}$
12	$p \in \{insideOverfill\}$	$q(A_k, B_l) \Rightarrow A_k^- \cap \partial B_l = \emptyset \wedge A_k^- \cap B_l^\circ \neq \emptyset$	$r \in \{disjoint\}$
13	$p \in \{containsOverfill\}$	$q(A_k, B_l) \Rightarrow \partial A_k \cap B_l^- = \emptyset \wedge A_k^\circ \cap B_l^\circ \neq \emptyset$	$r \in \{disjoint\}$
14	$p \in \{insideContains\}$	$q(A_k, B_l) \Rightarrow A_k^- \cap \partial B_l = \emptyset \wedge A_k^- \cap \partial B_l = \emptyset$	$r \in \{disjoint, meet\}$
15	$p \in \{coversCoveredBy\}$	$q(A_k, B_l) \Rightarrow A_k^- \cap B_l^\circ \neq \emptyset \wedge A_k^\circ \cap B_l^- \neq \emptyset$	$r \in \{disjoint, meet\}$
16	$p \in \{coversOverfill\}$	$q(A_k, B_l) \Rightarrow A_k^- \cap \partial B_l = \emptyset \wedge A_k^- \cap B_l^\circ \neq \emptyset$	$r \in \{disjoint, meet\}$
17	$p \in \{coveredByOverfill\}$	$q(A_k, B_l) \Rightarrow \partial A_k \cap B_l^- = \emptyset \wedge A_k^\circ \cap B_l^\circ \neq \emptyset$	$r \in \{disjoint, meet\}$

composition of two relationships  $p$  and  $q$  between  $A$  and  $B$ , and between  $B$  and  $C$  respectively, is able to at least partially derive the relationship  $r$  between  $A$  and  $C$ . A common way to denote this operation is  $p(A, B); q(B, C) \Rightarrow r(A, C)$ .

We model each of the 18-bit vectors as a *Binary Spatial Constraint Network* (BSCN). A BSCN is considered valid if it is *path-consistent* [6]. Path-consistency is computed by using the composition of topological relationships between simple regions with holes. For example take a BSCN  $M$  where  $M_{ij}$  refers to the constraints (relationships) between variables (objects)  $i$  and  $j$ . We say that  $M$  is path-consistent if it holds that  $M_{ij} \subseteq M_{ik}; M_{kj}$  for every  $i, j$ , and  $k$  variable of  $M$ .

To model a bit vector as a BSCN we consider a scene made up of two complex regions  $A$  and  $B$  with  $n$  and  $m$  simple region with holes components respectively such that  $A = \{A_0, A_1, \dots, A_n\}$ , and  $B = \{B_0, B_1, \dots, B_m\}$ . A BSCN is a graph constructed by considering each component as a variable (vertex of graph) and then assigning constraints between each variable as the topological relationships represented in the edges of the graph. The edges between two components of the same object are implicitly defined by the definition of complex regions, whereas the rest of edges are defined by the bit vector that is validated. For each bit vector we construct several scenarios with different numbers of components, and for each scenario we iterate over all possible assignments of predicates matching the bit vector. This is repeated for each bit vector until it is either validated or a constraint can be proven that invalidates the vector.

As seen in Table 2, for complex regions we have identified the 17 constraints which eliminate all the invalid bit vectors. The final set  $LT_{cr}$  is composed of 137209 18-bit vectors.



**Fig. 6.** Two spatial configurations with the same topological bit vector but a different 9-intersection matrix

### 5.3 Comparing the Models

Although the  $LTBV_{cr}$  model of localized topological relationships effectively exposes the local interactions among components of complex regions, it is not yet clear how expressive the model is in comparison to the 9IM. We determine the expressiveness of the LTP model for complex regions with respect to the 9IM based on the following: if every spatial configuration that is representable and distinguishable by the 9IM can also be represented and distinguished by the LTP model, then the LTP model is at least as *globally* expressive as the 9IM. Otherwise, we say that the LTP model is *globally* less expressive than the 9IM.

It turns out that a number of spatial configurations are distinguishable by the 9-intersection model (i.e., they have different matrix representations) but not by the LTP model. For example, both scenes in Figure 6 are represented by different 9-intersection matrices but by the single bit vector  $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$ , i.e., all pairs of components from the respective objects *overlap*. Thus, we conclude that the LTP model is *globally* less expressive than the global 9IM.

Conversely, we compare the *local* expressiveness of the models to determine whether every spatial configuration that is representable and distinguishable by the LTP model can also be represented and distinguished by the 9IM. Figures 7(a), and 7(b) illustrate two spatial configurations which are distinguished by two different bit vectors  $[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$ , and  $[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$  respectively. However, both scenes are represented only by a single 9-intersection matrix. Thus, we conclude that the 9IM is *locally* less expressive than the LTP model.

Due to the differences in expressiveness of the LTP model and the 9IM at both the global and local levels we cannot draw any conclusion as to the relative general expressiveness of the models. This is due to the fact that the models are fundamentally incomparable. Our goal is to produce a hybrid model of topological predicates that is at least as globally expressive as the 9IM. We observe that the scenes in Figure 6 cannot be differentiated due to the fact that a global matrix can include global information that cannot be determined from examining purely local information. For example, the bit vector for the scenes in Figure 6 merely indicates that there exists a local *overlap* between a component of complex region  $A$  and a component of complex region  $B$ , whereas both scenes are distinguished by two different global matrices because of the difference in the existence of intersection between the boundary of  $A$  and the exterior of  $B$ . The non-existence of this intersection in Figure 6a occurs because another local *overlap* exists such that the boundary of  $A$  is now part of the closure of  $B$ . To determine such global information from a bit vector, more than simply local information must be included in the LTP model.

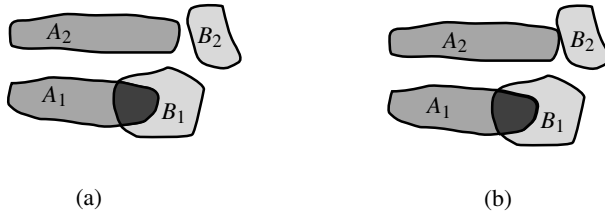


Fig. 7. Two spatial configurations with the same 9-intersection matrix but a different TBV

## 6 Maintaining Global Information

Since we have shown the incomparability of the expressiveness between the LTP and the 9IM, we must determine a different, hybrid view model that is at least as globally expressive as the 9IM model. As described in Section 5.3, some global information cannot be expressed by the LTP model, which keeps purely local information. In order to handle this issue, the necessary global information must also be represented in the model. In this section, we present a *hybrid topological predicate* (HTP) model that combines both local and global information into a single predicate. We refer to the set of all HTPs between complex regions as  $HT_{cr}$ . We begin by determining which global information must be included into the HTP model, and then define a new bit vector for HTPs and use it to derive a complete set of valid HTPs between complex regions. Finally, we compare the 9IM and the HTP model.

### 6.1 Hybrid Characterization of Topological Predicates

In order to determine which global information is needed in the HTP model, we examine the effects of local interactions on global interactions, and vice versa. To do this, we identify the collections of matrix entries in the global matrix that must be 1 (non-empty intersection) in order for a given local interaction to be possible. In other words, if an existence predicate yields *true*, we must define how it is expressed in the global matrix. For example, if a *meet* interaction occurs between any pair of components from two complex regions, then we know that in the global matrix representing the scene, the boundaries of the objects must intersect. In general, if any local interaction exists between two complex regions such that the interiors intersect, the boundaries intersect, or the interior intersects the boundary, then the corresponding matrix entry in the global 9-intersection matrix will be 1. This is because if any of these interactions occur locally, the existence of other local interactions cannot cause them to not be reflected at the global level. In contrast, any local interactions involving the exterior of either region may not be reflected in the global matrix because other local interactions can overwhelm them. For example, in Figure 6a, the interiors and boundaries of  $A_1$  and  $B_1$  intersect, and adding other local interactions cannot cause them to no longer intersect; thus, they will intersect in the global matrix. Locally,  $A_1$  and  $B_1$  *overlap*, thus their boundaries and interiors intersect with each other's exteriors. However, the inclusion of  $B_2$  in the scene causes the interaction involving the boundary of  $A_1$  with the exterior of  $B_1$  to not be reflected in the global matrix. This is because globally, the exterior of  $B_1$  is not considered independently, instead the exterior of  $B$  is considered as a whole.



We determine that the global information that is required to distinguish the scenes in Figure 6 from each other involves only the interactions of the exteriors of the objects. Based on [1], we know that the intersection between exteriors is always non-empty. Based on the observations detailed above, we must be able to represent the other four exterior interactions (i.e., intersections between interior and exterior, boundary and exterior, and their converses) in order to achieve a *one-to-many* relationship from the 9IM to the HTP model so that all cases that the 9IM can differentiate can also be uniquely identified by the HTP model.

To support the HTP model, we extend the topological bit vector into a new *hybridized topological bit vector* whose bit entries represent both local information and global information. The local information bits represent the eighteen ETPs between complex regions described earlier, whereas the global information bits represent the global exterior interactions. We denote the four meaningful global interactions involving the exterior of either object as *global interaction predicates* (GIPs) and define them as:  $ie = (A^+ \cap B^- \neq \emptyset)$ ,  $be = (\partial A \cap B^- \neq \emptyset)$ ,  $ei = (A^- \cap B^+ \neq \emptyset)$ , and  $eb = (A^- \cap \partial B \neq \emptyset)$ . GIPs are expressed by four global information bits in the hybridized topological bit vector. Thus, each element of  $HT_{cr}$  is represented by a 22-bit vector where the first 18 bits represent the local interactions, and the last four represent the GIP in the order we have just presented them. A bit in the vector is set to 1 if its corresponding ETP or GIP yields *true* and 0 otherwise. The set of all hybridized topological bit vectors is denoted  $HTBV_{cr}$ . Now, we can distinguish between the scenes in Figure 6: the bit vectors corresponding to Figure 6a and 6b are  $[0, 1, 0, 0, 1, 1]$ , and  $[0, 1, 1, 1, 1, 1]$  respectively.

## 6.2 Ensuring Expressive Power

The final step in the definition of hybridized topological predicates is the identification of the set  $HTBV_{cr}$  that represents the elements in  $HT_{cr}$ . We make the assertion that for the elements in  $HTBV_{cr}$ , the first 18 bits (local information) of all 22-bit vectors must match one of the elements in  $LTBV_{cr}$  identified following the method described in Section 5.2. This assertion is based on the fact that the information contained in those bits must be identical to an element in  $LTBV_{cr}$  or else it describes a topologically invalid scene. The addition of GIPs only reflects global information and thus does not affect the validity of the topological configuration described by the first 18 bits.

To identify the set of valid 22-bit vectors we provide a two stage method for determining the relationship between the set of 9-intersection matrices and the set of 22-bit vectors: (1) determine the set of 9-intersection matrices that correspond to each 18-bit configuration, and (2) create a 22-bit vector for each corresponding 18-bit vector and 9-intersection matrix by taking the four bits  $ie$ ,  $be$ ,  $ei$ , and  $eb$  of the 9-intersection matrix and extending the 18-bit vector into a 22-bit vector using these GIPs.

We achieve stage one using the matrix templates shown in Table 3. For each ETP we derive a set of template matrices so that each template shows how the existence of a local interaction can be expressed in a global 9-intersection matrix. The values of the templates are based on two observations: first, interactions between the exterior of one component of a complex region and the interior or boundary of a component of another complex region may not be expressed globally due to the existence of other local

**Table 3.** Templates for mapping 9IM to HTP. Hyphens indicate “*don’t care*” values.

$\begin{pmatrix} - & - & - \\ - & - & - \\ - & - & 1 \end{pmatrix}$ <i>disjoint<sub>e</sub></i>	$\begin{pmatrix} - & - & 1 \\ - & 1 & - \\ 1 & 1 & 1 \end{pmatrix}$ <i>fillingHole<sub>e</sub></i>	$\begin{pmatrix} - & - & 1 \\ - & 1 & - \\ 1 & - & 1 \end{pmatrix}$ <i>holeFilled<sub>e</sub></i>	$\begin{pmatrix} - & - & - \\ - & 1 & - \\ - & - & 1 \end{pmatrix}$ <i>meet<sub>e</sub></i>	$\begin{pmatrix} 1 & - & - \\ - & 1 & - \\ - & - & 1 \end{pmatrix}$ <i>equal<sub>e</sub></i>	$\begin{pmatrix} 1 & - & - \\ 1 & - & - \\ 1 & - & 1 \end{pmatrix}$ <i>inside<sub>e</sub></i>
$\begin{pmatrix} 1 & - & - \\ 1 & 1 & - \\ 1 & - & 1 \end{pmatrix}$ <i>holeCoveredBy<sub>e</sub></i>	$\begin{pmatrix} 1 & - & - \\ 1 & 1 & - \\ 1 & - & 1 \end{pmatrix}$ <i>coveredBy<sub>e</sub></i>	$\begin{pmatrix} 1 & 1 & 1 \\ - & - & - \\ - & - & 1 \end{pmatrix}$ <i>contains<sub>e</sub></i>	$\begin{pmatrix} 1 & 1 & 1 \\ - & 1 & - \\ - & - & 1 \end{pmatrix}$ <i>holeCovers<sub>e</sub></i>	$\begin{pmatrix} 1 & 1 & 1 \\ - & 1 & - \\ - & - & 1 \end{pmatrix}$ <i>covers<sub>e</sub></i>	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & - & - \\ 1 & - & 1 \end{pmatrix}$ <i>insideOverfill<sub>e</sub></i>
$\begin{pmatrix} 1 & 1 & 1 \\ 1 & - & - \\ 1 & - & 1 \end{pmatrix}$ <i>containsOverfill<sub>e</sub></i>	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & - & - \\ 1 & - & 1 \end{pmatrix}$ <i>insideContains<sub>e</sub></i>	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & - \\ 1 & - & 1 \end{pmatrix}$ <i>coversCoveredBy<sub>e</sub></i>	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & - \\ 1 & - & 1 \end{pmatrix}$ <i>coversOverfill<sub>e</sub></i>	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & - \\ 1 & - & 1 \end{pmatrix}$ <i>coveredByOverfill<sub>e</sub></i>	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & - \\ 1 & - & 1 \end{pmatrix}$ <i>overlap<sub>e</sub></i>

interactions. This argument is exposed in the previous section as the reason for the GIPs. As an example of this observation, in Figure 6a, the local interaction between the exterior of  $B_1$  and the boundary of  $A_1$  is not expressed globally because of the existence of  $B_2$ . Therefore, for each template all given values of 1 signify that the value must be set in the global matrix if the existence predicate bit is set in the bit vector. Values for which no global assumption can be made based on the local predicate are denoted in the template with a dash.

The second observation used to derive the templates is that the existence of a local interaction involving the interior or boundary of a component of one complex region and the interior or boundary of a component of another complex region asserts that the corresponding interaction will be reflected in the global matrix describing the scene. Thus, if an ETP is *true* for a scene and it involves one of these four interactions, then all of its corresponding matrix templates will also have a 1 in the corresponding matrix entry. The first observation leads us to denote template values that may *not* occur even if they occur in a local matrix from the same scene, whereas the second observation leads us to denote template values that must occur in the global matrix if they occur in any local matrix. Once the templates are computed, we compute all possible permutations of template matrices where dashes in the entries involving the exterior of a region are replaced with values of 1 such that the *non-empty-entry* set of the template is a subset of the *non-empty-entry* set of its corresponding 9-intersection matrix. These permutations show all possible ways a local interaction may be reflected in a global matrix.

Using the matrix templates, we construct a correspondence between global matrices and 18-bit configurations as follows. First, we consider the set  $K_m$  of all matrix templates whose *non-empty-entry* set is a subset of the *non-empty-entry* set of a given 9-intersection matrix  $m$  (the *non-empty-entry* set of a template matrix consists of all entries that are set to 1). Then we consider all elements of the powerset of  $K_m$  where their boolean sum<sup>1</sup> is equivalent<sup>2</sup> to  $m$ . For each of these elements, we create an 18-bit configuration in which a corresponding ETP has a value of 1 if any of its templates exist in the combination. If the resulting 18-bit configuration does not match one of the identified

<sup>1</sup> If an entry is equal to 1 in any matrix in the set, it is equal to 1 in the boolean sum of the matrices of that set.

<sup>2</sup> We consider two matrices to be equivalent if their *non-empty-entry* sets are equal.

topologically valid 18-bit vectors, it is discarded. The non-discarded configurations describe the same topological scene as the given 9-intersection matrix  $m$ .

In the second stage, we extend each 18-bit configuration to a 22-bit configuration by assigning the combination of GIPs from each 9-intersection matrix to the global information bits of the corresponding 22-bit configurations. This results in a set of 22-bit vectors that represent the elements in  $HTBV_{cr}$ . We know this set to be complete based on two arguments: first, the 18-bit configurations are proved to be complete based on the constraint and validation mechanism. Second, the values of the global information bits in a 22-bit vector are directly derived from the 9-intersection matrices that map to the corresponding 18-bit configuration. Any other combination of values for the global information bits would force a mapping into a global matrix that has been proven topologically invalid in Section 3.

Besides identifying the elements of  $HT_{cr}$ , the method described above also identifies a *one-to-many* relationship between a 9-intersection matrix and elements of  $HTBV_{cr}$ . For this reason we determine that the hybridized topological predicate model can distinguish more topological scenes than the 9IM, and is therefore more *globally* expressive than the 9IM while maintaining local information.

## 7 Querying with Local Information

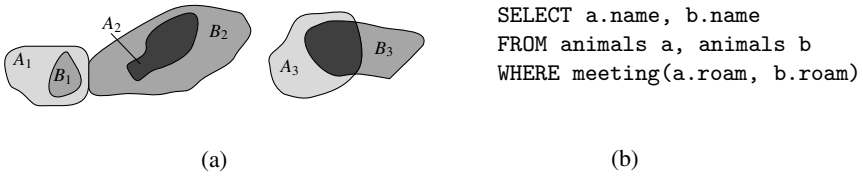
In this section, we introduce a proposal for making localized topological predicates available to a user intending to perform queries involving complex regions. We focus on leveraging features of a common database query language, SQL, to bring the LTP concept closer to the user. Section 7.1 introduces the notation that allows users to pre-define locality aware predicates that can be used in queries. In Section 7.2 we present sample queries that can make use of these predefined predicates but that can also be specified by embedding their definition within the query.

### 7.1 Constructing Locality Aware Topological Predicates

For the design of a syntactic construction for defining locality aware topological predicates, we focus on what the user may know and may want to express in her queries. The user needs not be aware of the underlying topological bit vector. Instead, the user simply wants to pose a query based on local topological relationships. For example, the user may want to retrieve all pairs of regions that have local situations of *disjoint* and *overlap* but where no *meet* situations occur. Thus, the user is specifying what *must* occur (*disjoint*, *overlap*), and what *must not* occur (*meet*). The rest of the local predicates

<pre>CREATE LTP ON &lt;type1&gt;, &lt;type2&gt; AS &lt;name&gt; (   [MUST ALL (&lt;predlist1&gt;)],   [MUST NOT ALL (&lt;predlist2&gt;)],   [MUST ONE (&lt;predlist3&gt;)],   [MUST NOT ONE (&lt;predlist4&gt;)]])</pre>	<pre>CREATE LTP ON region, region AS surrounded(   MUST ONE (equal,              inside, contains),   MUST NOT ALL (overlap))</pre>	<pre>CREATE LTP ON region, region AS meeting(   MUST ONE (holeFilled,             fillingHole, meet))</pre>
(a)	(b)	(c)

**Fig. 8.** SQL syntax and examples of DDL for predefining locality aware topological predicates



**Fig. 9.** Sample illustration and sample query using locality aware topological predicates

would be assessed as optional, meaning that they *may* (or may not) occur. In addition to specifying local existence predicates, the user should be able to specify any of the global interaction predicates that must, must not, and may occur. In implementation, the user can make use of the construct `CREATE LTP` which is defined as part of the query language's *data definition language* (DDL) and whose syntax is shown in Figure 8(a).

In the syntactic definition in Figure 8(a), `type1` and `type2` refer to the spatial data types upon which the predicates operate; these are currently only (complex) regions. The parameter name refers to a user defined label for the predicate that is being created. There are four optional clauses in the construct, divided into two groups: the ALL clauses and the ONE clauses. Each group has two options, the MUST and the MUST NOT. For the MUST clauses the user specifies a list of local predicates and GIPs that must appear to make the predefined predicate *true*. In the case of the MUST ALL clause, all local predicates and GIPs included in the clause must appear. For the MUST ONE clause, at least one of the predicates must appear. Similarly, the MUST NOT clauses allows the user to specify local predicates and GIPs that must not appear for the predicate to result in *true*. It also has two flavors: MUST NOT ALL, and MUST NOT ONE. All local predicates and GIPs that are not included in any clause are implicitly considered to be optional, thus they may or may not appear without changing the outcome of the predicate.

## 7.2 Local Topological Queries

Having defined a mechanism that allows the user to predefine locality aware topological predicates on complex regions, we move on to show two examples of how this predefined predicates can be used. We use Figure 9 to illustrate both sample queries. For the first example, assume that the elements in Figure 9(a) are complex regions that represent the roaming areas of two distinct species of animals. Although the areas of both species globally overlap, the user is interested in finding animals whose roaming areas somehow meet, as defined by the LTP *meeting* predefined in the previous section. Such a query would use the predefined predicate in the WHERE clause as shown in Figure 9(b).

From this example we notice how predefined LTPs can be used just as existing Boolean predicates. A system implementing LTPs should have system predefined predicates for all local topological predicates operating on the components of supported types. This means that the user may use locality aware existence predicates such as *fillingHole* and *containsOverfill* for a system supporting complex regions and their component simple regions with holes.

In lieu of predefining LTPs, users can also define them *inline* within the WHERE clause. The syntax definition of the sub-clause that is necessary for the inline definition is shown in Figure 10(a). In Figure 10(b) we show how this inline mechanism can be

<pre>WHERE LTP on &lt;attr1&gt;, &lt;attr2&gt; (   [MUST ALL (&lt;predlist1&gt;)],   [MUST NOT ALL (&lt;predlist2&gt;)],   [MUST ONE (&lt;predlist3&gt;)],   [MUST NOT ONE (&lt;predlist4&gt;)])</pre>	<pre>SELECT a.name, b.name FROM animals a, animals b WHERE a.avgsize&lt;50 AND b.avgsize&gt;50 AND b.diet=CARNIVORE AND LTP on a.roam, b.roam ( MUST ONE (fillingHole,             inside, coveredBy))</pre>
(a)	(b)

**Fig. 10.** Inline use of locality aware predicates within SQL queries: (a) syntax, (b) query

exploited to retrieve all animals and their predators whose average weight is less than 50 lbs. and whose roaming areas are surrounded by the roaming areas of larger carnivores.

## 8 Conclusion

In this paper, we have defined the concept of local topological predicates between complex regions. We have provided details for the identification of valid LTPs, and shown how these LTPs can be exploited by users through querying. The query language extensions described allow the users to define their own locality aware topological predicates that either can test for the existence of certain local interactions, or can be used to group LTPs together based their common attributes. The result of this work is a new model of topological predicates that is more expressive than the global 9IM and does not suffer from the dominance and composition problems.

Future work on this concept includes the actual implementation of LTPs within a spatial database and its application in real world scenarios. Another interesting topic of related research is the application of the LTP concept as part of indexing mechanisms for complex spatial objects.

## References

1. Schneider, M., Behr, T.: Topological Relationships between Complex Spatial Objects. *ACM Trans. on Database Systems* 31, 39–81 (2006)
2. Egenhofer, M., Clementini, E., Di Felice, P.: Topological Relations between Regions with Holes. *Int. Journal of Geographical Information Systems* 8, 128–142 (1994)
3. Clementini, E., Di Felice, P., Califano, G.: Composite Regions in Topological Queries. *Information Systems* 20, 579–594 (1995)
4. McKenney, M., Pauly, A., Praing, R., Schneider, M.: Preserving Local Topological Relationships. In: *ACM Symp. on Geographic Information Systems*, pp. 123–130. ACM Press, New York (2006)
5. Egenhofer, M.J.: Deriving the Composition of Binary Topological Relations. *Journal of Visual Languages and Computing* 2, 133–149 (1994)
6. Ladkin, P., Maddux, R.: On Binary Constraint Problems. *Journal of the Association for Computing Machinery* 41, 435–469 (1994)

# MOBIHIDE: A Mobile Peer-to-Peer System for Anonymous Location-Based Queries<sup>\*</sup>

Gabriel Ghinita<sup>1</sup>, Panos Kalnis<sup>1</sup>, and Spiros Skiadopoulos<sup>2</sup>

<sup>1</sup> Dept. of Computer Science  
National University of Singapore  
{ghinitag,kalnis}@comp.nus.edu.sg  
<sup>2</sup> Dept. of Comp. Science & Technology  
University of Peloponnese, Greece  
spiros@uop.gr

**Abstract.** Modern mobile phones and PDAs are equipped with positioning capabilities (e.g., GPS). Users can access public location-based services (e.g., Google Maps) and ask spatial queries. Although communication is encrypted, privacy and confidentiality remain major concerns, since the queries may disclose the location and identity of the user. Commonly, spatial  $\mathcal{K}$ -anonymity is employed to hide the query initiator among a group of  $\mathcal{K}$  users. However, existing work either fails to guarantee privacy, or exhibits unacceptably long response time.

In this paper we propose MOBIHIDE, a Peer-to-Peer system for anonymous location-based queries, which addresses these problems. MOBIHIDE employs the Hilbert space-filling curve to map the 2-D locations of mobile users to 1-D space. The transformed locations are indexed by a Chord-based distributed hash table, which is formed by the mobile devices. The resulting Peer-to-Peer system is used to anonymize a query by mapping it to a random group of  $\mathcal{K}$  users that are consecutive in the 1-D space. Compared to existing state-of-the-art, MOBIHIDE does not provide theoretical anonymity guarantees for skewed query distributions. Nevertheless, it achieves strong anonymity in practice, and it eliminates system hotspots. Our experimental evaluation shows that MOBIHIDE has good load balancing and fault tolerance properties, and is applicable to real-life scenarios with numerous mobile users.

## 1 Introduction

Consider the following scenario: Bob uses his GPS enabled mobile phone (e.g., iPAQ hw6515, Mio A701) to ask the query “Find the nearest AIDS clinic to my present location”. This query can be answered by a *Location-Based Service* (LBS), e.g., Google Maps, which is *not* trusted. To preserve his privacy, Bob does not contact the LBS directly. Instead he submits his query via a trusted

---

<sup>\*</sup> This work has been partially supported by project PENED 03 funded by the European Social Fund (75%) and the General Secretariat of Research and Technology (25%).

*pseudonym* service which hides his identity (services for anonymous web surfing are commonly available). Nevertheless, the query still contains the exact coordinates of Bob. One may reveal sensitive data by combining the location with other publicly available information. If, for instance, Bob uses his mobile phone within his residence, the untrustworthy owner of the LBS may infer Bob’s identity (e.g., through a white-pages service) and speculate that he suffers from AIDS. Bob may even hesitate to ask innocuous queries such as “Find the nearest restaurant”, in order to avoid unsolicited advertisement.

Recent research on LBS privacy focused on the  $\mathcal{K}$ -anonymity [17,20] technique, which is used in relational databases for publishing census, medical and other sensitive data. A dataset is  $\mathcal{K}$ -anonymous, if each record is indistinguishable from at least  $\mathcal{K}-1$  other records with respect to certain identifying attributes. In the LBS domain, a similar idea appears in Ref. [7,9,12,15], which employ *spatial cloaking* to conceal the location of the querying user  $u$ : Instead of reporting the coordinates of  $u$ , they construct an *Anonymizing Spatial Region* (ASR or  $\mathcal{K}$ -ASR) which encloses  $u$  and  $\mathcal{K}-1$  additional users. Typically, a central trusted server (called *location anonymizer*, or simply *anonymizer* in the sequel) exists between the users and the LBS. All users subscribe to the anonymizer and continuously update their position while they move. Each user sends his query to the anonymizer, which constructs the appropriate  $\mathcal{K}$ -ASR and contacts the LBS. The LBS computes the answer based on the  $\mathcal{K}$ -ASR, instead of the exact user location; thus, the response may contain false hits. The anonymizer filters the result and returns the exact answer to the user.

The centralized approach has several drawbacks; for example, the anonymizer may become bottleneck since it must handle frequent location updates as users move [8]. Most importantly, the centralized anonymizer poses a serious security threat. If it is compromised by an attacker, or forced to cooperate with a government agency, the history of all user movements and their queries may be revealed. For these reasons, two fully distributed systems emerged: (i) CLOAKP2P [5] is a Peer-to-Peer system which constructs  $\mathcal{K}$ -ASRs by considering users in the neighborhood of the querying user. (ii) PRIVÉ [8], on the other hand, clusters users in a hierarchical overlay network, resembling a distributed  $B^+$ -tree. Both systems minimize the security risk by distributing the sensitive information in numerous peers. However, we will show that CLOAKP2P fails to provide privacy for many user distributions, whereas PRIVÉ may suffer from slow response time, since root-level nodes constitute potential bottlenecks.

In this paper we propose MOBIHIDE, a Peer-to-Peer (P2P) system for anonymous location-based queries which addresses the problems of existing approaches. In MOBIHIDE the participating mobile devices form a hierarchical distributed hash table, based on the Chord P2P architecture [19], which indexes the locations of all users. In order to map the 2-D user locations to the 1-D Chord space, we employ the Hilbert space-filling curve [16].  $\mathcal{K}$ -ASRs are collaboratively assembled by peers in a distributed fashion, by choosing random groups of  $\mathcal{K}$  users (including the querying user) that are consecutive in the 1-D space. We prove that for uniform query distribution MOBIHIDE *guarantees* privacy,

and we show experimentally that even for skewed distributions, the probability of identifying the querying user is very close to the theoretical bound. A clear trade-off emerges between MOBIHIDE and existing state-of-the-art PRIVÉ: the latter provides anonymity guarantees under any query distribution, but has an hierarchical architecture. On the other hand, MOBIHIDE alleviates system hotspots, and still achieves strong anonymity in practice. Our experiments suggest that MOBIHIDE is resilient to failures, achieves good load balancing and supports efficiently the relocation of users (as users move) and the construction of  $\mathcal{K}$ -ASRs (while querying); therefore, it is scalable to a large number of mobile users.

The rest of the paper is organized as follows: Section 2 presents an overview of MOBIHIDE. Section 3 surveys the related work. Section 4 introduces our Hilbert-based randomized  $\mathcal{K}$ -ASR construction algorithm, whereas Section 5 describes the implementation of our system on top of Chord. Section 6 presents the experimental evaluation of MOBIHIDE. Finally, Section 7 concludes the paper and discusses directions for future work.

## 2 Overview of MobiHide

We assume a large number of users who carry mobile devices (e.g., mobile phones, PDAs) with embedded positioning capabilities (e.g., GPS). The devices have processing power and access the network through a wireless protocol such as WiFi, GPRS or 3G. Moreover, each device has an IP address and can establish point-to-point communication with any other device in the system through a base station (i.e., the two devices do not need to be within the range of each other). For security reasons, all communication links are encrypted. In addition, we assume the existence of a trusted central *Certification Server* (CS), where users are registered. Prior to entering the system, a user  $u$  must authenticate against the CS and obtain a certificate. Users having a certificate are trusted by all other users. Typically, a certificate is valid for several hours; it can be renewed by recontacting the CS. Apart from the certificate, the CS returns to  $u$  a list of possible entry points to the P2P network (i.e., IP addresses of on-line users). Note that the CS does not know the locations of the users and does not

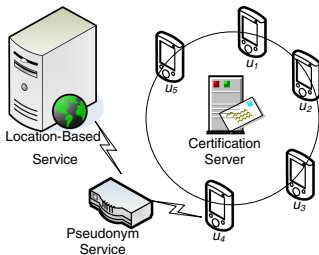


Fig. 1. System architecture

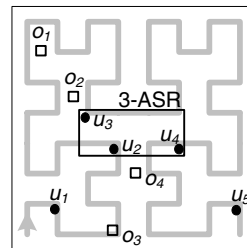


Fig. 2. Anonymized query,  $\mathcal{K}=3$



participate in the anonymization process; therefore, it is not a security threat or a bottleneck.

The mobile users self-organize into a P2P system (see Figure 1) based on the Chord [19] distributed hash table architecture, well-known for its good scalability and fault-tolerance properties. The P2P system defines a 1-D space of index keys; MOBIHIDE uses the Hilbert space-filling curve to map the 2-D user coordinates to 1-D space. The Hilbert curve is a continuous fractal (see Figure 2) which maps each region of the space to an integer. With high probability, if two points are close in the 2-D space, they will also be close in the Hilbert transformation [16].

Typically users ask Range or Nearest-Neighbor (NN) queries with respect to their location. In the example of Figure 2, user  $u_4$  asks for the nearest object to his location (i.e.,  $o_4$ ). Assume that the required degree of anonymity is  $\mathcal{K} = 3$  ( $\mathcal{K}$  may vary among users). MOBIHIDE identifies in a distributed manner a random set of 3 users (including  $u_4$ ) that are consecutive in the 1-D space (i.e.,  $u_2$ ,  $u_3$  and  $u_4$  in the example), and constructs the corresponding 3-ASR (i.e., the rectangle which encloses the 3 users). Next,  $u_4$  submits the 3-ASR NN query to the LBS through any existing pseudonym service [2]. Note that the pseudonym service hides the IP address of  $u_4$  but is not aware of the users' locations. Furthermore, it does not become a bottleneck, since each user may choose his preferred pseudonym service.

The LBS returns to  $u_4$  (through the pseudonym service) the NN of every point of the 3-ASR. Intuitively, the nearest neighbors of a region are all data objects inside the region plus the NNs of every point in the perimeter of the region [11]. In our example, these are objects  $o_2$  and  $o_4$ . Finally,  $u_4$  filters the false hits and determines his true NN (i.e.,  $o_4$ ). Note that the number of false hits depends on the  $\mathcal{K}$ -ASR; therefore we aim to minimize the size of the  $\mathcal{K}$ -ASR. Query processing at the LBS [11,12,15] is orthogonal to our work but outside the scope of this paper.

### 3 Background and Related Work

$\mathcal{K}$ -anonymity was first discussed in relational databases, where sensitive published data (e.g., census, medical) should not be linked to specific persons. Samarati and Sweeney [17,20] proposed the following definition: A relation satisfies  $\mathcal{K}$ -anonymity if every tuple is indistinguishable from at least  $\mathcal{K}-1$  other tuples with respect to every set of quasi-identifier attributes. Quasi-identifiers are sets of attributes (e.g., date of birth, gender, zip code) which can be linked to publicly available data to identify individuals. Machanava,jjhala et al. [14] proposed  $\ell$ -diversity, an anonymization method that extends  $\mathcal{K}$ -anonymity by providing diversity among the sensitive attribute values of the anonymized set.

Privacy in location-based services has recently attracted a lot of attention. Spatial  $\mathcal{K}$ -anonymity is defined as [8]:

**Definition 1 (Spatial  $\mathcal{K}$ -anonymity).** *Let  $A$  be a set of  $\mathcal{K}$  users with locations enclosed in an arbitrary spatial region  $\mathcal{K}$ -ASR. User  $u \in A$  is said to possess  $\mathcal{K}$ -*

*anonymity, if the probability of distinguishing  $u$  among the other users in  $A$  does not exceed  $1/\mathcal{K}$ , where  $\mathcal{K}$  is the required degree of anonymity.*

Note that: (i) The definition assumes a *snapshot* of the users' locations. Although we support user mobility,  $\mathcal{K}$ -anonymity is undefined across multiple snapshots. (ii) Spatial  $\mathcal{K}$ -anonymity does *not* depend on the size of the  $\mathcal{K}$ -ASR. In the extreme case, the  $\mathcal{K}$ -ASR can degenerate to a point, if  $\mathcal{K}$  users are at the same location. In general, we prefer small  $\mathcal{K}$ -ASRs, in order to minimize the processing cost at the LBS and the communication cost between the LBS and the mobile user. Nevertheless, some applications impose a lower bound on the size of the  $\mathcal{K}$ -ASR [15]. In such cases, the  $\mathcal{K}$ -ASR can be trivially scaled to satisfy the lower bound. The same procedure can also be used to avoid having users on the perimeter of the  $\mathcal{K}$ -ASR.

Also observe that the naïve solution of generating an arbitrary  $\mathcal{K}$ -ASR around the querying user, is not applicable. If, for instance, the user resides in a rural area, the  $\mathcal{K}$ -ASR may include only himself, whereas in a densely populated area, a too large  $\mathcal{K}$ -ASR will affect the query processing cost. Moreover, we cannot select  $\mathcal{K}-1$  random users and send  $\mathcal{K}$  distinct queries, because this would reveal the *exact* locations of  $\mathcal{K}$  users; this is not desirable for any anonymization technique.

Ref. [7] considers mobile users who send queries to the anonymizer together with a spatial cloaking range  $\delta_x, \delta_y$  and a temporal cloaking interval  $\delta_t$ . If  $\mathcal{K}-1$  other users generate queries within this cloaking box, the query is issued, otherwise it is dropped. Ref. [9], on the other hand, assumes that users report periodically their location to the anonymizer, and focuses on concealing the exact location without considering query processing. The anonymizer indexes the locations of all users with a Quad-tree [18]. For a user  $u$ , it traverses the Quad-tree until it encounters a quadrant which includes  $u$  and less than  $\mathcal{K}-1$  additional users. Then it selects the parent of that quadrant as  $\mathcal{K}$ -ASR. Casper [15] also employs a variation of the Quad-tree anonymization method. Casper attempts to build  $\mathcal{K}$ -ASRs by combining two neighboring quadrants, rather than going one level up in the tree every time more users are required. Finally, Ref. [12] introduces the HILBASR algorithm based on space-filling curves, and proposes an integrated framework for  $\mathcal{K}$ -ASR construction and query processing in LBS.

The previous approaches assume a centralized anonymizer. Recall from Section 1 that centralized approaches, among other drawbacks, are potential security threats. Closer to our approach is CLOAKP2P [5], which addresses the drawbacks of centralized anonymization by employing a fully distributed mobile Peer-to-Peer (P2P) system. In CLOAKP2P, the querying user  $u$  initiates  $\mathcal{K}$ -ASR construction by contacting all peers within a given physical radius  $r$ , which is a fixed system parameter. If the set of peers  $S_0$  found in the initial iteration is larger than  $\mathcal{K}$ , the closest  $\mathcal{K}$  of them are chosen to form the  $\mathcal{K}$ -ASR; otherwise, the process continues recursively, and all peers in  $S_0$  issue a request to all peers within radius  $r$ . Intuitively, CLOAKP2P determines a  $\mathcal{K}$ -ASR by finding the  $\mathcal{K}-1$  users closest to  $u$ . Unfortunately, this heuristic fails to achieve anonymity in many cases, since  $u$  tends to be closest to the center of the  $\mathcal{K}$ -ASR. We call this “*center-of- $\mathcal{K}$ -ASR*” attack. In Section 6 we demonstrate that, in many cases, an

attacker can identify  $u$  with probability much higher than  $1/\mathcal{K}$ . The experiments show that MOBIHIDE is considerably more secure compared to CLOAKP2P.

PRIVÉ [8] is another P2P system, which uses the Hilbert transformation to generate a sorted 1-D sequence of all users. PRIVÉ constructs *fixed* partitions of  $\mathcal{K}$  users each (except the last one, which may have up to  $2\mathcal{K}-1$  users). It is formally proved that this method *guarantees* anonymity (i.e., prevents identification of the query source) against any location-based attack, for any distribution of users and queries, even if an attacker knows the exact location of users. To generate fixed partitions, PRIVÉ must determine the absolute rank of each user in the sorted Hilbert sequence. To achieve this, it implements an overlay network which resembles a distributed  $B^+$ -tree. For each query the search must start at the root of the tree; this can overload the root peer. Although PRIVÉ has a load-balancing mechanism, its purpose is to equally share load among users during long periods of time, but it cannot avoid the root hotspot when the number of users or the query rate increases. In Section 6 we will show that even with 10,000 users and a moderate query rate, the response time is almost 10 minutes, while as many as 60% of the queries are rejected due to buffer overflows. In contrast, MOBIHIDE does not maintain fixed partitions, therefore it is much faster than PRIVÉ.

The privacy of user locations has also been studied in the context of related problems. Probabilistic Cloaking [4] does not apply spatial  $\mathcal{K}$ -anonymity. Instead, given an ASR, the LBS returns the probability that each candidate result satisfies the query, based on its location with respect to the ASR. Kamat et al. [13] focus on sensor networks and examine the privacy characteristics of different routing protocols. Hoh and Gruteser [10] describe techniques for hiding the trajectory of users in applications that continuously collect location samples.

MOBIHIDE is built on top of Chord [19], a Distributed Hash Table (DHT) protocol that supports scalable, fully decentralized key searching. Chord has a flat structure, where all peers have equal responsibilities and equally share the system load among themselves. Our work is also related to CANON [6], which is a framework for building hierarchical DHTs, while retaining the homogeneity of load and functionality offered by flat designs.

## 4 The MobiHide Spatial Anonymization Algorithm

We introduce MOBIHIDE, a P2P system which employs a randomized  $\mathcal{K}$ -ASR construction technique to offer query source anonymity, and is scalable to a large number of mobile users. Similar to PRIVÉ, MOBIHIDE is using the Hilbert ordering of the users' locations. However, instead of grouping users into fixed partitions, it forms a  $\mathcal{K}$ -ASR by randomly choosing  $\mathcal{K}$  consecutive users, including the querying user.

Let  $[u_1, \dots, u_N]$  be the sequence of all users, ordered by their Hilbert value. To allow random  $\mathcal{K}$ -ASR selection for the users at the start and end of the sequence, the 1-D space becomes a ring (or torus), instead of an array. Therefore,  $u_1$  is after  $u_N$  (and  $u_N$  is before  $u_1$ ). Figure 3 presents an example, where  $u_q$  is the user who issues a query. There are  $\mathcal{K}$  ways to choose a set of consecutive  $\mathcal{K}$

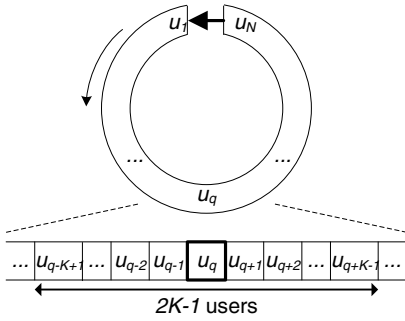


Fig. 3. Hilbert sequence ring

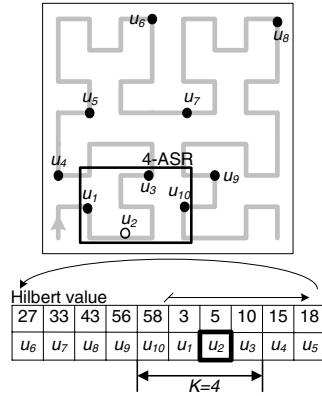


Fig. 4.  $\mathcal{K}$ -ASR construction in MOBIHIDE

users which includes  $u_q: [u_{q-\mathcal{K}+1} : u_q], [u_{q-\mathcal{K}+2} : u_{q+1}], \dots, [u_q : u_{q+\mathcal{K}-1}]$ . This is equivalent to choosing a random offset  $l \in [0, \mathcal{K}-1]$ , representing the offset of  $u_q$  in the resulting sequence. For example, if  $l = 0$ , the resulting sequence is  $[u_q : u_{q+\mathcal{K}-1}]$ . Observe that we only need information in the neighborhood of  $u_q$  in order to select the sequence (as opposed to PRIVÉ, which needs the global ranking). Therefore, MOBIHIDE works in a fully decentralized manner, and can be deployed on top of a scalable structure such as Chord.

Figure 4 shows an example of  $\mathcal{K}$ -ASR construction, where  $u_2$  is the querying user. Let  $\mathcal{K} = 4$  and assume that  $u_2$  randomly selects offset  $l = 2$ . According to the Hilbert ordering, the resulting sequence of users is  $[u_{10}, u_1, u_2, u_3]$ . The corresponding  $\mathcal{K}$ -ASR is the *minimum bounding rectangle* (MBR) which encloses these four users. In this particular example it was necessary to wrap around the Hilbert sequence (from  $u_{10}$  to  $u_1$ ). Observe that the “jump” in Euclidean distance due to wrapping, is not necessarily larger than other “jumps” that may occur within the sequence (e.g., from user  $u_8$  to  $u_9$ ). Therefore, the average size of the  $\mathcal{K}$ -ASRs (thus the query cost) is not affected significantly by wrapping. We investigate further this issue in Section 6.

**Theorem 1.** *If all users issue queries with the same probability (i.e., uniform distribution), MOBIHIDE guarantees query anonymity.*

*Proof.* Denote by  $P_Q$  the probability of a user issuing a query (same for all users). The query source generates a random offset  $l \in [0, \mathcal{K}-1]$ ; we denote by  $\langle u, l \rangle$  the event of user  $u$  generating a set of users with offset  $l$ . The probability  $P_{\langle u, l \rangle} = P_Q / \mathcal{K}$ . Refer to Figure 3, where  $u_q$  is issuing a query. Obviously,  $u_q$  must belong to the set associated to his query. To guarantee anonymity, the probability of identifying  $u_q$  as the query source must not exceed  $1/\mathcal{K}$ . We denote by  $A_q$  any set of users that includes  $u_q$ , and by  $P_{A_q}$  the probability of such a set being generated. We denote by  $P_{u_i}$  the probability of user  $u_i$  being the source of the query associated with  $A_q$ . Then,  $P_{u_i} > 0$  only for users  $[u_{q-\mathcal{K}+1} : u_{q+\mathcal{K}-1}]$ , and by symmetry,  $P_{u_{q-j}} = P_{u_{q+j}}$ . We have:

$$\begin{aligned}
 P_{u_q} &= \sum_{l=0}^{\mathcal{K}-1} P_{\langle u_q, l \rangle} = P_Q, & P_{u_i} &= \sum_{l=i-q}^{\mathcal{K}-1} P_{\langle u_i, l \rangle} = \frac{\mathcal{K} - i + q}{\mathcal{K}} P_Q, \quad i > q \\
 & & P_{u_q} + 2 \sum_{i=q+1}^{q+\mathcal{K}-1} P_{u_i} &= P_{A_q}
 \end{aligned}$$

The probability of pinpointing  $u_q$  as the query source is

$$\frac{P_{u_q}}{P_{A_q}} = \frac{P_Q}{\left(1 + 2 \sum_{i=1}^{\mathcal{K}-1} \frac{\mathcal{K} - i}{\mathcal{K}}\right) P_Q} = \frac{1}{\mathcal{K}}, \tag{1}$$

hence user  $u_q$  is  $\mathcal{K}$ -anonymous.

### 4.1 The Correlation Attack

In practice, the query distribution is not always uniform, hence Theorem 1 may not hold. In the extreme case, the same user (e.g.,  $u_q$ ) would send all queries and he would be included in all  $\mathcal{K}$ -ASRs. An attacker can intersect the  $\mathcal{K}$ -ASRs and pinpoint  $u_q$  as the querying user with high probability. It is more realistic, however, that many users ask queries, even if the query distribution is skewed. In this case, intersecting the  $\mathcal{K}$ -ASRs is unlikely to compromise the system, since the random sequence selection in MOBIHIDE distributes the anonymized regions in the entire space. In order to succeed, the attacker should know the *exact locations of all users*, to be able to reconstruct the Hilbert sequence. Then, he could find the users included in each  $\mathcal{K}$ -ASR by reverse-engineering the  $\mathcal{K}$ -ASR construction mechanism, and speculate that the users who appear more frequently are the ones who issued the queries.

Consider the extreme case where the attacker knows the exact location of all users and intercepts the set  $\mathcal{R}$  of  $\mathcal{K}$ -ASRs. We formalize the correlation attack as follows: (i) Construct a histogram  $F$  with the number of occurrences of every user in any of the queries. (ii) For each  $R \in \mathcal{R}$ : infer the query source as the user in  $R$  with the highest number of occurrences in  $F$ .

The correlation attack gives an attacker powerful means to infer the query source. PRIVÉ guarantees anonymization against this type of attack, but as discussed in Section 3, scales poorly as the number of users increases. MOBIHIDE cannot offer theoretical guarantees when the query distribution is extremely skewed. However, we believe that in practice this attack is hard to stage, since it is difficult for an attacker to know the exact locations of all users at each snapshot. Furthermore, we show experimentally (Section 6) that the probability of identifying the querying user in MOBIHIDE is very close to the theoretical bound  $1/\mathcal{K}$ , even if the attacker knows all users’ locations and the query distribution is skewed. Finally, observe that MOBIHIDE does not suffer from the “center-of- $\mathcal{K}$ -ASR” attack (see Section 3) because, by construction, the probability of  $u_q$  to be closest to the center of the  $\mathcal{K}$ -ASR is  $1/\mathcal{K}$ .

## 5 Implementation of MobiHide

MOBIHIDE users organize themselves into a Chord [19] P2P system. Chord is a Distributed Hash Table (DHT), where each peer (or node) has an  $m$ -bit key (the Hilbert value in our case), and it stores a routing table with pointers to other nodes (see Figure 5). The routing table at peer  $n$  with key  $key_n$  consists of:

- a *successor* and *predecessor* pointer to the node with the key that immediately follows (respectively, precedes)  $key_n$  on the ring
- a *successor list*, used mainly for fault tolerance, with a list of consecutive peers that follow  $n$  on the ring
- a *finger table*, with  $m$  pointers to nodes that are situated at  $2^i$  distances from  $n$  ( $i = 0, 1, \dots, m - 1$ ).

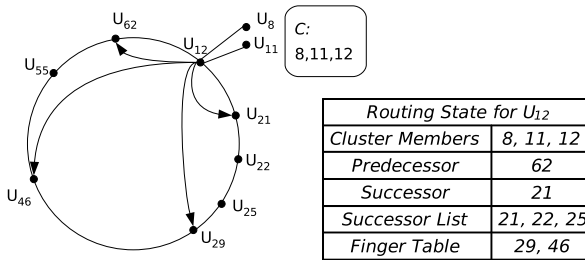


Fig. 5. MOBIHIDE implementation over Chord

We denote by  $\mathcal{H}(u)$  the Chord key of user  $u$ . Assume that each user is mapped to a distinct Chord node. When user  $u$  wants to ask a query, he initiates the  $\mathcal{K}$ -ASR construction procedure, denoted by  $\mathcal{K}$ -request.  $u$  generates a random offset  $l \in [0, \mathcal{K}-1]$ , and contacts the set  $P$  of  $l$  predecessors and the set  $S$  of  $\mathcal{K}-1-l$  successors on the Chord ring. The resulting  $\mathcal{K}$ -ASR is the MBR that encloses users in  $P \cup S \cup \{u\}$ . The complexity of a  $\mathcal{K}$ -request is  $O(\mathcal{K})$  overlay hops.

Since  $\mathcal{K}$  can be large (e.g., 50-100) in practice, we wish to reduce the number of hops, and hence the latency of  $\mathcal{K}$ -request. We introduce an additional level of hierarchy, such that each overlay node represents a *cluster of users*, rather than a single user. Each cluster has between  $\alpha$  and  $3\alpha-1$  users, where  $\alpha$  is a system parameter. If the cluster reaches  $3\alpha$ , a split is performed and an additional ring node is created. If the size falls below  $\alpha$ , a merge operation with another overlay node is performed<sup>1</sup>. We chose  $3\alpha$ , instead of  $2\alpha$ , as the upper bound on size, to minimize frequent merge and split operations. Each cluster has a representative, or cluster *head*, which is part of the Chord ring. In the example of Figure 5,  $u_{12}$  is the head of cluster  $\{u_8, u_{11}, u_{12}\}$ . The head's key on the ring is the maximum of all keys inside its cluster, in order to preserve the key ordering on the ring. The cluster membership is maintained by the head, and is replicated to all cluster

<sup>1</sup> Obviously, if more keys fall within a Chord segment, there will also be proportionally more nodes in that segment; therefore, hot-spots are avoided.

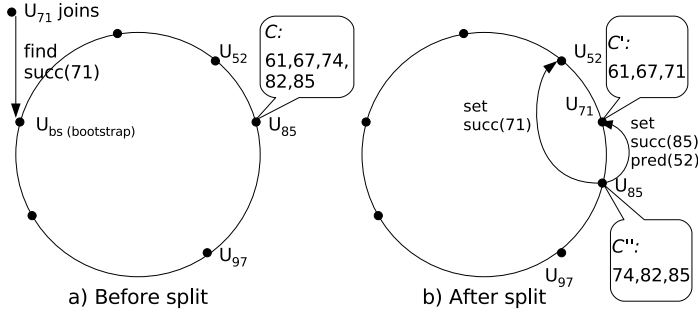


Fig. 6. Join and Split,  $\alpha=2$

members, to enhance fault-tolerance. Heads are rotated periodically to achieve load-balancing. We denote by  $C_u$  the cluster that contains user  $u$ , and by  $CH_u$  the head of  $C_u$ .

We further describe how various operations are performed in MOBIHIDE. For each operation, we consider two performance metrics:

- *latency*: the time to completion, measured as the number of overlay hops on the longest path followed. Multiple paths may be followed in parallel.
- *cost*: the communication cost of an operation, measured as the number of transmitted messages (communication cost typically prevails over CPU cost).

**Join and Departure.** User join is illustrated in Figure 6a. User  $u$  with key  $\mathcal{H}(u) = 71$  authenticates at the certification server and receives the address of some user  $u_{bs}$  inside the system.  $u_{bs}$  issues a search for key  $\mathcal{H}(u)$ , which returns the address of  $u_{85}$ , the successor of 71 on the ring.  $u$  contacts  $u_{85}$  and joins cluster  $C$ . Hence,  $C_u \equiv C$  and  $CH_u \equiv u_{85}$ . Upon  $u$ 's join,  $CH_u$  checks the new size of cluster  $C_u$ , and if  $size(C_u) = 3\alpha$ ,  $CH_u$  splits his cluster into two halves, in increasing order of key values. He appoints one of his cluster members,  $CH'_u$ , as head of the newly formed cluster. All nodes in the initial cluster are notified.  $CH_u$  and  $CH'_u$  also notify their predecessor and successor on the ring.  $CH'_u$  inherits a large part (if not all) of the finger table of  $CH_u$ ; the rest of the table is determined through the Chord stabilization process [19].

In our example, the new size of  $C$  is 6 and  $\alpha = 2$ , so  $u_{85}$  triggers a split operation (Figure 6b).  $u_{85}$  divides his cluster  $C$  into two halves,  $C'$  with members 61, 67 and 71, and  $C''$  with members 74, 82 and 85.  $u_{71}$  is appointed as head of  $C'$ , while  $u_{85}$  remains head for  $C''$ .  $u_{85}$  sets his predecessor pointer to  $u_{71}$ , and notifies the former predecessor  $u_{52}$  to change its successor from  $u_{85}$  to  $u_{71}$ . The complexity of join is  $O(\log N - \log \alpha)$  latency and  $O(\log N - \log \alpha + \alpha)$  communication cost (the last term stands for notifying all cluster members).

User  $u$  can depart gracefully, or fail; failure is addressed in Section 5.1. When  $u$  departs gracefully, he notifies his cluster head  $CH_u$ , who updates the cluster membership. If the departing node is cluster head, he appoints one of his members as new head. A merge can be triggered by departure. In this case, user  $CH_u$ s

---

<p><math>u</math>.findASR(<math>\mathcal{H}, \mathcal{K}</math>)</p> <p>  compute <math>rank_{\mathcal{H}}</math> in sorted order of <math>C_u</math></p> <p>  generate random offset <math>l</math></p> <p>  <math>before = \max(0, l - rank_H)</math></p> <p>  <math>after = \max(0, \mathcal{K} - l + rank_H - size(C_u))</math></p> <p>  <b>if</b> (<math>after &gt; 0</math>)</p> <p>    succ.FwdReq(<math>after, 1</math>)</p> <p>  <b>if</b> (<math>before &gt; 0</math>)</p> <p>    pred.FwdReq(<math>before, -1</math>)</p> <p>  wait for partial MBRs</p> <p>  <math>\mathcal{K}</math>-ASR = union of all received MBR</p>	<p><math>u</math>.<math>\mathcal{K}</math>-request(<math>\mathcal{K}</math>)</p> <p>  call <math>CH_u</math>.findASR(<math>\mathcal{H}(u), \mathcal{K}</math>)</p> <p><math>u</math>.FwdReq(<math>count, direction</math>)</p> <p>  <b>if</b> (<math>direction == 1</math>) /*Look Forward*/</p> <p>    return MBR of first <math>count</math> keys</p> <p>    <b>if</b> (<math>count &gt; size(C_u)</math>)</p> <p>      succ.FwdReq(<math>count - size(C_u), 1</math>)</p> <p>  <b>else</b> /*Look Backward*/</p> <p>    return MBR of last <math>count</math> keys</p> <p>    <b>if</b> (<math>count &gt; size(C_u)</math>)</p> <p>      pred.FwdReq(<math>count - size(C_u), -1</math>)</p>
---	---

---

**Fig. 7.** Pseudocode for  $\mathcal{K}$ -Request

triggering the merge contacts randomly either his successor  $s$  or predecessor  $p$  on the Chord ring to merge<sup>2</sup>.  $CH_u$  transfers his members (including himself) to the merging peer and ceases to be cluster head. All members are notified and the successor and predecessor pointers are updated.

**Relocation.** When user  $u$  moves to a new location, his Hilbert value  $\mathcal{H}(u)$  changes. If the new  $\mathcal{H}'(u)$  falls within the key range of other users in cluster  $C_u$ ,  $u$  only needs to inform his cluster head of the key change. Otherwise,  $u$  performs a graceful departure followed by a join. Since Hilbert ordering preserves locality, it is likely that the relocation will be within a small distance from the initial ring position. The worst case complexity of relocation is  $O(\log N - \log \alpha)$  latency and  $O(\log N - \log \alpha + \alpha)$  communication cost.

**$\mathcal{K}$ -request.** To generate a  $\mathcal{K}$ -ASR,  $u$  forwards a  $\mathcal{K}$ -request to his cluster head  $CH_u$  (unless  $u$  himself is the cluster head).  $CH_u$  generates a random offset  $l \in [0, \mathcal{K} - 1]$ . Then,  $CH_u$  examines the membership list of his cluster  $C_u$  and determines how many users in  $C_u$  will belong to the  $\mathcal{K}$ -ASR.  $CH_u$  computes the values  $before$  and  $after$  corresponding to the number of users in  $\mathcal{K}$ -ASR that are *outside*  $C_u$  and precede (respectively, follow) the set of keys in  $C_u$ .  $CH_u$  issues a request for the MBR<sup>3</sup> of these members to his predecessor  $p$  and successor  $s$ . In  $p$  and  $s$  the same procedure is followed recursively, until  $\mathcal{K}$  users are found.  $CH_u$  waits for all answers, and assembles the  $\mathcal{K}$ -ASR as the union of the received MBRs. The pseudocode<sup>4</sup> for  $\mathcal{K}$ -request is given in Figure 7. The complexity is  $O(\mathcal{K}/\alpha)$  in terms of both latency and communication cost. Once the  $\mathcal{K}$ -ASR is assembled,  $u$  can submit it to the LBS using his preferred pseudonym service.

<sup>2</sup> Alternatively, an interrogation phase can find which of  $s$  or  $p$  has fewer members, and merge with that one (to avoid cascaded splits and to equalize cluster sizes).

<sup>3</sup>  $CH_u$  only acquires the MBR, not the exact location of users in other clusters.

<sup>4</sup> We use the Remote Procedure Call convention  $u.routine()$ ; i.e.,  $u$  is the node where *routine* is executed.



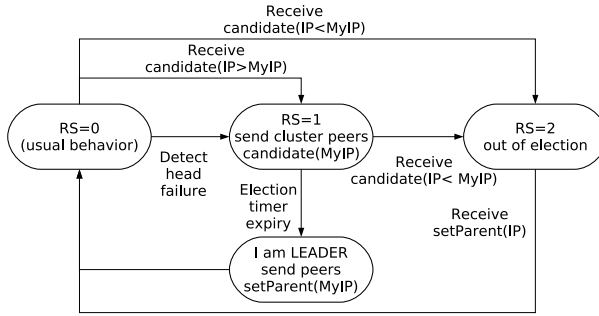


Fig. 8. Leader Election Protocol

### 5.1 Fault-Tolerance and Load Balancing

MOBIHIDE inherits the good fault-tolerance properties of Chord [19]. Similar to Chord, some of the pointers to other peers (i.e., successor and predecessor pointers, the successor list and the finger table) may be temporarily corrupted (e.g., when a user fails). Such pointers are corrected periodically through a stabilization process. In addition to stabilization, MOBIHIDE implements an intra-cluster maintenance mechanism. Each cluster head periodically (i.e., every  $\delta t$  seconds) checks if all cluster members are alive, by sending *beacon* messages; beacons contain the current cluster membership in addition to the successor and predecessor nodes of the head on the Chord ring. If a user fails to respond for  $2\delta t$  seconds, he is considered failed and is removed from the cluster. Similarly, a non-head node that does not receive a beacon from his head for  $2\delta t$  seconds, concludes that the head has failed and initiates a leader election protocol (see Figure 8). The RecoveryState ( $RS$ ) variable of each node indicates whether the node is in normal operation ( $RS = 0$ ) or participates in the election protocol. Since the cluster membership is replicated at all cluster nodes, recovery is facilitated. Upon detecting leader failure, node  $n$  enters the  $RS = 1$  state, sends a *candidate*( $n.IP$ ) message to all peers in the cluster and sets an election timer large enough to allow other peers to respond to the candidature proposal. When a node receives the *candidate*( $IP$ ) message, it initiates its own candidature only if its address is smaller than  $IP$ ; otherwise, it enters the  $RS = 2$  state and waits for a *setParent* message. The user with the smallest address declares himself leader and notifies all other cluster members, as well as the predecessor and successor on the ring.

To prevent unequal load sharing, a simple rotation mechanism is enforced among cluster members. The rotation is triggered when a certain load threshold is reached. This threshold is measured in terms of number of messages sent/received, since the communication cost is predominant in terms of both energy consumption and fees paid to the service provider. When the cluster head  $CH$  transfers leadership to another cluster member  $CH'$ , he transfers his routing state on the Chord ring and the cluster membership to  $CH'$ . Observe that the Chord key does not change, since it is the maximum key among all cluster members. Therefore, the overhead for the P2P network is minimal.

## 6 Experimental Evaluation

We implemented MOBIHIDE on top of Chord in the p2psim [1] suite, a packet-level simulator for P2P systems. We consider topologies with 1sec average round-trip delay, a typical value for wireless devices. To highlight the behavior of our system, we only consider packet loss as an effect of queueing at the processing nodes, and not as a result of link faults. Our dataset corresponds to the San Francisco Bay Area (Figure 9) and is constructed with the *Network-based Generator of Moving Objects* [3], which models the movement of mobile users on public road infrastructures. We consider scenarios with 1,000 to 10,000 users and anonymization degree  $\mathcal{K}$  between 10 and 160. If not stated differently, we set  $\alpha = 5$  (see Section 5). We compare MOBIHIDE against the two existing distributed spatial anonymization systems (i.e., CLOAKP2P and PRIVÉ).

**Anonymization Strength.** Theorem 1 theoretical demonstrates that MOBIHIDE guarantees  $\mathcal{K}$ -anonymity for a uniform query distribution. To complete our study, we also evaluate the anonymity strength of MOBIHIDE for skewed query distributions. To this end, we assume 10K users and 10K queries and consider a Zipfian query distributions with  $\vartheta = 0.8$ . We start by focusing on the “*center-of- $\mathcal{K}$ -ASR*” attack. We revisit the query scenario from Ref [8], and present a comparison of MOBIHIDE against PRIVÉ and CLOAKP2P. Let us denote by  $u_c$  the closest user to the center of the  $\mathcal{K}$ -ASR. Figure 10 illustrates the probability of  $u_c$  being the query source (i.e., the user initiating the query). Theoretically, to achieve anonymity, the above probability should be bounded by  $1/\mathcal{K}$ . In other words, the performance of the evaluated algorithms should be *under* line  $1/\mathcal{K}$  (the dotted line of Figure 10). CLOAKP2P, for  $\mathcal{K} \geq 20$ , does not satisfy the theoretical bound. For instance, for  $\mathcal{K}=40$ , the probability of  $u_c$  being the query source is 10%, i.e., four times the maximum allowed bound ( $1/\mathcal{K}=2.5\%$ ). The users are likely to come uniformly from all directions; hence,  $u_c$  is disclosed as the query source. Contrary, PRIVÉ and MOBIHIDE always satisfy the theoretical bound. Notice that in some cases, the MBR of the  $\mathcal{K}$ -ASR may contain a few more than  $\mathcal{K}$  users. This is why the results for PRIVÉ and MOBIHIDE are not identical to the  $1/\mathcal{K}$  line.



San Francisco Bay Area

Fig. 9. Dataset

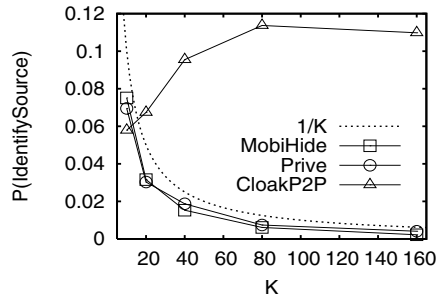


Fig. 10. “center-of- $\mathcal{K}$ -ASR” attack

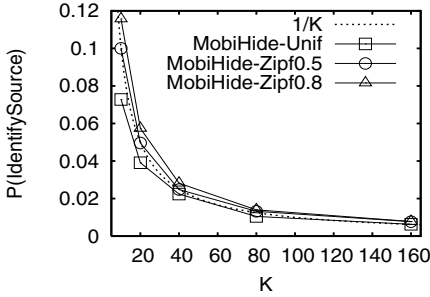
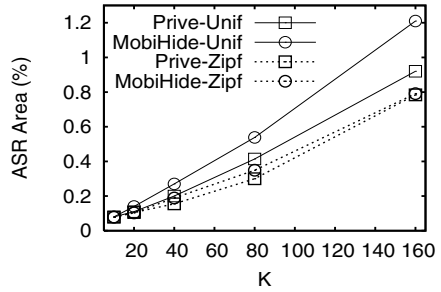


Fig. 11. Correlation attack (MOBIHIDE)

Fig. 12.  $\mathcal{K}$ -ASR Area

In Figure 11 we consider the correlation attack (see Section 4.1). We assume the extreme case, where the attacker knows the exact locations of all users (recall that this attack is unlikely to occur in practice). We show the results for uniform and Zipf query distribution, with  $\vartheta = 0.5$  and  $\vartheta = 0.8$ . As expected, for uniform distribution anonymity is always preserved. Actually, in this case MOBIHIDE behaves almost identical to PRIVÉ (not shown in the graph). Anonymity is also entirely preserved for  $\vartheta = 0.5$ . As the distribution becomes more skewed, MOBIHIDE may fail to preserve anonymity by a small margin. In most cases, however, the probability of identifying the query source is very close to the theoretical bound  $1/\mathcal{K}$ . In the worst case, for  $\mathcal{K} = 160$ ,  $\vartheta = 0.8$ , the probability of identifying the query source was  $1.2/\mathcal{K}$ . Observe that in Figure 11 we did not consider CLOAKP2P, as it can be easily compromised by the much simpler “center-of- $\mathcal{K}$ -ASR” attack. Since it fails to provide anonymity in many cases, we will not consider CLOAKP2P any further.

**$\mathcal{K}$ -ASR Size.** MOBIHIDE wraps around the Hilbert sequence in order to handle users near the start/end of the sequence. In some cases, this may yield  $\mathcal{K}$ -ASRs with larger area, compared to PRIVÉ; consequently, the query processing cost will increase. To investigate this issue, we considered uniform and Zipf ( $\vartheta = 0.8$ ) query distributions over a set of 10K users and varying  $\mathcal{K}$ . In Figure 12 we plot the average area of the  $\mathcal{K}$ -ASRs as a percentage of the entire dataspace. Observe that for the Zipf distribution the two systems behave almost identical, while for uniform distribution MOBIHIDE generates 25% larger  $\mathcal{K}$ -ASRs in the worst case. Therefore, we tradeoff at most 25% in additional query processing cost, but we obtain far superior system scalability as we will show next.

**Scalability (response time).** The most important advantage of MOBIHIDE is its increased scalability due to the highly decentralized structure. Here, we evaluate the response time of the system for 1K, 5K and 10K users. The querying users are selected with a Zipf ( $\vartheta = 0.8$ ) distribution<sup>5</sup>. We use exponential distribution to model the query rate, and the mean is varied between 0.5 and 60 $quh$  (Queries per User per Hour). Processing time at each node is exponentially distributed with mean 50 $ms$ . This is a realistic processing time that includes CPU

<sup>5</sup> MOBIHIDE behaves even better for uniform distribution of the querying user.

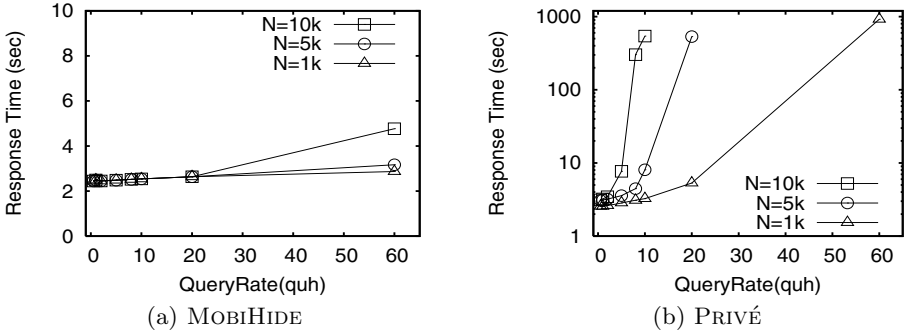
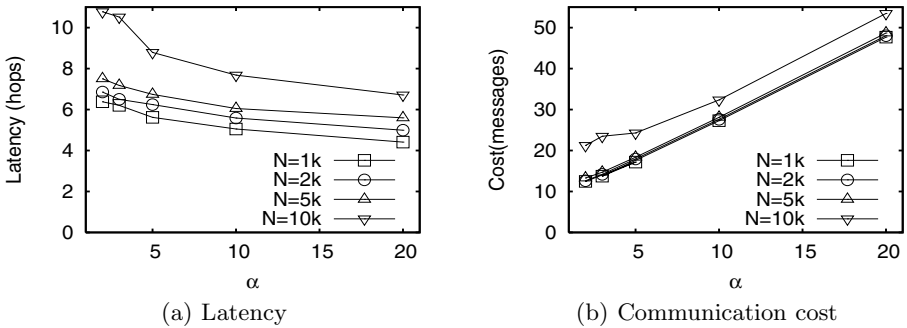
Fig. 13. Scalability,  $\mathcal{K} = 40$ 

Fig. 14. Join

processing and network buffer access. We set  $\mathcal{K}=40$  and inject queries for a period of  $600\text{sec}$ . From Figure 13(a), we can see that the response time is short (i.e., does not exceed  $5\text{sec}$ ) even for large user populations and high query rates. Note that the experiment assumes unbounded message queues at the nodes; therefore the drop rate of requests is 0. We also considered bounded queues (size = 100); in the worst case, the drop rate was 3.4%.

In Figure 13(b) we repeated the same experiment for PRIVÉ. Observe that the response time grows sharply with the query rate, due to delays at the root node. For 10K users and  $10\text{quh}$  the response time is almost  $600\text{sec}$  (whereas, MOBIHIDE needs only  $2.5\text{sec}$ ). Again, these results are for unbounded queues. For the bounded case (queue size = 100), the drop rate was 26% for  $8\text{quh}$ ; for  $10\text{quh}$  the drop rate surges as high as 60%. From the previous experiments it is obvious that MOBIHIDE outperforms PRIVÉ.

**Join.** In this experiment, we measure the latency (i.e., number of hops) and communication cost (i.e., total number of messages) for the user join operation. Starting from a stable system, an additional 10% of the initial user population joins randomly the system. Figure 14(a) shows the latency for  $N = 1\text{K}, 2\text{K}, 5\text{K}$  and  $10\text{K}$  users, for varying  $\alpha$  (recall that the cluster size is between  $\alpha$  and  $3\alpha$ ). The plot

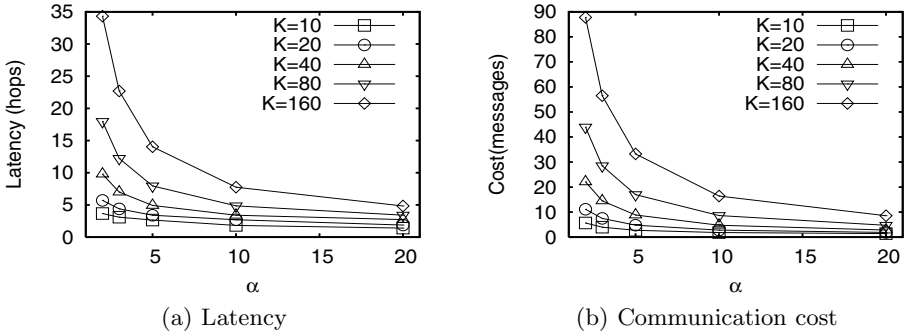
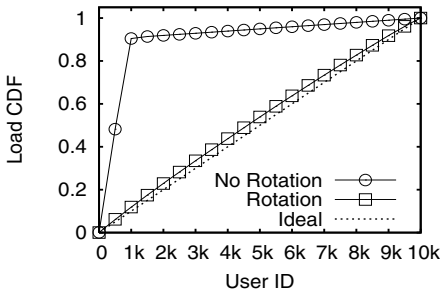
Fig. 15.  $\mathcal{K}$ -Request Operation

Fig. 16. Load Balancing

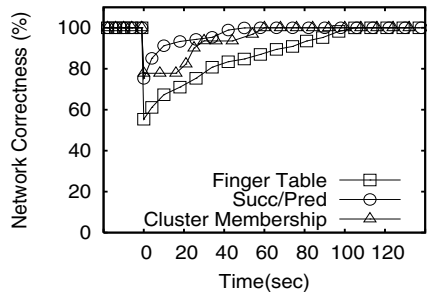


Fig. 17. Fault Tolerance

confirms the theoretical expected complexity  $O(\log N - \log \alpha)$ . For low  $\alpha$  values, we observe a slight increase, due to the increasing proportion of split operations. In terms of communication cost (see Figure 14(b)), the dominant factor is  $O(\alpha)$  due to the intra-cluster notification. There is a tradeoff between join latency and communication cost in terms of  $\alpha$ . For low  $\alpha$  values, the cluster maintenance cost is lower, but the latency increases. Furthermore, a low  $\alpha$  also causes increased latency and communication cost during  $\mathcal{K}$ -requests, as we will show shortly. Our experiments suggest that a value  $5 < \alpha < 10$  is likely to yield good results in practice.

**$\mathcal{K}$ -Request.** We consider a 10K user population with 10K uniformly distributed queries (note that the distribution does not influence the latency or communication cost in the absence of queuing). Figure 15(a) and 15(b) show the average latency and communication for constructing the  $\mathcal{K}$ -ASRs ( $\alpha$  is varied). Both the latency and communication cost are favored by larger  $\alpha$  values. However, a compromise must be reached among the  $\mathcal{K}$ -Request performance, maintenance cost and system scalability. Larger  $\alpha$  determines higher maintenance cost and also yields a more centralized system, with inferior peak-load performance.

**Load Balancing.** Due to the hierarchical nature of MOBIHIDE, the cluster heads that participate on the Chord ring bear more load than other cluster

members. Here, we evaluate the rotation mechanism of MOBIHIDE which aims at distributing the load evenly. We set  $\alpha=5$ ,  $\mathcal{K}=20$  and simulated a 10K user network, where an average of 3.6 $quh$  are generated. The total simulated time is 3 hours, and a rotation is triggered at every 300 messages received by a node. Figure 16 shows the cumulative distribution function (CDF) of the sorted node loads. Without rotation, the roughly 1,000 cluster heads (i.e.,  $10000/2\alpha$  as  $2\alpha$  is the average cluster size) bear 90% of the system load. With rotation, the load balancing is very close to the ideal (i.e., linear CDF, plotted as dotted line). Note that, for a load unit setting of 300 and a rotation cost of  $2\alpha$  messages, the rotation overhead is only  $2\alpha/300 = 3\%$ . This overhead can be decreased further by increasing the load unit.

**Fault Tolerance.** In this experiment we evaluate the fault-tolerance features of MOBIHIDE. We consider 10K users and  $\alpha=5$ . Chord performs periodical maintenance for its pointers. The respective timers are set at 3 $sec$  for the successor/predecessor, 10 $sec$  for the successor list and 30 $sec$  for the finger table pointers. The intra-cluster beacon timer  $\delta t = 10sec$ . We consider three network correctness metrics: (i) the intra-cluster correctness, measured as the ratio of correct cluster membership entries out of the total entries, (ii) the succ/pred correctness, measured as the ratio of correct successors/predecessors over the total number of successor/predecessor pointers, and (iii) we define similarly the correctness of finger tables. Note that, for correct execution of  $\mathcal{K}$ -request operations, only the successor/predecessor and intra-cluster membership need to be 100% accurate; the finger table pointers are only used for join and relocation operations, and their inaccuracy can only cause a slight increase in latency. Figure 17 shows the evolution in time of the three metrics, starting with a correct network, when 25% of the users fail simultaneously;  $t = 0$  is the time of failure. We observe that the succ/pred and intra-cluster correctness are established after 60 $sec$ . For the intra-cluster correctness, it takes the system roughly three purge intervals ( $6\delta t$ ) to detect head failure, elect new leaders and establish correct cluster membership. The finger table is restored after 120 $sec$ .

## 7 Conclusion

While location-based services become essential in supporting a broad area of applications (navigation systems, emergency services, etc), new privacy concerns arise for LBS users (e.g. in the near future GSM phones will be equipped with a “clipper” chip that accurately tracks users). In this paper, we propose MOBIHIDE, a scalable P2P system for anonymous LBS queries. MOBIHIDE indexes users into a hierarchical Chord network, according to the 1-D Hilbert ordering of their coordinates, and builds  $\mathcal{K}$ -ASRs by randomly choosing Hilbert sequences of  $\mathcal{K}$  users. Our results confirm that in practice, MOBIHIDE outperforms existing solutions: our system provides strong anonymity, it is fault-tolerant, and scales to large numbers of mobile users.

In future work, we plan to address the issue of anonymizing user trajectories, as opposed to user locations. Furthermore, we plan to investigate efficient

methods to anonymize queries for infrastructure-less environments, such as ad-hoc wireless networks (Wi-Fi, Bluetooth), where point-to-point communication channels do not exist between any pair of users, and only users within a limited physical range can be contacted.

## References

1. p2psim: The Peer-to-Peer Network Simulator, <http://pdos.csail.mit.edu/p2psim>.
2. Tor: Anonymity Online, <http://tor.eff.org/>
3. Brinkhoff, T.: A framework for generating network-based moving objects. *Geoinformatica* 6(2), 153–180 (2002)
4. Cheng, R., Zhang, Y., Bertino, E., Prabhakar, S.: Preserving User Location Privacy in Mobile Data Management Infrastructures. In: *Proc. of Privacy Enhancing Technology Workshop* (2006)
5. Chow, C.-Y., Mokbel, M.F., Liu, X.: A Peer-to-Peer Spatial Cloaking Algorithm for Anonymous Location-based Services. In: *ACM International Symposium on Advances in Geographic Information Systems*, ACM Press, New York (2006)
6. Ganesan, P., Gummadi, K., Garcia-Molina, H.: Canon in G Major: Designing DHTs with Hierarchical Structure. In: *Proc. of ICDCS*, pp. 263–272 (2004)
7. Gedik, B., Liu, L.: Location Privacy in Mobile Systems: A Personalized Anonymization Model. In: *Proc. of ICDCS*, pp. 620–629 (2005)
8. Ghinita, G., Kalnis, P., Skiadopoulos, S.: PRIVE: Anonymous Location-Based Queries in Distributed Mobile Systems. In: *Proc of WWW* (2007)
9. Gruteser, M., Grunwald, D.: Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In: *Proc. of USENIX MobiSys* (2003)
10. Hoh, B., Gruteser, M.: Protecting Location Privacy through Path Confusion. In: *Proc. of SecureComm* (2005)
11. Hu, H., Lee, D.L.: Range Nearest-Neighbor Query. *IEEE TKDE* 18(1), 78–91 (2006)
12. Kalnis, P., Ghinita, G., Mouratidis, K., Papadias, D.: Preserving Anonymity in Location Based Services. Technical Report TRB6/06, National University of Singapore (2006)
13. Kamat, P., Zhang, Y., Trappe, W., Ozturk, C.: Enhancing Source-Location Privacy in Sensor Network Routing. In: *Proc. of ICDCS* (2005)
14. Machanavajjhala, A., Gehrke, J., Kifer, D., Venkitasubramaniam, M.: l-Diversity: Privacy Beyond k-Anonymity. In: *Proc. of ICDE* (2006)
15. Mokbel, M.F., Chow, C.Y., Aref, W.G.: The New Casper: Query Processing for Location Services without Compromising Privacy. In: *Proc. of VLDB* (2006)
16. Moon, B., Jagadish, H.V., Faloutsos, C., Saltz, J.H.: Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *IEEE TKDE* 13(1), 124–141 (2001)
17. Samarati, P.: Protecting Respondents' Identities in Microdata Release. *IEEE TKDE* 13(6), 1010–1027 (2001)
18. Samet, H.: *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading (1990)
19. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications. *IEEE/ACM Transactions on Networking* 11(1), 17–32 (2003)
20. Sweeney, L.: k-Anonymity: A Model for Protecting Privacy. *Int. J. of Uncertainty, Fuzziness and Knowledge-Based Systems* 10(5), 557–570 (2002)

# Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy\*

Ali Khoshgozaran and Cyrus Shahabi

University of Southern California  
Department of Computer Science  
Information Laboratory (InfoLab)  
Los Angeles, CA 90089-0781  
{jafkhosh, shahabi}@usc.edu

**Abstract.** In this paper we propose a fundamental approach to perform the class of Nearest Neighbor (NN) queries, the core class of queries used in many of the location-based services, without revealing the origin of the query in order to preserve the privacy of this information. The idea behind our approach is to utilize one-way transformations to map the space of all static and dynamic objects to another space and resolve the query *blindly* in the transformed space. However, in order to become a viable approach, the transformation used should be able to resolve NN queries in the transformed space accurately and more importantly prevent malicious use of transformed data by untrusted entities. Traditional encryption based techniques incur expensive  $O(n)$  computation cost (where  $n$  is the total number of points in space) and possibly logarithmic communication cost for resolving a KNN query. This is because such approaches treat points as vectors in space and do not exploit their spatial properties. In contrast, we use Hilbert curves as efficient one-way transformations and design algorithms to evaluate a KNN query in the Hilbert transformed space. Consequently, we reduce the complexity of computing a KNN query to  $O(K \times \frac{2^{2N}}{n})$  and transferring the results to the client in  $O(K)$ , respectively, where  $N$ , the Hilbert curve degree, is a small constant. Our results show that we very closely approximate the result set generated from performing KNN queries in the original space while enforcing our new location privacy metrics termed *u-anonymity* and *a-anonymity*, which are stronger and more generalized privacy measures than the commonly used  $K$ -anonymity and cloaked region size measures.

## 1 Introduction

An important class of spatial queries consists of nearest-neighbor (NN) query and its variations. These queries search for data objects that minimize a distance-based function with reference to one or more query objects (e.g., points). In location-based services, a group of mobile users want to find the location of their  $K$  closest objects to their current

---

\* This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), IIS-0238560 (PECASE), IIS-0324955 (ITR), and unrestricted cash gifts from Google and Microsoft. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.



location (KNN). One obvious requirement with KNN queries is that the location of the query point(s) needs to be known in order to perform the query. However, in many applications such as in location-based services, a user may not want to reveal its location in order to preserve his/her privacy.

In this paper, we propose blind evaluation of Nearest Neighbor queries in order to preserve users' location from being revealed to location servers addressing such queries. For clarity reasons, for the rest of this paper, we will focus on location-based services in the 2-D space as the motivating application since it is clear that the query point is identical to the user location and hence its hiding preserves user's location privacy. While this application by itself is important enough to justify this research effort, we believe that the blind evaluation of KNN queries is fundamental and core to many other privacy preserving applications in sensor networks, online mapping services, geospatial information systems and numerous other applications in geospatial decision making.

Protecting users locations while responding to a KNN query is challenging due to the fact that there is an interesting dilemma in resolving such queries: while precise query location is needed to generate the result set for a KNN query, the privacy constraints of the problem does not allow revealing users' location information to the untrusted entity responding to such queries. In order to resolve this dilemma, we propose a fundamental approach based on utilizing the power of one-way transformations to preserve users' location privacy by encoding the space of all static and dynamic objects and answering the query blindly in the encoded space.

There is an inherent limitation in using traditional encryption techniques for blind evaluation of KNN queries. To illustrate, assume our server uses recently proposed encryption techniques to compute the encryption of the Euclidean distance between an encrypted point (i.e., the query origin) and each point of interest [8]. These encrypted distances can then be sent back to the client who can decrypt them and find the top  $K$  results. Trivially, this protocol satisfies our definition of blind KNN evaluation (see Section 3) since the location of neither the query point nor the result set is revealed to the server. However, the main limitation here is that distance between query point and each and every point of interest must both be computed and transferred to the client, i.e.,  $O(n)$  computation and communication complexity where  $n$  is the size of the database. There are cryptographic binary search communication protocols that may reduce the communication complexity to logarithmic; however, the computation complexity at the server cannot be reduced further. This is because the points of interest are treated as vectors with no exploitation of the fact that they are in fact points in space. Instead, we use Hilbert curves to transform original space to an encoded space stored at the server. Consequently, the server's encoded space still has the property that the nearby points stay close to each other and hence can reduce the KNNs computational complexity to  $O(K \times \frac{2^{2N}}{n})$  where  $N$ , the curve order, is a small constant. Moreover, since only the  $K$  closest points are sent back to the client, the communication complexity becomes  $O(K)$ . We also introduce two new location privacy metrics termed user-based anonymity or *u-anonymity* and area-based anonymity or *a-anonymity*. These metrics are stronger and more generalized than the privacy measures commonly used by the  $K$ -anonymity and spatial cloaking based approaches. We analytically prove that our technique satisfies these two stronger privacy metrics.

We have performed several experiments to evaluate the effectiveness of our approach. As detailed in Section 7, we show that our proposed technique achieves a very close approximation of performing KNN queries in the original space by generating a result set whose elements on average have less than 0.08 mile displacement to the elements of the actual result set in a 26 mile by 26 mile area containing more than 10000 restaurants. We also show that a malicious attacker gains almost no useful knowledge about the parameters of our encoding techniques, even when significant knowledge about the key is compromised. In other words, a nominal displacement error in approximating only one of the key parameters, (a meter displacement in a 670 square mile area) results in no useful information for compromising our encryption scheme.

We stress that our technique does not always generate the exact ground-truth answer for a query because of its nature of reducing the dimensionality of data. However we believe there are many use case scenarios in location-based services where a *satisfactory* approximation of the result is still useful as long as users' privacy is preserved.

## 2 Related Work

The closest set of studies to our work is the class that preserves user location privacy using the *cloaking* techniques. With this approach, a trusted *anonymizer* is usually in charge of receiving user's precise location information and trying to disguise it by blurring user's exact location by (for example) extending it from a *point* location to an *area* (spatial extent) and sending a region containing several other users instead of a point to the server. A similar approach based on the concept of *K-anonymity* is extending the *cloaked* area until it is large enough to include a minimum of  $K - 1$  other users. Hence, the user's location cannot be distinguished from the location of the other  $K - 1$  users in the same extended area. This extended area will then be used to resolve spatial queries such as NN queries. Several techniques based on cloaking and *K-anonymity* have been proposed in the literature to reduce the resolution of the user's location information [1, 2, 4–6, 13, 14].

Cloaking and *K-anonymity* approaches have some limitations. First, by design cloaking relies on a trusted entity to "anonymize" users' locations which means all queries should trust the *anonymizer* during the system's normal mode of operation. The anonymizer will also become a single point of failure and a potential scalability bottleneck as several handshakes must occur between the user and anonymizer to exchange user profiles and anonymity measures. Another limitation of cloaking techniques in general is that either the quality of service or overall system performance degrades significantly as users choose to have more strict privacy preferences. For example, if the user requires a better *K-anonymity*, the system needs to increase  $K$  for that user which would result in a larger cloaked area and hence less accurate query response. Alternatively, if one requires to maintain the quality of service the location server has to resolve the spatial query for each and every point in the cloaked region and send the entire bulky result to the anonymizer to be filtered out. This will obviously affect the overall system performance, communication bandwidth and server throughput and results in more sophisticated query processing. Finally, the concept of *K-anonymity* does not work in all scenarios. For example, in a less populated area, the size of the extended area can be prohibitively large in order to include  $K - 1$  other users. Some studies try to address

this limitation by proposing more robust ways of determining the area of cloaking [11]. However, they will still need a trusted anonymizer to be able to respond to user queries and in the most optimistic scenario will reveal the region a user is located in, to an untrusted location server. In Section 6, we explain how our proposed approach eliminates the need for an anonymizer and why the accuracy of the result only depends on the quality of the transformation and remains consistent for all users.

### 3 Preliminaries

In this section, we first formally define the problem of blindly evaluating a KNN query and briefly discuss our approach and its use of one-way transformations. We also study the challenges associated with finding the right transformations and review an important class of many-to-one dimensional mappings called *space filling curves* which are used in our approach to achieve location privacy.

#### 3.1 Formal Problem Definition

Given a set of static objects  $S = (o_1, o_2, \dots, o_n)$  in 2-D space, a set of users  $U = (u_1, u_2, \dots, u_M)$  and a set of dynamic query points  $Q = (q_1, q_2, \dots, q_m)$ , the KNN query with respect to query point  $q_i$  finds a set  $S' \subset S$  of  $K$  objects where for any object  $o' \in S'$  and  $o \in S - S'$ ,  $D(o', q_i) \leq D(o, q_i)$  where  $D$  is the Euclidean distance function. In a typical KNN query scenario, the static objects represent points of interest (POI) and the query points represent user locations. We now define some of the properties a location server should possess in order to enable user location protection while responding to a KNN query.

*Definition 1. u-anonymity:* While resolving a KNN query, the user issuing the query should be indistinguishable among the entire set of users. In other words, for each query  $Q$ ,  $P(Q) = \frac{1}{M}$  where  $P(Q)$  is the probability that query  $Q$  is issued by a user  $u_i$  and  $M$  is the total number of users. Note that this definition ensures the server does not know which user queried from a point  $q_i$ ; however, we also need to ensure that the server does not know which point the query  $Q$  is issued from. This requirement is captured in Definition 2.

*Definition 2. a-anonymity:* While resolving a KNN query, the location of the query point should not be revealed. In other words, for each query  $Q$ ,  $P'(Q) = \frac{1}{\text{area}(A)}$ , where  $A$  is the entire region covering all the objects in  $S$ , and  $P'(Q)$  is the probability that query  $Q$  was issued by a user located at any point inside  $A$ .

Note that Definitions 1 and 2 impose much stronger privacy requirements than the commonly used  $K$ -anonymity [18, 4, 5, 11, 1, 6, 13, 14], in which a user is indistinguishable among  $K$  other users or his location is blurred in a cloaked region  $R$ . The above definitions of location privacy are free of metrics such as  $K$  and  $R$ . They are in fact identical to an extreme case of setting  $R = A$  for spatial cloaking, and an extreme case of setting  $K = M$  for  $K$ -anonymity.

*Definition 3. Result set anonymity:* The location of all points of interest in the result set should be kept secret from the location server. More precisely  $\dot{P}(o_j) = 1/n$  for

$j = 1 \dots n$  where  $\dot{P}(o_j)$  is the probability that  $o_j$  is a member of the result set for query  $Q$  and  $n$  is the total number of POI's.

*Definition 4. Blind evaluation of KNN:* We say a KNN query is blindly evaluated if the *u-anonymity*, *a-anonymity* and *result set anonymity* constraints defined above are all satisfied. In other words, in blind evaluation of KNN, the identity and location of the query point as well as the result set should not be revealed. We term our approach *blind evaluation of KNN queries* because it attempts to prevent any leak of information to essentially blind the server from acquiring information about a user's location. For the rest of the paper, we use the term *user* to refer to the user located at query point  $P$  issuing the query  $Q$ . The following example shows how the above properties should be satisfied in a typical KNN query. Suppose a user asks for his 3 closest gas-stations. In this case a *malicious entity* should acquire neither the location of the user (i.e., *a-anonymity*) nor its identity (i.e., *u-anonymity*) nor the actual location or identity of any of the 3 closest gas stations in the response set (i.e., *result set anonymity*) while the user should receive the actual points of interest matching his query.

Based on the above properties, we term a location server *privacy aware* if it is capable of blindly evaluating a KNN query while providing accurate results. The challenge in blind evaluation of KNN queries is that the above two constraints cannot be perfectly met at anytime. If precise KNN is desired for each query, one should reveal his exact location and this violates the privacy constraint imposed on the problem (i.e., preserving user's location). Therefore an ideal approach should respect both constraints as much as possible i.e., it should be a very *close* approximation of  $S'$  (we will define the notion of closeness in Section 3.2) while keeping  $P(Q)$ ,  $P'(Q)$  and  $\dot{P}(o_j)$ 's as low as possible.

### 3.2 Space Encoding

In this section, we introduce our novel approach for protecting user's location from the malicious location servers. Our approach is based on transforming the static objects in the 2-D space as well as dynamic query points by mapping them to another space using a one-way transformation and addressing the query in the transformed space. As mentioned earlier we address the issue of location privacy in the context of location-based services and thus focus on the 2-D space of static objects (i.e., points of interest) and dynamic query points (i.e., users). Transforming such a 2-D space requires using a one-way function to map each point from the original space to a point in the transformed space. A transformation is one-way if it can be easily calculated in one direction (i.e., the forward direction) and is computationally impossible to calculate in the other (i.e., backward) direction [20]. The process of transforming the original space with such a one-way mapping can be viewed as *encrypting* the elements of the 2-D space. With this view, in order to make decryption possible the function has to allow fast computation of its inverse given some extra knowledge, termed *trapdoor* [19]. In practice, many one-way transformations may be reversible even without the knowledge of the trapdoor but the process must be too complex (equivalent to exhaustive try) to make such transformation computationally secure.

Therefore, as depicted in Figure 1, any one-way transformation which respects the proximity of the original space can replace the first black box in Figure 1 to make the

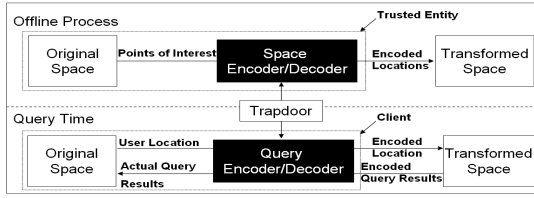


Fig. 1. Space Encoding

location server *privacy aware*. We view such one-way transformations as modules that *encode* the original space into another space which is capable of addressing encoded transformed queries. In order to enable decoding of query results one should define the notion of a trapdoor or a *key* for the space encoding module. In this paper we use the properties of our mapping function as the trapdoor for fast decryption of query results. Such trapdoor will be only provided to the user to reverse the encoded results and get the response set back in its original format.

Note that hiding the location of the query point is different than hiding its identity and the focus of this paper is on hiding the locations of the query points and result sets. No matter what type of space encoder is used, a user will only have to report its encoded position to the location server in order to get the exact location of her result set back. In Section 6 we will discuss how this property separates user anonymization issues from protecting user’s location.

Identifying the right *space encoders* is very challenging because there are several one-way transformations which could be applied to a 2-D space of objects (e.g., random perturbation of points), however, the majority of such transformations do not respect the notion of distance and proximity. The transformations that respect such properties are the only candidates resulting in satisfactory KNN query in an encoded space. We term such transformations *complete KNN-invariant* if performing the KNN query in the transformed space and decoding the result set back to the original space, generates a result set exactly equal to the result set obtained from performing the query in the original space. However, as we will discuss in Section 4, our proposed approach generates an approximation of the actual result for each KNN due to its nature of reducing the dimensionality of data. Therefore we define a weaker notion of *closeness* for a transformation and call it semi KNN-invariant (or for simplicity KNN-invariant) if it yields satisfactory values for the two metrics introduced in the following definition.

*Definition 5.* Suppose the actual result of a KNN query, issued by a user located at point  $Q$  is  $R = (o_1, o_2, \dots, o_K)$ , and it is approximated by a transformation  $T$  as  $R' = (o'_1, o'_2, \dots, o'_K)$ .  $T$  is KNN-invariant if it yields acceptable values for the following two metrics:

*Metric 1:* The *Resemblance*, denoted by  $\alpha$ , defined as

$$\alpha = \frac{|R \cap R'|}{|R|} \tag{1}$$

where  $|R|$  denotes the size of a set  $R$ . In fact  $\alpha$  measures what percentage of the points in the actual query result set  $R$  are included in the approximated result set  $R'$ .

*Metric 2: The Displacement*, denoted by  $\beta$ , defined as

$$\beta = \frac{1}{K} \left( \sum_{i=1}^K \|Q - o'_i\| - \sum_{i=1}^K \|Q - o_i\| \right) \quad (2)$$

where  $\|Q - o_i\|$  is the Euclidean distance between the query point  $Q$  and  $o_i$ . Therefore  $\beta$  measures how *closely*  $R$  is approximated by  $R'$  on average. Obviously, since  $R$  is the ground truth,  $\beta \geq 0$ .

Although there is no fixed threshold for acceptable  $\alpha$  and  $\beta$  values, depending on the application and the scenario, certain values may or may not be considered satisfactory. In Section 7 we will evaluate our approach against these two metrics and will show that it uses an effective KNN-invariant transformation.

In this paper, we study an important class of transformations called space filling curves as candidate space encoders for our framework. Such curves have interesting properties which have made them a popular tool in different domains such as querying multi-dimensional data and image compression [12, 16]. It is important, however, to note that we are not claiming that space filling curves are the best possible encoders. In Section 3.3 we show how such space filling curves can be treated as one-way functions if certain properties of those curves are kept secret from malicious attackers.

### 3.3 Space Filling Curves

Introduced in 1890 by an Italian mathematician G. Peano [17], space filling curves belong to a family of curves which pass through all points in space without crossing themselves. The important property of these curves is that they retain the *proximity* and *neighboring* aspects of the data. Consequently, points which lie close to one another in the original space mostly remain close to each other in the transformed space. One of the most popular members of this class is Hilbert curves [7] since several studies show the superior clustering and distance preserving properties of these curves [3, 9, 12, 15].

Similar to [15] we define  $H_d^N$  for  $N \geq 1$  and  $d \geq 2$ , as the  $N^{\text{th}}$  order Hilbert curve for a  $d$ -dimensional space.  $H_d^N$  is therefore a linear ordering which maps an integer set  $[0, 2^{Nd} - 1]$  into a  $d$ -dimensional integer space  $[0, 2^N - 1]^d$  as follows:

$H = \ell(P)$  for  $H \in [0, 2^{Nd} - 1]$ , where  $P$  is the coordinate of each point in the  $d$ -dimensional space. We call the output of this function its *H-value* throughout the paper. Note that it is possible for two or more points to have the same H-value in a given curve.

As mentioned above, our motivating application is location privacy and therefore we are particularly interested in 2-D space and thus only deal with 2-D curves ( $N = 2$ ). Therefore  $H = \ell(X, Y)$  where  $X$  and  $Y$  are the coordinates of each point in the 2-D space. Figure 2 illustrates a sample scenario showing how a Hilbert curve can be used to transform a 2-D space into H-values. In this example, points of interest (POI) are traversed by a second order Hilbert curve and are *indexed* based on the order they are visited by the curve (i.e.,  $H$  in the above formula). Therefore, in our example the points  $a, b, c, d, e$  are represented by their H-values 7, 14, 5, 9 and 0, respectively. Depending on the desired resolution, more fine-grained curves can be recursively constructed as depicted in Figure 3.

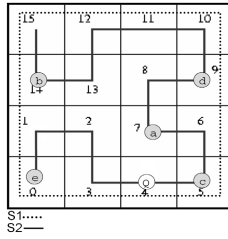


Fig. 2. A  $H_2^2$  Pass of the 2-D Space

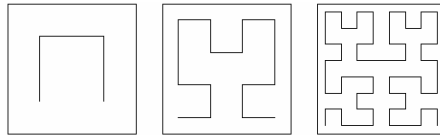


Fig. 3. First Three Orders of Hilbert Curves

As we will show in Section 7 the most interesting feature of Hilbert curves is how they can act as KNN-invariant transformations with satisfactory values of  $\alpha$  and  $\beta$  when used in location-based services. This property suits our approach as we are interested to address the KNN query in a transformed space and still get satisfactory results. Furthermore, another important property of a Hilbert curve that makes it a very suitable tool for our proposed scheme is that  $\ell$  becomes a one-way function if the curve parameters are not known. These parameters, which collectively form a *key* for this one-way transformation, include the curve’s starting point  $(X_0, Y_0)$ , curve orientation  $\theta$ , curve order  $N$  and curve scale factor  $\Gamma$ . We term this key, *Space Decryption Key* or *SDK* where  $SDK = \{X_0, Y_0, \theta, N, \Gamma\}$ .

Therefore a malicious entity, not knowing this key, has to exhaustively check for all combinations of curve parameters to find the right curve by comparing the H-values for all points of interest. As we show in Theorem 1, we make it computationally impossible to reverse the transformation and get back the original points. Even a nominal error in approximating curve parameters will generate a completely different set of H-values. We now prove two important properties of our approach which give more insight on the security of our proposed method.

**THEOREM 1.** The complexity of a brute-force attack to find the transformation key discussed above is  $O(2^{4p})$  where  $p$  is the number of bits used to discretize each parameter.

**PROOF.** In order to accurately find the curve’s starting point, it should exactly lie on the intersection of two edges (lines) coming from each of the  $X$  and  $Y$  axes. Therefore one has to locate the exact values of both  $X_0$  and  $Y_0$  in the continuous domain of  $X$  and  $Y$  axes. Theoretically, the probability of finding the right value for the above two parameters in a continuous space is zero. However, in real world scenarios, the attacker can approximate  $X_0$  and  $Y_0$  by constructing the finest grid possible to guarantee

that his best guess  $(X'_0, Y'_0)$  located at an intersection of two edges, lies very close to  $(X_0, Y_0)$  so that  $|X_0 - X'_0| \leq \varepsilon$  and  $|Y_0 - Y'_0| \leq \varepsilon$ . When  $\varepsilon$  is sufficiently small then replacing  $(X_0, Y_0)$  with  $(X'_0, Y'_0)$  generates a set of H-values indifferentiable from the original set. The attacker should thus search the entire space exhaustively for a very close approximation of this starting point. Using  $p$  bits the attacker can generate  $2^p$  candidate values on each axis. Therefore, assuming a square region covering all POI's, the attacker's entire search space for the starting point will have  $2^p * 2^p$  elements. Similarly, the entire continuous  $360^\circ$  space for  $\theta$  should be discretized to the finest possible extent to ensure that  $|\theta - \theta'| \leq \varepsilon$  for at least one value of  $\theta'$ . With  $q$  bits, that attacker can generate  $2^q$  different candidate values of  $\theta$  each corresponding to a curve orientation. The curve scale factor  $\Gamma$  is a continuous number between 0 and 1 and thus similarly,  $r$  bits can divide the 0 to 1 range into  $2^r$  values each can approximate  $\Gamma$  so that  $|\Gamma - \Gamma'| \leq \varepsilon$  for at least one value of  $\Gamma'$ . Assuming  $N$  different possibilities for the curve order, the entire solution space will have  $2^p * 2^p * 2^q * 2^r * N$  elements. Assuming  $2^q = O(2^p)$  and  $2^r = O(2^p)$  and since  $N \ll 2^p$ , the complexity of an exhaustive search is  $O(2^{4p})$  where  $p$  is the number of bits used by the attacker to represent each parameter.  $\square$

Note that for a given  $N$ , there is an upper bound for  $p$ , after which there is no reason to increase  $p$  further because  $\varepsilon$  becomes sufficiently small to estimate  $X_0$  and  $Y_0$  accurately. However, by simply increasing  $N$  to  $N + 1$  we can make the curve twice condense in each direction that results in a new threshold of  $\varepsilon' = \frac{\varepsilon}{2}$  for the curve's starting point and similar tighter thresholds for other curve parameters. Therefore, a linear increase in  $N$  will make  $\varepsilon$  exponentially smaller and thus  $p$  should increase linearly with  $N$  as well to ensure close approximation of curve parameters. However, as Theorem 1 shows, increasing  $p$  will result in an exponential increase of the search space. Consequently,  $N$  is chosen large enough to make reversing an  $H_2^N$  mapping impossible and thus to make  $H$  act as a one-way mapping. Hence, we consider this transformation as a *space encryption scheme* whose key is the curve parameters (i.e., *SDK*).

**THEOREM 2.** Using an  $H_2^N$  Hilbert curve to encode the space satisfies the *a-anonymity*, *u-anonymity* and *result set anonymity* properties defined in Section 3.1.

**PROOF.** An  $H_2^N$  fills a  $2^N * 2^N$  grid in the 2-D space visiting each point exactly once. Theorem 1 states that having an H-value for the query point  $Q$ , one cannot reverse the process to find  $\ell^{-1}(X_Q, Y_Q)$  because  $H$  is one-way for large values of  $N$  (i.e., curve degree) and thus  $Q$  cannot be located anywhere in the grid. With  $n$  and  $A$  being the total number of POI's and the entire region covering these  $n$  objects, respectively (see Definition 1), there are  $2^p * 2^p$  equiprobable choices for the location of  $Q$  and thus  $P'(Q) = 1/2^{2p} = \frac{1}{\text{area}(A)}$ . Furthermore, since no information beyond the H-value of the query point is needed to resolve the query,  $Q$  could be issued by any user  $u_i$  and thus  $P(Q) = \frac{1}{M}$  where  $M$  is the total number of users. Finally, for each static object  $o$ ,  $\ell^{-1}(X_o, Y_o)$  cannot be found and thus  $\dot{P}(o_i) = P'(Q) = 1/2^{2p} \ll 1/n$  because  $2^{2p} \gg n$ .  $\square$



## 4 2-Phase Query Processing

Making a query processing engine privacy-aware based on our idea of space transformation discussed above, requires a two-step process consisting of an offline encryption of original space followed by online query processing. First, during an offline process, necessary data structures and encryption schemes are utilized to encode the space of POI's. Next, during an online process, the query is resolved in the transformed space and is then decoded to obtain the original points satisfying the query in the 2-D space. The following sections describe the details of these two phases and the modules performed in each phase.

### 4.1 Offline Space Encryption

Figure 4 depicts Algorithm 1, the Offline Space Encryption algorithm. The first step of this phase is to choose the curve parameters from which the curve will be constructed and the value of *SDK* will be determined. These parameters are listed in Section 3.3. Next, assuming the entire area covering all points of interest is a square  $S_1$ , an  $H_2^N$  Hilbert curve is constructed starting from  $(X_0, Y_0)$  in a (possibly larger) square  $S_2$  surrounding  $S_1$  until the entire  $S_2$  is traversed (see Figure 2). After visiting each point  $P$ , its H-value =  $\ell(P.X, P.Y)$  is computed using *SDK*. We use an efficient bitwise interleaving algorithm from [3] to compute the H-values for points of interest. This process is performed once for all points of interest and thus at the end of this step, a look-up table *DB* which consists of H-values for all POI's is constructed. Note that the size of *DB* is only dependant upon the number of POI's to be indexed and not the size of the region in which they are located. The result of applying Algorithm 1 on the example from Figure 2 looks like the following look-up table:  $DB = \{(0), (5), (7), (9), (14)\}$  where each element is the point's index in the curve (i.e., its H-value).

### 4.2 Online Query Processing

Algorithm 2 (Figure 5) summarizes the online query resolution process and its two modules KNN-Encode and KNN-Resolve. Using the look-up table *DB*, we can now show how the result of a KNN query is evaluated in the transformed space. For each query point  $Q$  located at position  $(X_Q, Y_Q)$ , KNN-Encode uses *SDK* to compute  $H = \ell(X_Q, Y_Q)$ . The value of  $H$ , along with  $K$  (i.e., the number of desired nearest neighbors), is all KNN-Resolve needs to resolve a query using *DB*. During this phase we begin searching from both directions in *DB* starting from  $\ell(X_Q, Y_Q)$  until  $K$  closest

```

KNN-CreateIndex (S) {
  Generate SDK=F(X0,Y0,θ,Γ);
  While (S ≠ ∅) {
    nextPoi=S.getNextPOI();
    H= ℓ(nextPoi.X,nextPoi.Y);//using SDK
    DB=DB U (H);
  }
}

```

Fig. 4. Offline Space Encryption

<pre> KNN-Encode(K,X<sub>Q</sub>,Y<sub>Q</sub>) {   H= ℓ(X<sub>Q</sub>,Y<sub>Q</sub>);//using SDK   R= KNN-Resolve (H,K);   While {R ≠ ∅}   {     O=R.GetNextH-Value();     {Poi.X, Poi.Y}= ℓ<sup>-1</sup>(O) //using SDK     Result = Result U Poi;   }   Result = Sort (Result); } </pre>	<pre> KNN-Resolve (H,K) {   IndexMore=H; IndexLess=H-1;   Count=0; R= ∅;   While (Count&lt;K)   {     if (DB.containsKey(qIndexMore))       While (qIndexMore.HasMoreElem &amp; count&lt;K){         R= R U qIndexMore.NextPOI; count++; }     if (DB.containsKey(qIndexLess))       While (qIndexLess.HasMoreElem &amp; count&lt;K){         R= R U qIndexLess.NextPOI; count++; }     qIndexMore++; qIndexLess--;   }   Return R={ℓ(o<sub>1</sub>), ℓ(o<sub>2</sub>)... , ℓ(o<sub>k</sub>)}; } </pre>
---	---

**Fig. 5.** Online KNN Query Resolution

matches are found. Note that these matches are nothing but  $K$  (distinct or overlapping)  $H$ -values. Knowing  $SDK$ , KNN-Encode transfers the result set back to the original 2-D space, using  $H^{-1}$  to decrypt the  $H$ -values of all points in result set. To illustrate, in our example, having  $K = 3$ , and  $Q = (2, 0)$ , KNN-Encode computes  $H = 4 = \ell(2, 0)$  and calls KNN-Resolve(4, 3) to obtain  $R = \{(0), (5), (7)\}$ . Next,  $H^{-1}$  is applied to all above  $H$ -values to obtain their original 2-D coordinates.

We can now derive the complexity of the KNN-Resolve module which represents the overall query processing complexity. As discussed in Section 4.1, an  $H_2^N$  Hilbert curve divides the entire space into  $2^{2N}$  equally spaced indices. This division, results in an average density of  $\frac{n}{2^{2N}}$  POI's per each  $H$ -value where  $n$  is the total number of POI's. Therefore, finding the  $K$  closest objects in this space will on average require only  $K \times \frac{2^{2N}}{n}$   $H$ -value comparisons. Therefore, the overall complexity of our online query processing scheme is  $O(K \times \frac{2^{2N}}{n})$  compared to  $O(n)$  if traditional encryption schemes were used. Also the communication complexity of our scheme is  $O(K)$  since the result set generated by KNN-Resolve includes only the  $K$  matching points compared to an  $O(\log(n))$  complexity using traditional encryption schemes and to  $K$ -anonymity or cloaking approaches in which the query result has to be generated for  $K - 1$  other points or an entire region, respectively. The efficiency of our query processing algorithm is more pronounced with real-world datasets where the value of  $n$  is significantly large.

Notice that the order of the result set might not be accurate because there are cases in which elements with smaller difference in  $H$ -values to  $Q$  are actually located further from it compared to other objects with larger  $H$ -value difference. However, this is essentially resolved by simply having the entire result set back in its original format. Knowing  $Q$ 's location, KNN-Encode sorts the result set in the correct order. This is a very efficient process given the relatively small values of  $K$ . In addition, it is important to note that the result set of a KNN query might not precisely match the actual  $K$  nearest neighbors of a user because of loss of a dimension in the transformed space. Depending on different curve parameters and the data distribution, the accuracy of the result may vary. In Section 7 we conducted several experiments with real-world datasets and show that the *Resemblance* and *Displacement* values are acceptable for many real applications.

### 5 Dual Curve Query Resolution

Using a single Hilbert curve as a space encoder for KNN query processing discussed in Section 4 has two major drawbacks. We first discuss these two drawbacks and then introduce our *Dual Curve Query Resolution* approach or *DCQR* which overcomes the weaknesses of the former scheme and generates significantly more satisfactory results.

A closer study of Hilbert curves reveals two important properties of such curves. First, consider the 1st degree curve of Figure 6 (the left image). The curve naturally is constructed by traversing a U-shaped pattern. Regardless of its orientation, such a curve will fill the space at a specific direction at any given time sweeping the space in a clockwise fashion. Starting from the first degree curve of Figure 6, the curve misses one side in its first traversal. As the curve order grows, the number of missed sides grows exponentially as well so that an  $H_2^N$  curve misses  $M = 2^{2N} - 2^{N+1} + 1$  sides of a  $(2^N - 1)$  by  $(2^N - 1)$  grid. The above property of the curve will make H-values of certain points farther as  $N$  increases. For instance the Euclidean distance between points  $a$  and  $d$  is similar to the that of points  $b$  and  $c$  in the original 2-D space, however due to the above property,  $a$  and  $d$ 's H-values will be significantly further from each other as compared to H-values of  $b$  and  $c$ . This difference grows exponentially as  $N$  grows. Therefore points closer to two quadrants of the space (i.e., the first and last quadrants filled by the curve) will be *spatially furthest* from one another in the transformed space.

The second drawback of using a single Hilbert curve is due to the fact that such space-filling curves essentially reduce the dimensionality of the space from 2 (or in general case  $N$ ) to 1. Naturally, each element in the 1-D space constructed by the Hilbert curve will have two nearest neighbors compared to the original case where each element (except those at the edges) has four (or in general case  $2N$ ) nearest neighbors. Therefore as [10] suggests, in the best case scenario, only half of these nearest neighbors in 2-D space will remain a nearest neighbor of the same point in the transformed 1-D space.

The above two properties result in a loss of precision and thus a negative effect on overall quality of returned results. We mitigated this issue by replicating the same curve and rotating it 90 degrees. Our intuition is to index the same data simultaneously by two perpendicular curves and ask each one independently to resolve a KNN query using modules discussed in Section 4. Having two different result sets in the original domain, we merge the results and choose the  $K$  best candidates among the  $2K$  points of the sets.

We now discuss how *DCQR* ensures a better quality of results. By rotating the degree  $N$  curve, all lower degree curves constructing the main curve will be rotated as well. At each curve order, the curve rotation ensures that the missed sides generated by the discontinuation of the curve (such as the missed sides between points  $a$  and  $d$  in

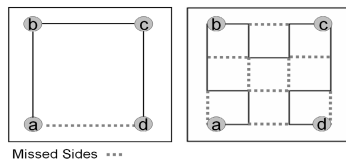


Fig. 6. Missed Sides of 2 by 2 and 3 by 3 Grids for  $H_2^1$  and  $H_2^2$ , respectively

Figure 6, will be covered by the rotated curve. Therefore, the points deemed spatially far from each other in one curve will be indexed correctly in the other curve. This will address the first issue when using Hilbert curves for indexing. *DCQR* also mitigates the effect of the second property discussed above by transforming the 2-D space to two 1-D spaces. Therefore each point will now have two nearest neighbors in each curve. It is important however, to note that these two neighbor pairs can (and do) often have overlaps and that is the main reason the dual curve approach will generate (significantly more accurate) approximate answers. Furthermore, with regards to complexity, knowing the first curve's *SDK* makes it easy to derive the key for the second curve (curve order and scale factor are the same while curve orientation and starting point are rotated 90 degrees). Therefore, the complexity of finding *DCQR*'s keys differs from what we derived for a single curve approach in Section 3.3 by a constant factor. The query processing discussed in Section 4 should also be modified slightly to work with the new dual curve scheme as follows (note that these modifications do not change the query computation and communication complexities derived in Section 4.2).

### 5.1 Offline Space Encryption for *DCQR*

During this phase, we again assume that the entire static objects set is located inside a square  $S1$ . Consequently two Hilbert curves  $H_2^N$  and  $H_2^N$  are constructed based on *SDK* to sweep the (possibly larger) square  $S2$  (surrounding  $S1$ ), until the entire  $S2$  is traversed. Visiting each point,  $H$  and  $H'$  will compute  $\ell(X, Y)$  and  $\ell'(X, Y)$  respectively in the similar fashion discussed in Section 4.1. After this process is performed once for all POI's, the two sequences of  $H$  and  $H'$ -values will form two separate look-up tables  $DB$  and  $DB'$ .

### 5.2 Online Query Processing for *DCQR*

Similarly, the query processing follows the logic from Section 4.2 with the difference that for each query point  $Q$ , we compute  $H = \ell(X_Q, Y_Q)$  and  $H' = \ell'(X_Q, Y_Q)$  using *SDK* and *SDK'*, respectively. We then initiate two parallel query resolution schemes applying  $H$ -value and  $H'$ -value to  $DB$  and  $DB'$ , respectively and simultaneously retrieve  $K$  closest matches for each curve separately. Similar to Section 4.2, we decrypt the two results sets and choose the  $K$  best candidates (based on their Euclidean distance to  $Q$ ).

## 6 Proposed End-to-End Architecture

In previous sections, we showed in detail how we can utilize Hilbert curves as space encoders to blindly resolve KNN queries. As we mentioned earlier, the focus of this paper is on hiding locations and not *identities* of static objects or query points. However, in order to propose a complete solution, we briefly discuss how we can extend our proposed scheme to deal with non-spatial attributes of each POI (such as its identity or name) in the following way; Similar to *SDK*, we define a *Textual Decryption Key* or *TDK*, which is used to encrypt (decrypt) the non-spatial attributes of each POI during the offline space encryption (online query resolution) phase. Therefore, during the offline phase, in addition to the steps discussed in Section 5.1, we generate *TDK*

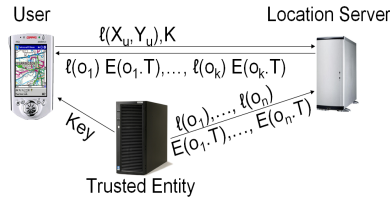


Fig. 7. DCQR Architecture for KNN Query Processing

and  $TDK'$  and use them to encrypt the textual attributes of each point  $P$  represented by  $E(P.T)$  where  $E$  is the function used to encrypt  $P.T$  (the textual attributes of  $P$ ). The above modifications will add a new attribute to  $DB$  and  $DB'$  to include any textual information of POI's. Therefore  $DB$  and  $DB'$  become look-up tables with the schemas (H-value,  $E(o.T)$ ) and ( $H'$ -value,  $E'(o.T)$ ), respectively.

### 6.1 End-to-End Query Processing

We are now ready to explain how a KNN-invariant one-way transformation can be used for blinding KNN queries in location-based services. The *client* (e.g., a portable device) issues a K-nearest-neighbor (KNN) query and provides its own location. Without loss of generality, we assume the client location is a point and is identified by two values such as its latitude and longitude. In order to make the location server privacy-aware, we first assume the architecture of Figure 7 which details the sequence of client-server communications required in order to resolve a KNN query and use the algorithms discussed in Section 5 to modify the classic location-based services architecture in the following three ways:

1) A *trusted entity* is added to the architecture. The main task of the trusted entity is to perform the KNN-CreateIndex module once and to create and update their encoded indexes and identities. A second functionality of the trusted entity is to provide users with  $(SDK, SDK')$  and  $(TDK, TDK')$  pairs required to decrypt query results. We refer to these four values as *Key Pairs*. Finally, the trusted entity provides the location server with the two look-up tables  $DB$  and  $DB'$  instead of the original dataset and keeps the two key pairs secret from the location server. Note that unlike an anonymizer, the trusted entity is not involved in the query processing.

2) Users will perform the KNN-Encode module and use the two key pairs embedded in their devices to decrypt the result set returned to them from the location server and get back the location of the returned points as well as their textual attributes. Note that in order to prevent users from being able to access the encrypted result set received from the location server and learning the transformation, the key pairs should be embedded in tamper-proof devices. Furthermore, in order to remain anonymous, users generate a random *session-id* for each KNN query request in order to enable the client and server to communicate with each other during the course of each KNN query.

3) The un-trusted location server will perform the KNN-resolve module to construct the two results sets and returns them to the user.

## 7 Experimental Evaluation

We have conducted several experiments to evaluate the performance of our proposed approach. The effectiveness of *DCQR* is determined in terms of 1) the effect of the curve order  $N$  on our proposed indexing, 2) accuracy of the result sets in terms of the Displacement and Resemblance metrics defined in Section 3.2, using *DCQR* instead of a single curve, and 3) *DCQR*'s vulnerability to attacks. We were unable to compare our approach with other approaches discussed in Section 2 because they mostly evaluate performance, based on the size of the  $K$ -anonymity set, the size of the cloaked region or the effectiveness of the anonymization techniques used and our approach is free of these metrics and satisfies stronger privacy metrics defined in Section 3. We have also performed other experiments that investigate the effect of other key parameters on quality of indexing and demonstrate our fast overall system response time (typically less than 0.5 seconds even for large values of  $K$  and  $N$ ). We do not discuss these experiments here due to lack of space and we plan to fully investigate them in an extended version of this paper. Our experiments are performed on a real-world dataset obtained from NAVTEQ covering a 26 mile by 26 mile area surrounding the city of Los Angeles which contains more than 10000 restaurants. Experiments were run on an Intel *P43.20* GHz with 2 GB of RAM.

### 7.1 The Curve Order $N$

In our first set of experiments, we evaluate the effectiveness of our proposed indexing technique. It is important to analyze the curve behavior for different values of  $N$  (i.e., curve order) and to decide on the value of *SDK* and use it throughout the rest of our experiments. For the first set of experiments, we measure the effectiveness of two  $H_2^N$  curves in indexing POI's for fixed values of  $X_0 = Y_0 = \theta = 0$  and  $\Gamma = 1$  and varying  $N$  from 1 to 15 for the first curve (note that *SDK* of the dual curve, i.e., *SDK'*, can be derived from *SDK*). We measure the minimum and average number of POI's which are assigned the same H-value for each value of  $N$ . It is clear that having a large number of POI's with the same H-value has a negative effect on Resemblance and Displacement metrics because the location server has no way of choosing a *closer* point, in a set of POI's with the same H-value while responding to a KNN query. Varying  $N$ , makes an entirely different curve and thus changes the assignments of H-values to POI's significantly. Figure 8 shows how POI/H-value changes with  $N$ . It suggests acceptable values of this number (i.e., POI/H-value  $\leq 2$ ) for curves with  $N \geq 8$ . Our next experiments confirm this intuition.

### 7.2 The Single Curve Approach vs. *DCQR*

In the second sets of experiments we first compare the single curve approach with *DCQR* in terms of the Displacement and Resemblance metrics defined in Section 3.2. As Figure 9 illustrates, for different curve orders (i.e.,  $N$ ) and different values of  $K$  (i.e., different KNN queries), *DCQR* outperforms the single curve approach for both metrics, achieving lower average Displacement and higher Resemblance values.

Next, we evaluate *DCQR* using the same metrics. As shown in Figure 10, for a fixed value of  $N = 12$ , an increase in  $K$  improves the Resemblance while it does not have a

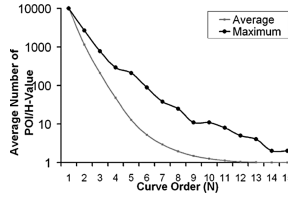


Fig. 8. Curve Order Vs. H-Values

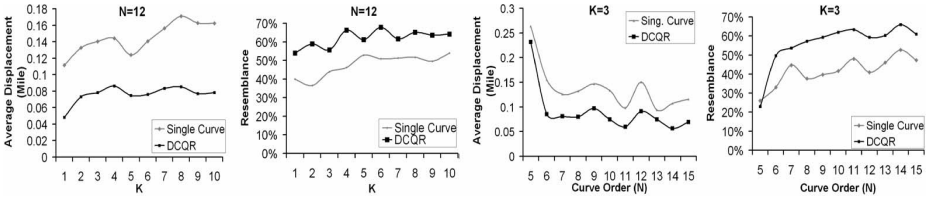


Fig. 9. Comparing Single Curve Approach vs. *DCQR* for Different Values of *K* and *N*

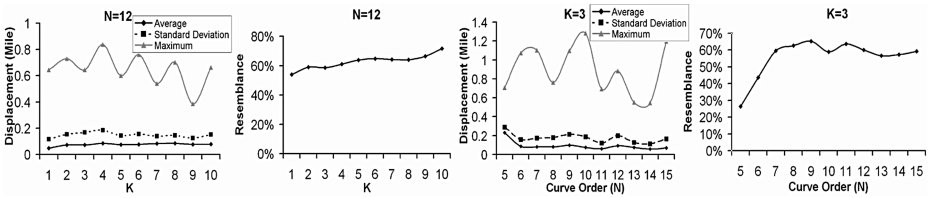
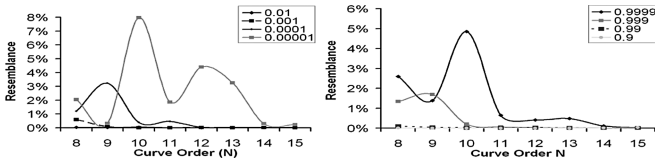


Fig. 10. *DCQR* Performance vs. *K* and *N*

significant effect on the Displacement. The reason is that as *K* increases, the result set size grows twice as fast (since using *DCQR*, its size is  $2K$  for each *KNN* query) which in turn increases the chance of visiting the right points as we move on the curve. However, searching for more *POI*'s on the curve also causes moving further away from the query point's index on the Hilbert curves which might increase the probability of hitting a missed side and thus including a false positive in the result set. However this negative effect is nominal and the Displacement stays less than 0.08 mile for all possible values of *K*. Similarly, for a fixed value of  $K=3$ , while the Displacement takes satisfactory values (less than 0.09 mile on average) for  $N \geq 8$ , Resemblance usually improves as *N* grows, confirming our intuition from the first set of experiments. Similar trends were observed for other fixed values of *K* and *N*.

### 7.3 *DCQR*'s Vulnerability to Attacks

Our last set of experiments empirically evaluates the vulnerability of our proposed approach against malicious attackers to confirm the hypotheses discussed in Section 3.3 for the one-wayness of transformations used in *DCQR* and the security of *SDK* based



**Fig. 11.** Attacking SDK by Approximating  $|Y_0 - Y'_0|$  (left) and  $\frac{\Gamma}{\Gamma'}$  (right)

on the following two extreme scenarios. First we assume the malicious location server (which is capable of becoming the most powerful attacker due to its access to  $DB$  and  $DB'$ ), has somehow gained precise knowledge for the values of  $X_0, \theta, \Gamma$  and  $N$  and only needs to find  $Y_0$ . Using  $p$  bits, it divides the Y-axis to  $2^p$  distinct values hoping to get close enough to  $Y_0$ . For each of its guesses  $Y'_0$ , the location server forms an *SDK* and performs the *KNN-CreateIndex* module to compare the resulting look-up table against  $DB$  (or  $DB'$ ) and measures the Resemblance metric to evaluate  $Y'_0$ . Figure 11 (left) illustrates the result of this attack for  $p$  taking 12, 15, 18 and 22 bits (which correspond to a minimum of  $10^{-2}, 10^{-3}, 10^{-4}$  and  $10^{-5}$  mile displacement between  $Y_0$  and  $Y'_0$ ), respectively. The location server's best guess is where it uses the maximum number of bits (i.e.,  $p = 17$  and  $|Y_0 - Y'_0| \simeq 1\text{meter}$ ) which results in a look-up table less than 10% similar to  $DB$ . Note that the location server does not even know which H-values belong to the above 10% subset of  $DB$  and thus even by getting very close to real curve parameters, the key cannot be compromised.

Similar to the above case, we now assume that the malicious location server knows the exact values of  $X_0, Y_0, \theta$  and  $N$  and should only approximate the value of  $\Gamma$  with  $\Gamma'$ . Taking the same approach, the location server uses 4, 7, 10 and 14 bits so that the value of  $\frac{\Gamma}{\Gamma'}$  approaches 0.9, 0.99, 0.999 and 0.9999, respectively. Figure 11 (right) shows that in the best case where it uses the maximum number of bits, (i.e.,  $p = 14$ ) the generated look-up table bears less than 5% similarity to  $DB$  again without the location server knowing the subset of points indexed accurately. Therefore the last two sets of experiments demonstrate the strong robustness of our proposed scheme against malicious attacks.

## 8 Conclusion and Future Work

In this paper, we discussed the problem of location privacy in location-based services. We studied the challenges of achieving location privacy and introduced a novel way of blindly evaluating KNN queries, an important class of spatial queries in location-based services, by using one-way space transformations to map objects and query points into an unknown space and evaluate the query in that space. The major contributions of our work can be summarized as follows:

- We proposed blind evaluation of queries using Hilbert curves as space encoders and introduced *DCQR*, our proposed Dual Curve Query Resolution approach and designed an  $O(K \times \frac{2^{2N}}{n})$  computation and  $O(K)$  communication algorithm which enables *DCQR* to resolve KNN queries in the transformed space (where  $n$  is the total number of POI's and  $N$ , the curve order, is a small constant).



- We introduced two new privacy metrics, *u-anonymity* and *a-anonymity*, which are much stronger and more generalized than the privacy constraints of commonly used *K*-anonymity and spatial cloaking based approaches.
- We analytically proved the one-wayness property of our space encoding technique and showed how *DCQR* achieved the result set anonymity as well as *u-anonymity* and *a-anonymity* metrics to become privacy-aware.
- We studied a set of powerful attacks based on the number of bits used to encode the space and empirically evaluated the resilience of *DCQR* against these attacks.
- We conducted extensive experiments to show the superior properties of our blind KNN query resolution scheme.

We intend to study other space mappings and identify new KNN-invariant transformations and propose efficient ways of turning them into space encoders allowing exact answers to be generated for KNN queries while still respecting the location privacy constraints discussed in this paper.

## References

1. Beresford, A.R., Stajano, F.: Location privacy in pervasive computing. *IEEE Pervasive Computing* 2(1), 46–55 (2003)
2. 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)
3. Faloutsos, C., Roseman, S.: Fractals for secondary key retrieval. In: *PODS '89: Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 247–252. ACM Press, New York, NY, USA (1989), doi:10.1145/73721.73746
4. Gedik, B., Liu, L.: A customizable *k*-anonymity model for protecting location privacy
5. Gruteser, M., Grunwald, D.: Anonymous usage of location-based services through spatial and temporal cloaking. In: *MobiSys. USENIX* (2003)
6. Gruteser, M., Liu, X.: Protecting privacy in continuous location-tracking applications. *IEEE Security & Privacy* 2(2), 28–34 (2004)
7. Hilbert, D.: Über die stetige abbildung einer linie auf ein flächenstück. *Math. Ann.* 38, 459–460 (1891)
8. Indyk, P., Woodruff, D.P.: Polylogarithmic private approximations and efficient matching. In: *Theory of Cryptography, Third Theory of Cryptography Conference*, pp. 245–264. New York, NY, USA (2006)
9. Jagadish, H.V.: Linear clustering of objects with multiple attributes. In: *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pp. 332–342. ACM Press, Atlantic City, NJ (1990)
10. Jagadish, H.V.: Analysis of the hilbert curve for representing two-dimensional space. *Inf. Process. Lett.* 62(1), 17–22 (1997)
11. Kalnis, P., Ghinita, G., Mouratidis, K., Papadias, D.: Preserving anonymity in location based services. *A Technical Report TRB6/06*, National University of Singapore (2006)
12. Lawder, J.K., King, P.J.H.: Querying multi-dimensional data indexed using the hilbert space-filling curve. *SIGMOD Record* 30(1), 19–24 (2001)
13. Mokbel, M.F.: Towards privacy-aware location-based database servers. In: Barga, R.S., Zhou, X. (eds.) *ICDE Workshops*, p. 93. IEEE Computer Society Press, Los Alamitos (2006)

14. Mokbel, M.F., Chow, C.-Y., Aref, W.G.: The new casper: Query processing for location services without compromising privacy. In: Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, pp. 763–774. ACM Press, New York (2006)
15. Moon, B., Jagadish, H.v., Faloutsos, C., Saltz, J.H.: Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering* 13(1), 124–141 (2001), doi:10.1109/69.908985
16. Pincioli, F., Combi, C., Pozzi, G., Negretto, M., Portoni, L., Invernizzi, G.: A peano hilbert derived algorithm for compression of angiocardiographic images. In: *Computers in Cardiology*, pp. 81–84. IEEE Computer Society Press, Los Alamitos (1991)
17. Sagan, H.: *Space-Filling Curves*. Springer, Heidelberg (1994)
18. Samarati, P., Sweeney, L.: Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression
19. Schroeder, M.R.: *Number Theory in Science and Communication*. Springer, Heidelberg (1984)
20. Stinson, D.R.: *Cryptography, Theory and Practice*. Chapman & Hall/CRC (2002)

# Enabling Private Continuous Queries for Revealed User Locations

Chi-Yin Chow and Mohamed F. Mokbel

Department of Computer Science and Engineering, University of Minnesota  
{cchow,mokbel}@cs.umn.edu

**Abstract.** Existing location-based services provide specialized services to their customers based on the *knowledge* of their exact locations. With untrustworthy servers, location-based services may lead to several privacy threats ranging from worries over employers snooping on their workers' whereabouts to fears of tracking by potential stalkers. While there exist several techniques to preserve location privacy in mobile environments, these techniques are limited as they do not distinguish between location privacy (i.e., a user wants to hide her location) and query privacy (i.e., a user can reveal her location but not her query). This distinction is crucial in many applications where the locations of mobile users are publicly known. In this paper, we go beyond the limitation of existing cloaking algorithms as we propose a new *robust* spatial cloaking technique for *snapshot* and *continuous* location-based queries that clearly distinguishes between location privacy and query privacy. By this distinction, we achieve two main goals: (1) supporting private location-based services to those customers with public locations, and (2) performing spatial cloaking on-demand basis only (i.e., when issuing queries) rather than exhaustively cloaking every single location update. Experimental results show that the robust spatial cloaking algorithm is scalable and efficient while providing anonymity for large numbers of continuous queries without hiding users' locations.

## 1 Introduction

The emergence of the state-of-the-art location-detection devices, e.g., cellular phones, global positioning system (GPS) devices, and radio-frequency identification (RFID) chips, results in a location-dependent information access paradigm, known as *location-based services*. Location-based services provide convenient information access for mobile users who can issue location-based *snapshot* or *continuous* queries to a database server at anytime and anywhere. Examples of *snapshot* queries include “*where is my nearest gas station*” and “*what are the restaurants within one mile of my location*”, while examples of *continuous* queries include “*continuously report my nearest police car*” and “*continuously report the taxis within one mile of my car*”. Although location-based services promise safety and convenience, they threaten the security and privacy of their customers [1, 2, 3, 4]. With untrustworthy servers, an adversary may access sensitive information about individuals based on their issued location-based queries.

For example, an adversary may check a user's habit and interest by knowing the places she seeks. In many real life scenarios, GPS devices have been used in stalking personal locations [5, 6, 7].

To tackle the privacy threats in location-based services, several spatial cloaking algorithms have been proposed for preserving user location privacy (e.g., [8, 9, 10, 11, 12, 13]). The key idea of spatial cloaking algorithms is to blur the exact user location information into a spatial region that satisfies certain privacy requirements. Privacy requirements can be represented in terms of  $k$ -anonymity [14] (i.e., a user location is indistinguishable among  $k$  users) and/or minimum spatial area (i.e., a user location is blurred into a region with a minimum size threshold).

Unfortunately, existing techniques for preserving location privacy have limited applicability as they do not distinguish between *location* and *query* privacy. In many applications, mobile users cannot hide their locations as these locations are publicly known. In other applications, mobile users do not mind to reveal their exact location information; however, they would like to hide the fact that they issue some location-based queries as these queries may reveal their personal interests. So far, none of the existing spatial cloaking algorithms support this distinction between location privacy and query privacy where it is always assumed that users have to hide both their locations and their queries. Examples of applications that call for this distinction between location privacy and query privacy include:

- **Business operation.** A courier business company has to know the locations of its employees to decide which employee is the nearest one to collect a certain package. However, the company is not allowed to keep track of the employees' behavior in terms of their location-based queries. Thus, company employees reveal their location information, but not their query information.
- **Monitoring system.** Monitoring systems (e.g., transportation monitoring) rely on the accuracy of user locations to provide their valuable services. In order to convince users to participate in these systems, certain privacy guarantees should be imposed for users' behavior through preserving the privacy of the users' location-based queries although users' locations will be revealed.
- **Regular working hours.** During daytime, the locations of company employees are publicly known as their office cubes. Yet, these employees still want to maintain their privacy when they issue location-based queries as these queries would reveal their private personal interests.

In this paper, we present a new *robust* spatial cloaking algorithm that clearly distinguishes between *location* privacy and *query* privacy where mobile users can entertain private *snapshot* and *continuous* location-based queries even if their locations are revealed. With this distinction, we achieve two main goals: (1) supporting private location-based services to those customers with public locations, and (2) performing spatial cloaking on-demand basis only (i.e., when issuing queries) rather than exhaustively cloaking every single location update

as in traditional spatial cloaking techniques. The main idea of our *robust* spatial cloaking algorithms is to anonymize the link between user locations and location-based queries in this a way that even the user locations are known, an adversary would not be able to link the user location to the submitted query. This paradigm is a radical shift from almost all of existing location privacy techniques that aim to anonymize the user location itself with the assumption that if an adversary cannot get access to the user location, then the adversary cannot know the user query.

To achieve our goals, we go through three main steps. First, we identify two main adversary attacks, namely, *query sampling* and *query tracking* attacks, that take place in almost all existing location privacy techniques when distinguishing between location and query privacy. Second, we identify two main properties, namely, *k-sharing region* and *memorization*, that if satisfied in any location cloaking technique, the underlying technique will be free of the *query sampling* and *query tracking* attacks. Finally, we present our *robust* algorithm that possesses the *k-sharing region* and *memorization* properties to support private *continuous* location-based queries for users with public location information. In general, the contributions of this paper can be summarized as follows:

- We introduce a novel query privacy notion in which mobile users are either obligated or willing to reveal their locations, yet they do not want to be identified as the issuer of their location-based queries. This privacy notion is relaxed from the widely used notion that considers hiding the user location and user query in one process. Several applications can make use of our new notion to enhance the overall quality of location-based services.
- We show that directly applying existing spatial cloaking techniques to the new query privacy notion would immediately result in two privacy attack models, namely, *query sampling* and *query tracking* attacks, that can be used by adversaries to infer the actual querying users.
- We identify two main properties, namely, *k-sharing region* and *memorization* that if applied to any location cloaking technique, would make it free from the introduced attack models.
- We propose a new *robust* spatial cloaking technique that: (a) distinguishes between location privacy and query privacy, (b) employs the *k-sharing region* and *memorization* properties, and (c) supports *continuous* location-based queries.
- We provide experimental evidence that the robust spatial cloaking algorithm is scalable in terms of supporting large numbers of users and continuous queries, efficient in terms of supporting various user privacy requirements, provides high-quality services without compromising users' query privacy.

The rest of the paper is organized as follows: Section 2 highlights the related work. The underlying system model is presented in Section 3. Section 4 outlines the adversary attacks. Section 5 presents the required properties to avoid the identified adversary attacks. Our proposed robust spatial cloaking technique is presented in Section 6. Section 7 delineates experimental evaluation of our proposed techniques. Finally, Section 8 concludes the paper.

## 2 Related Work

Almost all previous techniques for location privacy do not distinguish between location privacy and query privacy where the main focus is always to preserve the location privacy. Preserving query privacy is done as a by product of preserving the location privacy. For example, if a certain user wants to have her location  $k$ -anonymized, then each query issued by this user will be also  $k$ -anonymized. Unfortunately, these techniques would not work if the location *cannot* be anonymized. In general, existing research efforts for preserving location privacy can be categorized based on three orthogonal dimensions, namely *employed techniques*, *underlying system architecture*, and *user privacy requirements*:

- **Employed techniques.** Based on the underlying employed technique, location privacy techniques can be classified to either: (a) reporting false locations [15, 16] where the main idea is to cheat the server by either generating a set of  $n$  locations in which only one of them is true [15] or aligning the actual location to the nearest prescribed landmark location [16], or (b) spatial cloaking [8, 9, 10, 11, 12, 13, 17, 18] where the main idea is to blur user locations into spatial regions that satisfy certain privacy requirements. In this paper, we focus on spatial cloaking techniques as they are more efficient, accurate, and most commonly used than false location techniques.
- **Underlying system architecture.** Based on the underlying system architecture, location privacy techniques utilize either: (a) a centralized architecture in which a third trusted party is responsible in cloaking the locations of mobile users [8, 9, 10, 11, 12, 13], or (b) a peer-to-peer architecture in which mobile users collaborate with other peers to find cloaked spatial regions [18, 19, 20]. In this paper, we do not have any assumption for the underlying system architecture as our proposed techniques can be applied to both centralized and peer-to-peer architecture.
- **User privacy requirements.** Based on user privacy requirements, location privacy techniques consider at least one of two main privacy requirements: (a)  $k$ -anonymity in which the user wants to be indistinguishable among  $k$  users [9, 10, 11, 12, 13, 18], and (b) minimum area in which the user wants to have a blurred region with an area size at least  $A_{min}$  [8, 12, 18]. Our proposed techniques in this paper support both of these requirements.

In terms of *continuous* queries, existing research efforts for location privacy techniques (e.g., see [8, 9, 10, 12, 13, 15, 16, 17, 18]) focus only on the case of *snapshot* queries with no direct extension to the case of *continuous* queries.

Our proposed *robust* location privacy technique distinguishes itself from all previous techniques in the following: (1) It distinguishes between location privacy and query privacy, thus it can still provide anonymity to location-based queries even if the user locations are known, (2) It preserves the privacy of *continuous* queries as well as *snapshot* queries, (3) It does not hold any assumption about the system architecture as it can work for both centralized and peer-to-peer architecture.

### 3 System Architecture

Figure 1 depicts the underlying system architecture that can employ our techniques. In general, the system architecture includes three main entities, *mobile users*, *spatial cloaking techniques*, and back-end *database server*.

#### 3.1 Mobile Users

To distinguish between location privacy and query privacy, mobile users register in the system with a privacy profile  $(k_l, k_q)$ , where  $k_l$  indicates that the user wants her *location* to be  $k_l$ -anonymous, i.e., the *cloaked* region for user *location* should contain at least  $k_l$  users, while  $k_q$  indicates that the user wants her *query* to be  $k_q$ -anonymous, i.e., the *cloaked* region for user *query* can be reported by at least  $k_q$  users (if all these users want to issue queries). Based on the values of  $k_l$  and  $k_q$ , we distinguish between two privacy modes:

- *Public location with private query* ( $k_l = 1, k_q > 1$ ). In this mode, users are willing or obligated to reveal their locations ( $k_l = 1$ ), yet they do not want adversaries to link their locations to the queries they issue ( $k_q > 1$ ). Thus, the user location is simply sent to a database server without any perturbation processing while the location information of queries is *cloaked* into spatial regions that satisfy the query privacy requirement before forwarding them to the database server.
- *Private location with private query* ( $k_l > 1, k_q > 1, k_q \geq k_l$ ). In this mode, users want to hide both their locations and query information. However, users have the luxury to request different privacy requirements for their locations and queries. In this case, both user locations and queries are blurred into spatial regions according to the privacy requirements before sending them to the database server.

It is important to note that in all cases,  $k_q \geq k_l$ ; otherwise, a user query would degrade the degree of privacy protection of the user location. For example, consider the extreme case of a user with  $k_q = 1$ , even this user has very high

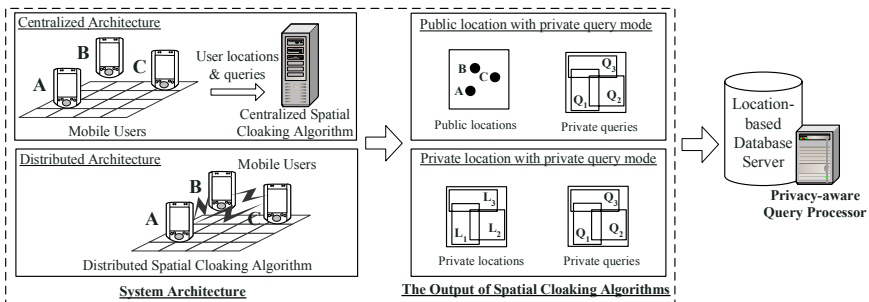


Fig. 1. System architecture and privacy modes

$k_l$ , whenever the user issues a query, the user will be the only person within the query region, thus her personal location can be immediately revealed. Another thing to note is that all existing spatial cloaking techniques implicitly consider only the case of  $k_l = k_q$ , where no distinction is made between location privacy and query privacy.

For simplicity, we do not include the minimum area requirements for the spatial cloaked region. As have been proposed in the literature, integrating the minimum area requirements can be done simply by aligning the cloaked area from the  $k$ -anonymity requirement to a grid area that satisfies the minimum area requirements [12, 18].

### 3.2 Spatial Cloaking Techniques

At the core of the system, spatial cloaking techniques are employed to blur the user locations and queries into spatial regions that satisfy each user profile. Our *robust* spatial cloaking techniques can be incorporated into either centralized or distributed architecture. In the centralized architecture (top left corner of Figure 1), a third trusted party is employed to blur the user locations and/or queries into cloaked spatial regions while in the distributed architecture (bottom left corner of Figure 1), system users employ a peer-to-peer spatial cloaking algorithm to blur their locations and/or queries into cloaked regions. Regardless of the architecture, the output of the spatial cloaking algorithm has two sets. The first output set is either a set of exact point locations in case of public location mode,  $k_l = 1$  (represented as black dots  $A$ ,  $B$ , and  $C$  in Figure 1) or a set of *cloaked* location regions in case of private location mode,  $k_l > 1$  (represented as regions  $L_1$ ,  $L_2$ , and  $L_3$  in Figure 1). The second output set is spatial query regions for the query privacy requirements  $k_q \geq k_l$  (represented as query rectangles  $Q_1$ ,  $Q_2$ , and  $Q_3$  in Figure 1).

### 3.3 Database Server

At the back-end of the system, a *privacy-aware query processor* is embedded inside the location-based database server in order to tune its functionalities to deal with cloaked spatial regions for user locations and user queries rather than exact point locations. The details of the privacy-aware query processor is beyond the scope of this paper where it has been well studied in [12, 21].

## 4 Privacy Leakage in Spatial Cloaking Techniques

This section presents two privacy attack models that can be used by adversaries to link users with revealed locations to their queries. The first attack, *query sampling*, is applicable for *snapshot* queries while the second attack, *query tracking*, is applicable for *continuous* queries. For these two attacks, we briefly discuss the applicability of the following spatial cloaking algorithms: the adaptive interval cloaking [10], CliqueCloak [9],  $k$ -area cloaking [11], Casper [12], hilbASR [13], nnASR [13], and the uncertainty cloaking [8].



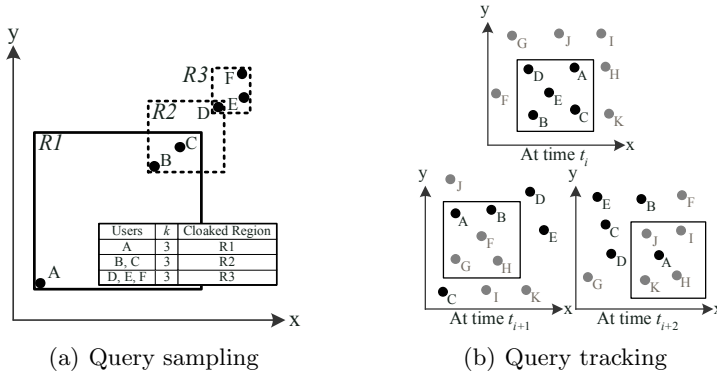


Fig. 2. Privacy attack models

### 4.1 Query Sampling Attacks

In many cases, the location distribution of users is not uniform, e.g., there are many users in a shopping mall (a dense area), but there are only a few users in a small cafe near the mall (a sparse area). Thus, the users located in sparse areas become outliers in the system [13]. As a result of this non-uniform user location distribution, most of existing spatial cloaking algorithms tend to generate larger cloaked spatial regions for the users in sparse areas than that of users in dense areas. If the user location information in a sparse area is known, then, using the *query sampling* attack, an adversary can link this location to a certain query.

**Attack scenario.** Figure 2(a) gives an example of the *query sampling* attack where there are six users  $A, B, C, D, E,$  and  $F$ . Since almost most of existing spatial cloaking techniques do not distinguish between location privacy and query privacy, users can provide only one value for  $k$ -anonymity ( $k=3$ ). The cloaking result is that user  $A$  has  $R_1$  as its cloaked region, users  $B$  and  $C$  have  $R_2$  as their cloaked region, while users  $D, E,$  and  $F$  have  $R_3$  as their cloaked region. In case that user locations are publicly known, an adversary can see that user  $A$  is an outlier to the system. Then, from the cloaked region  $R_1$ , the adversary can infer that the query is sent by user  $A$  located in the sparse area. The main idea is that if the query has been issued by any other user, the cloaked spatial region must first cover the surrounding users in the dense area to generate a smaller cloaked spatial region. As a result, given the knowledge of the user locations, an adversary can link location-based queries to their users.

**Analysis.** With the exception of CliqueCloak [9] and hilbASR [13], all other spatial cloaking techniques suffer from the *query sampling* attack. For example, the techniques that rely on  $k$ -anonymity (the adaptive interval cloaking [10], Casper [12], and nnASR [13]) would simply result in a scenario similar to that of Figure 2(a). On the other hand, spatial cloaking techniques that rely only on a spatial area (e.g.,  $k$ -area cloaking [11] and uncertainty cloaking [8]) may end up to the case where only one user is located at the cloaked spatial region. Given

the public knowledge of the locations of these users, it would be trivial to link a query to its issuer. Both the CliqueCloak algorithm [9] and hilbASR [13] are free from the *query sampling* attack as these algorithms ensure that a cloaked spatial region  $R$  contains at least  $k$  users and all these  $k$  users report  $R$  as their cloaked regions. However, the CliqueCloak algorithm suffers from high computational cost as it can support only  $k$ -anonymity up to  $k = 10$  [9] while the static version of hilbASR lacks flexibility in supporting various  $k$ -anonymous requirements [13].

## 4.2 Query Tracking Attacks

For continuous queries, mobile users have to continuously report their location information to a database server. Although the location information of a query is cloaked as regions, an adversary could link consecutive time snapshots together to identify the query issuer.

**Attack scenario.** Figure 2(b) gives an example of the *query tracking* attack where there are eleven mobile users  $A$  to  $K$ . At time  $t_i$ , user  $A$  issues a five-anonymous continuous query. Cloaking algorithms would give an area that contains users  $A, B, C, D$ , and  $E$ . Assuming uniform distribution, an adversary can only guess that this query is coming from any of these five users within the query area. At time  $t_{i+1}$ , mobile users change their locations while the five-anonymous continuous query is still running. An adversary can see that currently the continuous query area contains users  $A, B, F, G$ , and  $H$ . By linking the snapshots of the continuous query at time  $t_i$  and  $t_{i+1}$ , the adversary can guess that the query issuer is either  $A$  or  $B$  as they are the only common users between these two snapshots. Similarly at time  $t_{i+2}$ , the adversary can conclude that  $A$  is the user query issuer as  $A$  is the only common user within the query area for all three consecutive snapshots.

**Analysis.** Since all of our studied cloaking techniques focus only on the case of *snapshot* queries, these algorithms will suffer from the *query tracking* attack.

## 5 Privacy-Preserving Properties

In this section, we identify two main general properties, namely, *k-sharing region* and *memorization*, that if employed by any spatial cloaking technique, the cloaking technique will be free from *query sampling* and *query tracking* attacks:

- ***k-sharing region.*** Employing the *k-sharing region* property would directly eliminate the *query sampling* attack. The main idea of the *k-sharing region* property is to define a more restrictive  $k$ -anonymity requirement: *A cloaked spatial region not only contains at least  $k$  users, but the region is also shared by at least  $k$  of these users.* Figure 3(a) depicts the result of applying the *k-sharing region* property to the example given in Figure 2(a). Each cloaked region  $R_1$  and  $R_2$  is reported and shared by at least three users. Thus, with the knowledge of user locations and regardless of the user distribution in the space, an adversary cannot link a query to a certain user.

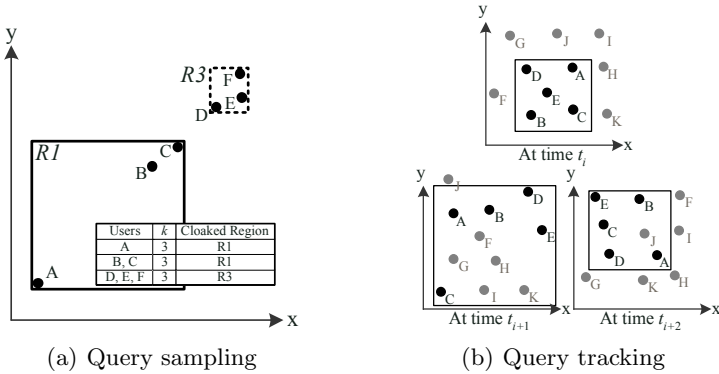


Fig. 3. Solutions for privacy attack models

- **Memorization.** Employing the *memorization* property would directly eliminate the *query tracking* attack. The main idea of the *memorization* property is that the spatial cloaking algorithm has to memorize the users who are contained in the cloaked spatial region of a continuous query at the time when the query is initially issued. Then, with each snapshot of the query, the spatial cloaking algorithm should make sure that these initial users are still within the query cloaked area. Figure 3(b) depicts the result of applying the *memorization* property to the example given in Figure 2(b). The cloaked query regions at all instances of the continuous query at time  $t_i$ ,  $t_{i+1}$ , and  $t_{i+2}$  include the five users  $A$ ,  $B$ ,  $C$ ,  $D$ , and  $E$ . Thus, an adversary cannot narrow down his search to less than the five original users.

It is important to note that both the *k-sharing region* and *memorization* properties are algorithm-independent. So, any spatial cloaking algorithm that has these properties is free from the *query sampling* and *query tracking* attacks.

## 6 Robust Spatial Cloaking Algorithm

This section presents our *robust* spatial cloaking algorithm that distinguishes between location privacy and query privacy while supporting continuous queries. The main idea is to *group* a set of mobile users together such that the cloaked query region for each mobile user in a *group*  $G$  is the spatial region that includes all users in  $G$ . The rest of this section is organized as follows: Section 6.1 introduces the *dynamic group* concept which is the main underlying idea of our proposed *robust* spatial cloaking algorithm. Section 6.2 discusses the algorithm details. Section 6.3 depicts that the proposed algorithm satisfies both the *k-sharing region* and *memorization* properties.

### 6.1 Dynamic Group Concept

The main idea of the *dynamic group* concept is to group users together based on their privacy requirements where each group of users has at least one user

currently issuing a location-based query. Formally, a group of users should have the following three properties:

1. The number of users in a group is equal to or larger than the most restrictive  $k$ -anonymity query requirement among all *querying* users in the group.
2. All users in the same group report the same cloaked spatial region as their cloaked query regions. This spatial region is the minimum region (aligned to some grid) that includes all users belong to that group.
3. For each group, if there are more than one user issuing the same query, the query is only registered once with the database server.

In general, a user is not allowed to issue a snapshot or continuous query unless the user belongs to a certain *group*. Users may leave their groups once their queries are terminated. Similarly, users may join a new group whenever they want to issue new queries. In the same time, users may be added to or removed from some groups to help other users form a cloaked query area. A user can be either in a *grouped* or *ungrouped* state. Initially, all users are in the *ungrouped* state. Whenever a user joins an existing group or form a new group, the user becomes in the *grouped* state. Only *grouped* users are allowed to issue location-based queries. Each user maintains a tuple  $U = (id, L, K_l, K_q, \mathcal{Q}, G)$ , where  $id$  is a unique user identifier,  $L$  is the user's current location,  $K_l$  and  $K_q$  are the location anonymity and query anonymity privacy requirements, respectively,  $\mathcal{Q}$  is a set of queries sent by the user, and  $G$  is the identifier of the group where the user is assigned to. Setting  $G$  to *null* indicates that the user is in the *ungrouped* state. For each group, we maintain a tuple  $G = (id, \mathcal{M}, R, K, \mathcal{Q})$ , where  $id$  is a unique group identifier,  $\mathcal{M}$  is a set of users assigned to  $G$ ,  $K$  is the most restrictive  $k$ -anonymity query privacy requirement of all users assigned to  $G$ ,  $R$  is the cloaked spatial region of  $G$ , and  $\mathcal{Q}$  is a set of queries issued by at least one user assigned to  $G$ . For each query, we maintain a tuple  $Q = (id, S_l, S_c)$ , where  $id$  is a unique query identifier (all queries with the same content have the same  $id$ ),  $S_l$  memorizes the set of members when the query is issued by a member, and  $S_c$  is the set of current members that were included in  $S_l$ .

## 6.2 Algorithm

Our *robust* spatial cloaking algorithm has four main modules: (1) *Query registration* which is called whenever a user wants to issue a snapshot or continuous location-based query, (2) *Query termination* which is called whenever a user wants to terminate its previously issued query, (3) *Group join* which is called from the *query registration* module to find the most suitable group for the querying user, and (4) *Group leave* which is called by the user to act as a cleanup process whenever the user wants to disconnect from the system. Details of these modules as follow:

**Query registration.** Algorithm [1](#) depicts the pseudo code of the query registration module. This module is called only when a user issues a snapshot or continuous location-based query. The goal of *query registration* is to find a suitable group

**Algorithm 1.** Robust Spatial Cloaking: Issue a Query

---

```

1: procedure QUERYREGISTRATION (Query  $Q$ , User  $U$ )
2: if  $U.G = null$  then
3:   GROUPJOIN( $Q$ ,  $U$ ) (See Algorithm 3)
4: else
5:    $G \leftarrow U.G$ 
6:   if  $|G.M| > U.K_q$  then
7:      $Q.S_l \leftarrow G.M$ ;  $Q.S_c \leftarrow G.M$ 
8:     if  $Q \notin G.Q$  then
9:        $G.Q \leftarrow G.Q \cup \{Q\}$ 
10:      Send  $Q$  to the database server as cloaked region  $G.R$ 
11:     end if
12:   else
13:     GROUPLEAVE( $U$ )
14:     GROUPJOIN( $Q$ ,  $U$ ) (See Algorithm 3)
15:   end if
16: end if

```

---

$G$  that matches the querying user location and the query privacy requirements. Then, the cloaked region for the issued query is the minimum spatial region  $R$  that includes all users in  $G$ . To avoid having mobile users lying on the cloaked area boundary, the minimum spatial region  $R$  is aligned to a certain grid. The *query registration* module starts by checking the status of the querying user (Line 2 in Algorithm 1). If the querying user is *ungrouped*, i.e., does not belong to any current group ( $U.G = null$ ), then we call the *group join* module to find the suitable group for the user (Line 3 in Algorithm 1). Then, the algorithm terminates as the query cloaked region would be computed from the group that the user will join. On the other side, if the querying user is already in the *grouped* state (i.e., belongs to a group  $G$ ), we check if the current user group does satisfy the user query privacy requirement, i.e., the number of users within  $G$  ( $|G.M|$ ) is equal to or greater than the user query anonymity ( $U.K_q$ ) (Line 6 in Algorithm 1). If this is the case, we set the query's  $S_l$  and  $S_c$  to the users within  $G$  ( $G.M$ ) (Line 7 in Algorithm 1). Then, we check if the user query is already registered by some other users in the same group. If this is the case, we do nothing as the current user can share the query answer with other users. However, if the issued query is a new one, we add it to the current outstanding queries of  $G$  and send it as cloaked region to the database server (Lines 8 to 11 in Algorithm 1). Finally, if the current user group  $G$  does not satisfy the user query privacy requirements, the user has to leave  $G$  and join another group that would be more suitable to the query privacy requirement (Lines 13 to 14 in Algorithm 1). In this case, the query cloaked region will be produced from the new group that the user will join.

**Query termination.** Algorithm 2 depicts the pseudo code of the query termination module. This module is called when a user decides to terminate its outstanding continuous query or when the result of the snapshot query is received. The main idea of the algorithm is to update the user and group information with respect to the terminated query, and unregister the query if there are no other group members that are interested in this query. This process is done in two phases. In the first phase, we clean the group information while in the second phase we clean the user information. For the first phase, we start by checking if

**Algorithm 2.** Robust Spatial Cloaking: Terminate a Query

---

```

1: procedure QUERYTERMINATION (Query  $Q$ , User  $U$ )
2:  $G \leftarrow U.G$ 
3: if no other querying users in  $G$  are interested in  $Q$  then
4:   Wait until  $|S_c| - |S_l| \geq k$ , unregister  $Q$  with the database server;  $G.Q \leftarrow G.Q - \{Q\}$ 
5:   if  $G.Q$  is empty then Annihilate  $G$ , and mark all users in  $G$  as ungrouped; return;
6: end if
7:  $U.Q \leftarrow U.Q - \{Q\}$ 
8: if  $U.Q$  is empty then
9:    $G.K \leftarrow \max(\forall U_i.K_q, U_i \in G.M \wedge U_i.Q \neq \{\emptyset\})$ ;
10:  if  $|G.M| > G.K$  then
11:    Determine a centroid of all querying users in  $G$ 
12:    Remove  $|G.M| - G.K$  non-querying members that are furthest away from the centroid
13:  end if
14: end if

```

---

there are any other group members in  $G$  who are interested in the terminated query  $Q$  (Line 3 in Algorithm 2). If this is not the case, we remove  $Q$  from the list of outstanding queries in  $G$ . Also, we have to unregister  $Q$  from the database server. To do this process safely, we wait until at least  $k$  users in  $Q$ 's  $S_l$  have left  $G$  before we can safely unregister  $Q$  from the server. The key idea of suspending query termination is to mix the query termination event with at least  $k$  related group removal events, in order to avoid an adversary linking the query termination event to a particular user with a probability higher than  $1/k$  (Line 4 in Algorithm 2). After terminating  $Q$ , if there are no more querying users in  $G$ , we annihilate the group  $G$  by marking all group members as *ungrouped* while updating their tuples accordingly (Line 5 in Algorithm 2). The second phase (cleaning user information) is invoked only if group  $G$  is still outstanding. In this phase, we start by removing the terminated query  $Q$  from the list of outstanding queries associated with the user  $U$  (Line 7 in Algorithm 2). Then, we update the group privacy information  $G.K$  to be the current maximum privacy requirements of all querying users within  $G$  (Line 9 in Algorithm 2). Since, we are updating the maximum group privacy requirement  $G.K$ , we may end up in having  $G.K$  less than the number of current user in  $G$  ( $|G.M|$ ). In this case,  $G$  is considered to have additional  $|G.M| - G.K$  users than what it needs. So, we aim to release all these additional group members as this would mainly reduce the group region area  $G.R$  and in the same time allow released users to either form new groups or join other existing groups that could be more suitable to their privacy requirements. To do this process, we remove the  $|G.M| - G.K$  non-querying members that are furthest away from the centroid of all querying users in the group. (Lines 11 to 12 in Algorithm 2). The key idea of using the centroid of all querying members is to minimize the group region area  $G.R$  with respect to querying members. Minimizing a group region area would result in better accuracy in the query answer reported from the database server.

**Group join.** Algorithm 3 depicts the pseudo code of the group join operation. The key idea of this module is to find a group for a user that is suitable to the user query privacy requirement. We start by finding a set of groups  $\mathcal{G}$  covering the user location, and then sort them by their group region area in an increasing order (Lines 3 to 4 in Algorithm 3). The main idea of sorting based on the area

**Algorithm 3.** Robust Spatial Cloaking: Group Join

---

```

1: procedure GROUPJOIN (Query  $Q$ , User  $U$ )
2:  $\mathcal{G}' \leftarrow \{\emptyset\}$ ;
3:  $\mathcal{G} \leftarrow$  all existing groups  $G$  that cover the user location, i.e.,  $U.L \in G.R$ 
4: Sort  $\mathcal{G}$  by the area of  $G.R$  in an increasing order
5: for each group  $G$  in  $\mathcal{G}$  do
6:   if  $|G.M| + 1 \geq U.K_q$  then
7:      $\mathcal{G}' \leftarrow \mathcal{G}' \cup \{G\}$ 
8:     if  $Q \in G.Q$  then  $G.M \leftarrow G.M \cup \{U.id\}$ ;  $Q.S_c \leftarrow G.M$ ;  $Q.S_l \leftarrow G.M$  return
9:   end if
10: end for
11: if  $\mathcal{G}' \neq \text{null}$  then
12:    $G \leftarrow$  the first group in  $\mathcal{G}'$ 
13:    $G.M \leftarrow G.M \cup \{U.id\}$ ;  $Q.S_c \leftarrow G.M$ ;  $Q.S_l \leftarrow G.M$ ;  $G.Q \leftarrow G.Q \cup \{Q\}$ 
14:   Send the query in  $U.Q$  to the database server as cloaked region  $G.R$ 
15: else
16:   if the number of ungrouped users  $< U.K_q$  then
17:     Suspend the request for a certain period of time
18:     Go to Line 2
19:   end if
20:   Construct a new group  $G$ 
21:    $U.G \leftarrow G$ ;  $G.M \leftarrow \{U.id\}$ ;  $Q.S_c \leftarrow G.M$ ;  $Q.S_l \leftarrow G.M$ ;  $G.Q \leftarrow \{Q\}$ ;  $G.K \leftarrow U.K_q$ 
22:   Add  $G.K$  ungrouped users that are closest to  $U.L$  into  $G$ 
23:   Send the query in  $U.Q$  to the database server as cloaked region  $G.R$ 
24: end if

```

---

is to give preference to those groups with minimum region area  $G.R$ . Then, we join the user to a group  $G$  based on following prioritized cases with the first one is the highest priority while the last one is the lowest priority:

1. If there is a group  $G \in \mathcal{G}$  satisfying the user's query privacy requirement, i.e.,  $|G.M| + 1 \geq U.K_q$  and the user's required query  $Q$  has already registered in  $G$ , i.e.,  $Q \in G.Q$ , we simply assign the user  $U$  to  $G$ , and set  $Q$ 's  $S_l$  and  $S_c$  to the users within  $G$  ( $G.M$ ). Notice that, due to the pre-sorting step, if there are several groups with this property, we pick the group  $G$  with the minimum region area  $G.R$  (Lines 5 to 10 in Algorithm 3).
2. If there is a group  $G \in \mathcal{G}$  satisfying the user's query privacy requirement, i.e.,  $|G.M| + 1 \geq U.K_q$  but does not have the query  $Q$  among its query list, we assign  $U$  to  $G$ . In this case, we would need to register  $Q$  with  $G$  and send  $Q$  to the database server as the cloaked region  $G.R$ . Notice that if there multiple groups  $G$ , we would select the one with the minimum area  $G.R$  (Lines 5 to 10 in Algorithm 3).
3. Otherwise, we check whether there are enough number of *ungrouped* users to construct a new group for the user. In case that the number of *ungrouped* users is less than the user's query privacy requirement, we suspend the request a prescribed period of time, and then it restarts (Lines 17 to 18 in Algorithm 3). On the other side, if we are able to construct a new group  $G$  for the user  $U$ , we add  $G.K$  (that is equal to  $U.K_q$ ) *ungrouped* users that are closest to the user's location to  $G$ , in order to satisfy the user's query privacy requirement (Lines 20 to 22 in Algorithm 3). The reason of adding nearby *ungrouped* users to  $G$  is to minimize the group region size. Finally, the algorithm registers the query with a database server with the cloaked region  $G.R$  from by the locations of the new group members.

**Algorithm 4.** Robust Spatial Cloaking: Group Leave

---

```

1: function GROUPLAIVE (User  $U$ )
2: //  $U$  is a tuple of a leaving user
3:  $G \leftarrow U.G$ 
4: if  $U.Q$  is not empty then
5:   for each  $Q \in U.Q$  do
6:     QUERYTERMINATION ( $Q, U$ )
7:   end for
8: end if
9: for each  $Q \in G.Q$ ; if  $U.id \in Q.S_c$  then  $Q.S_c \leftarrow Q.S_c - \{U.id\}$ 
10:  $U.G \leftarrow \text{null}$ ;  $G.M \leftarrow G.M - \{U.id\}$ ;
11: if  $|G.M| < G.K$  then
12:   Determine a centroid of all querying users in  $G$ 
13:   Add one ungrouped users that is closest to the centroid into  $G$ 
14: end if

```

---

**Group leave.** Algorithm 4 depicts the pseudo code of the group leave module. This module is executed when a user  $U$  decides to leave the system. The first thing to do when is to go through all outstanding queries of  $U$  and terminate them one by one (Lines 4 to 7 in Algorithm 4). If the user is included in some query memorization sets, the user is removed from these sets (Line 9 in Algorithm 4). Then, we set the user group to *null* and remove the user from its current group. It may happen that removing this user would reduce the number of users in the group ( $|G.M|$ ) to be less than the most restrictive query privacy requirement ( $G.K$ ). If this is the case, we add another *ungrouped* user, if possible, that is the closest to the centroid of querying users (Lines 12 to 13 in Algorithm 4).

### 6.3 Correctness

In this section, we depict that the robust spatial cloaking algorithm has the *k-sharing region* and *memorization* privacy-preserving properties, and thus is free from *query sampling* and *query tracking* privacy attacks.

**The robust spatial cloaking algorithm is free from query sampling attacks.** For each group  $G$ , the number of users in  $G$  ( $|G.M|$ ) is guaranteed to satisfy the most restrictive query privacy requirement among the querying members. As depicted in Line 22 in Algorithm 3 and Line 13 in Algorithm 4, whenever the number of users becomes less than the most restrictive query privacy requirement, we immediately add more users to the group. Thus, the group region satisfies the query privacy requirement of all querying members. Since we only report the group region  $G.R$  as the location information of all snapshot or outstanding continuous queries, so the group region is always shared by all group members. Therefore, the algorithm possesses the *k-sharing region* privacy-preserving property.

**The robust spatial cloaking algorithm is free from query tracking attacks.** We will consider four cases, *no member admission or removal*, *non-querying member admission*, *querying member admission*, and *member removal*. Let  $S$  be the set of members located within the group region  $G.R$  at the time when a query  $Q$  is initially registered with a database server. First, if there is no



member admission or removal, the subsequent  $G.R$  must contain all members in  $S$ . Thus, the algorithm has the memorization privacy-preserving property. Second, a non-querying member admission does not affect the memorization property, because  $G.R$  must still contain all members in  $S$ . Third, a querying member admission also does not affect this property, since this member has been located within  $G.R$  for some time, i.e., we do not expand  $G.R$  to include any new members. Thus, an adversary cannot link the newly issued query to a particular user within  $G.R$ . Fourth, for each registered query, the query is only unregistered with a database server after at least  $k$  candidate issuers have left  $G$ . Thus, an adversary cannot link the query to these candidate issuers with a probability higher than  $1/k$ . Therefore, the algorithm possesses the *memorization* privacy-preserving property.

## 7 Experimental Result

In this section, we experimentally evaluate the robust spatial cloaking algorithm, in terms of scalability and privacy requirements. In all experiments of this section, we use the Network-based Generator of Moving Objects [22] to generate a set of moving objects. The input to the generator is the road map of Hennepin County in Minnesota, USA. The output of the generator is a set of moving objects that move on the road network of the given map. We consider continuous nearest-neighbor queries for a randomly selected object type, i.e., “continuously report my nearest *object*”. Furthermore, we consider three performance metrics, *cloaked region area*, *number of continuous queries*, and *number of groups*. The *cloaked region area* metric is defined as the ratio of the average area of the cloaked spatial region reported to a database server to the entire system area, while the *number of continuous queries* metric is defined as the total number of query registration. The *number of groups* metric is the total number of groups created by the robust spatial cloaking algorithm. In all experiments, 10% of system users issue continuous queries.

### 7.1 Scalability

Figures 4 and 5 give the scalability of the robust spatial cloaking algorithm with increasing the number of users from 10K to 50K (with three different levels of  $k$ -anonymity for their queries [1, 10], [10, 50], and [50, 100]), and the number of object types from 10 to 100 (with different numbers of users from 20K to 50K), respectively. In this experiment, there are 50 object types. Figure 4(a) depicts that the cloaked spatial region area reduces with increasing the number of users. When there are more users, the user density is higher, so each group generally has smaller region area. With more querying users, the number of groups and registered continuous queries increases, as depicted in Figures 4(b) and 4(c), respectively. Furthermore, the cloaked region area gets larger, when the users have more restrictive query privacy requirements (Figure 4(a)). This is because we need to assign more users to a group, in order to satisfy the querying user’s privacy requirement. Figures 4(b) and 4(c) depict that there is less number of groups and continuous

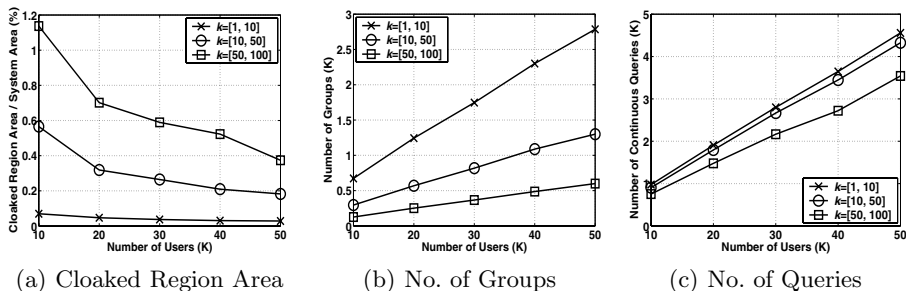


Fig. 4. Number of users

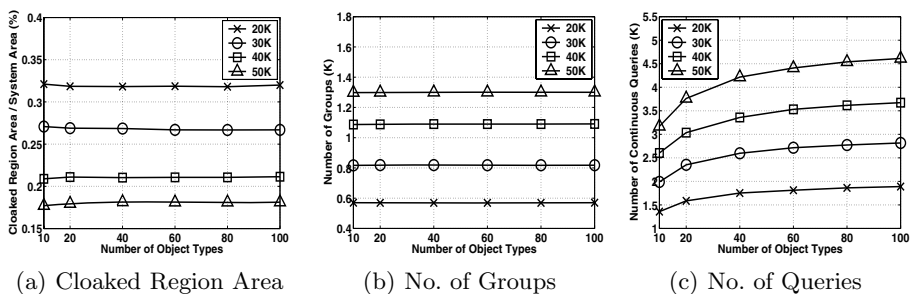


Fig. 5. Number of object types

queries, respectively, as the users have more restrictive query privacy requirement. With more restrictive privacy requirement, there are more users in a group that leads to a higher chance for them to share query answer.

Figure 5 depicts the performance of our robust spatial cloaking algorithm with respect to various number of object types. In this experiment, the query  $k$ -anonymity requirement is between 10 and 50. Figures 5(a) and 5(b) give that the cloaked region area and number of groups are only slightly affected by increasing the number of object types. However, the number of registered continuous queries rises with more different object types (Figure 5(c)). This is due to the fact that if there are more different object types, there is a higher chance for the users issuing continuous queries for distinct object types in a group. As a result, each group has to register more continuous queries with a database server with increasing the number of object types.

## 7.2 Effect of Query Privacy Requirement

In this experiment, we increase the  $k$ -anonymity query privacy requirement  $K_q$  from 10 to 160 with various number of users from 20K to 50K, and the number of object types is 50. With more restrictive query privacy requirements, more users are assigned to each group, so the group region area increases (Figure 6(a)). When the group region area gets larger, there is a higher chance for a querying

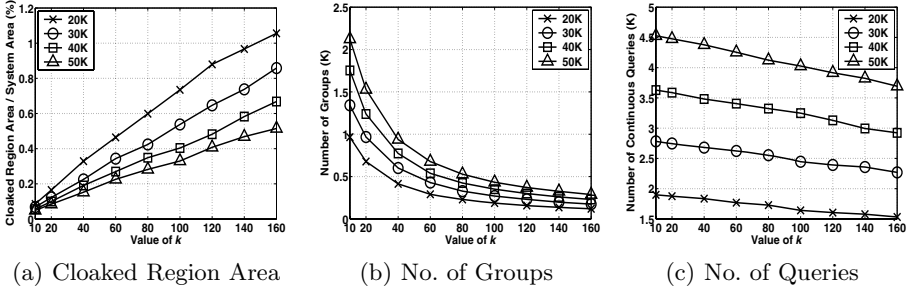


Fig. 6.  $k$ -anonymity query privacy requirement

user joining an existing group. Thus, the number of groups reduces with increasing the value of  $k$ , as depicted in Figure 6(b). With more querying users in a group, more users are interested in the same object type, i.e., they can share the query answer; and therefore, the number of registered continuous queries drops when the query privacy requirement gets more restrictive (Figure 6(c)).

## 8 Conclusion

In this paper, we have introduced a new privacy notion in which mobile users can protect their query privacy even if their locations are revealed. This privacy notion is crucial in many applications where users are obligated or willing to reveal their locations. We show that with this new privacy notion, existing techniques for preserving the privacy of location-based queries would fail as these techniques do not distinguish between location privacy and query privacy. Namely, we identify two privacy attacks models, *query sampling* and *query tracking* that take place upon distinguishing between location privacy and query privacy. Then, we outline two main properties, namely *k-sharing region* and *memorization* that if satisfied by location privacy techniques would make them resilient to the identified attack. Then, we present a *robust* spatial cloaking technique that: (1) clearly distinguishes between location privacy and query privacy, (2) supports *continuous* and *snapshot* location-based queries, (3) employs both the *k-sharing region* and *memorization* properties, hence, free from the identified attacks. Experimental results show that the robust spatial cloaking algorithm is scalable and efficient in terms of large numbers of mobile users, object types, and various privacy requirements.

## References

- [1] Ackerman, L., Kempf, J., Miki, T.: Wireless Location Privacy: A Report on Law and Policy in the United States, the European Union, and Japan. Technical Report DCL-TR2003-001, DoCoMo Communication Laboratories, USA (2003)
- [2] Barkhuus, L., Dey, A.K.: Location-Based Services for Mobile Telephony: a Study of Users' Privacy Concerns. In: Proceeding of the IFIP Conference on Human-Computer Interaction, INTERACT (2003)

- [3] Beresford, A.R., Stajano, F.: Location Privacy in Pervasive Computing. *IEEE Pervasive Computing* 2(1), 46–55 (2003)
- [4] Warrior, J., McHenry, E., McGee, K.: They Know Where You Are . *IEEE Spectrum* 40(7), 20–25 (2003)
- [5] Foxs News: Man Accused of Stalking Ex-Girlfriend With GPS. (September 4, 2004), <http://www.foxnews.com/story/0,2933,131487,00.html>
- [6] USAToday: Authorities: GPS System Used to Stalk Woman. (December 30, 2002) <http://usatoday.com/tech/news/2002-12-30-gps-stalker.x.htm>
- [7] Voelcker, J.: Stalked by Satellite: An Alarming Rise in GPS-enabled Harassment. *IEEE Spectrum* 47(7), 15–16 (2006)
- [8] Cheng, R., Zhang, Y., Bertino, E., Prabhakar, S.: Preserving User Location Privacy in Mobile Data Management Infrastructures. In: *Proceedings of Privacy Enhancing Technology Workshop* (2006)
- [9] Gedik, B., Liu, L.: Location Privacy in Mobile Systems: A Personalized Anonymization Model. In: *ICDCS* (2005)
- [10] Gruteser, M., Grunwald, D.: Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In: *MobiSys* (2003)
- [11] Gruteser, M., Liu, X.: Protecting Privacy in Continuous Location-Tracking Applications. *IEEE Security and Privacy* 2(2), 28–34 (2004)
- [12] Mokbel, M.F., Chow, C.-Y., Aref, W.G.: The New Casper: Query Processing for Location Services without Compromising Privacy. In: *VLDB* (2006)
- [13] Kalnis, P., Ghinita, G., Mouratidis, K., Papadias, D.: Preserving Anonymity in Location Based Services. Technical Report TRB6/06, Department of Computer Science, National University of Singapore (2006)
- [14] Sweeney, L.:  $k$ -anonymity: A Model for Protecting Privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems* 10(5), 557–570 (2002)
- [15] Hong, J.I., Landay, J.A.: An Architecture for Privacy-Sensitive Ubiquitous Computing. In: *MobiSys* (2004)
- [16] Kido, H., Yanagisawa, Y., Satoh, T.: An Anonymous Communication Technique using Dummies for Location-based Services. In: *ICPS* (2005)
- [17] Duckham, M., Kulik, L.: A Formal Model of Obfuscation and Negotiation for Location Privacy. In: Gellersen, H.-W., Want, R., Schmidt, A. (eds.) *PERVASIVE 2005*. LNCS, vol. 3468, pp. 152–170. Springer, Heidelberg (2005)
- [18] Chow, C.Y., Mokbel, M.F., Liu, X.: A Peer-to-Peer Spatial Cloaking Algorithm for Anonymous Location-based Services. In: *ACM GIS*, ACM Press, New York (2006)
- [19] Ghinita, G., Kalnis, P., Skiadopoulos, S.: PRIVÉ: Anonymous Location-Based Queries in Distributed Mobile Systems. In: *WWW* (to appear, 2007)
- [20] Mokbel, M.F., Chow, C.-Y.: Challenges in Preserving Location Privacy in Peer-to-Peer Environments (Invited paper). In: *Proceedings of the International Workshop on Information Processing over Evolving Networks, WINPEN*. (2006)
- [21] Mokbel, M.F., Chow, C.-Y., Aref, W.G.: The New Casper: A Privacy-Aware Location-based Database Server (Demonstration). In: *ICDE* (2007)
- [22] Brinkhoff, T.: A Framework for Generating Network-Based Moving Objects. *GeoInformatica* 6(2), 153–180 (2002)

# Computing a $k$ -Route over Uncertain Geographical Data

Eliyahu Safra<sup>1</sup>, Yaron Kanza<sup>2</sup>, Nir Dolev<sup>1</sup>, Yehoshua Sagiv<sup>3,\*</sup>, and Yerach Doytsher<sup>1</sup>

<sup>1</sup> Department of Transportation and Geo-Information, Technion, Haifa, Israel  
{safra,dolev,doytsher}@technion.ac.il

<sup>2</sup> Department of Computer Science, University of Toronto, Toronto, Canada  
yaron@cs.toronto.edu

<sup>3</sup> School of Engineering and Computer Science, The Hebrew University, Jerusalem, Israel  
sagiv@cs.huji.ac.il

**Abstract.** An uncertain geo-spatial dataset is a collection of geo-spatial objects that do not represent accurately real-world entities. Each object has a *confidence value* indicating how likely it is for the object to be correct. Uncertain data can be the result of operations such as imprecise integration, incorrect update or inexact querying. A  $k$ -route, over an uncertain geo-spatial dataset, is a path that travels through the geo-spatial objects, starting at a given location and stopping after visiting  $k$  correct objects. A  $k$ -route is considered shortest if the expected length of the route is less than or equal to the expected length of any other  $k$ -route that starts at the given location. This paper introduces the problem of finding a shortest  $k$ -route over an uncertain dataset. Since the problem is a generalization of the traveling salesman problem, it is unlikely to have an efficient solution, *i.e.*, there is no polynomial-time algorithm that solves the problem (unless  $P=NP$ ). Hence, in this work we consider heuristics for the problem. Three methods for computing a short  $k$ -route are presented. The three methods are compared analytically and experimentally. For these three methods, experiments on both synthetic and real-world data show the tradeoff between the quality of the result (*i.e.*, the expected length of the returned route) and the efficiency of the computation.

## 1 Introduction

Spatial datasets store objects that represent real-world geographical entities. When such datasets are *uncertain*, users who see only the information stored in the dataset cannot be sure whether objects correctly represent real-world entities. However, we assume that users can verify the correctness of objects by using additional information or by visiting the geographical locations of these objects. In such datasets, each object has a *correctness value* of either *true* or *false*, and a *confidence value*; yet, users do not know the correctness values. Thus, when querying uncertain datasets, users consider the confidence of an object as the probability that the correctness value of the object is *true*. Applications over uncertain datasets should be able to utilize confidence values.

Some cases in which uncertain datasets occur are integration of heterogeneous sources, incorrect updates and inexact querying. We start by describing the first case.

---

\* This author was supported by The Israel Science Foundation (Grant 893/05).

When integrating two geo-spatial sources, the result consists of pairs and singletons. A pair is correct if it comprises two objects that represent the same real-world entity in the different sources. A singleton (*i.e.*, a set that contains a single object) is correct if it represents a real-world entity that does not have a corresponding object in the other source. In the absence of keys, integration can be done by using object locations [34] or by using locations and additional attributes [17]. However, since locations are inaccurate, it is uncertain whether any given pair of the result is correct, that is, whether its two objects indeed represent the same real-world entity. Thus, the result of the integration is an uncertain spatial dataset.

Incorrect data manipulation can also yield uncertain datasets. The following example illustrates this.

*Example 1.* Consider a dataset of hotels, and suppose that no key constraint is imposed on this dataset. An incorrect insertion of data into the dataset may cause some hotel to appear twice with two different rates. In this case, users cannot know which object shows the correct rate of the hotel. Updates can cause a similar problem, for instance, when the name of some hotel is replaced with a name of a different hotel that already exists in the dataset.

Andritsos et al. [1] showed how to assign confidence values to objects in such cases.

Another important usage of uncertain datasets is representing the result of queries that contain an *imprecise condition*, namely, an adjective instead of a comparison between an attribute and a value. For example, *find good restaurants*, rather than *find restaurants that have a rating of five stars*. Additional examples are *find a luxury hotel*, *find a popular tourist site*, etc. The ability to cope with such queries is important in systems that are designed to answer requests for information, formulated by non-expert users. Such queries are useful when providing tourist and municipal information to laymen who send their request through some limited device, such as a cellular phone. When processing requests that are sent from a mobile device, one should bear in mind that the answer may depend on the location of the user.

Recently, *location-based services* have become a growing and important area in both research and commerce. Location-based services supply information through mobile devices, and the answer to a particular request depends on the location from which the request was sent, *i.e.*, the location of the mobile device [21]. For instance, a user who asks about a nearby restaurant will get different answers when the user location is in Times Square, Manhattan, and in Piccadilly Circle, London.

In this paper, we consider a specific location-based service of finding the *shortest  $k$ -route* over an uncertain dataset. In this application, the input consists of an uncertain geo-spatial dataset, a location and some  $k$ . The output is a route that starts at the given location and goes via objects of the given dataset. The route is such that the expected distance from the starting point till visiting  $k$  correct objects is minimal. The following examples demonstrate the need for providing this service.

*Example 2.* Consider a user located in Times Square, Manhattan, that is looking for an *inexpensive and good restaurant nearby*. The answer to this query can be a list of restaurants that presumably satisfy the request. However, it can also be an uncertain dataset that contains all the restaurants in Manhattan, such that the confidence value of

each restaurant is correlated with the likelihood that the user will consider this restaurant as inexpensive and good. Suppose that the user wants to compare three *good and inexpensive* restaurants before deciding in which one to dine. The user may also want to walk as little as possible when visiting restaurants until she sees three that she likes. In this case, the information system should find a 3-route starting at the location of the user in Times Square and going through restaurants in the dataset in a way that increases the likelihood to visit three inexpensive, good restaurants after a short walk.

There are many other scenarios in which finding the shortest  $k$ -route can be useful. For instance, before leasing or buying a house, it may be reasonable to visit and compare several options, and to do that efficiently means to go through a short route. Also, for planing a tour in some city or in some country, it may be useful to use such an application.

Finding the shortest  $k$ -route can be seen as the spatial version of computing top- $k$  answers to a query. In many information-retrieval systems and also in some database applications, the result of a query contains only the top- $k$  answers to the query. For instance, search engines on the World-Wide Web may present to users only the top 1000 results out of the millions of answers to a query. In geographical applications, answers should not be ranked merely according to how well they match the query. Objects should be returned with a recommended route to take in order to visit them. Moreover, choosing such a route may have an influence on how the objects are ranked in the answer to the user. The shortest  $k$ -route that we propose in this paper is one way of doing it.

The problem of finding the shortest  $k$ -route is a generalization of the traveling salesman problem (TSP). TSP (in the version where the salesman need not return to the origin) is the same as finding the shortest  $k$ -route in the case where there is no uncertainty (*i.e.*, all the objects have confidence equal to one) and  $k$  is equal to the number of objects in the dataset. Since TSP is known to be NP-hard, we do not expect to find an efficient polynomial-time algorithm to the shortest  $k$ -route problem. Thus, we settle for heuristics.

In this paper, we introduce three novel algorithms for finding a short  $k$ -route and we explain the differences between them. We also present the results of extensive experimentation that compares the algorithms on different types of data. Our experiments were conducted on both synthetic data and real-world data.

The main contributions of this paper are as follows.

- We introduce the problem of finding the shortest  $k$ -route over uncertain geo-spatial datasets.
- We present three algorithms for finding a short  $k$ -route and explain the different behaviors of these algorithms.
- We conducted thorough experiments that show the tradeoff between effectiveness of the algorithm (*i.e.*, the ability to compute a path with a short expected length) and its efficiency.

## 2 Framework

In this section, we formally present our framework and the problem of finding a shortest  $k$ -route over uncertain datasets.

**Uncertain Geo-Spatial Datasets.** A *geo-spatial dataset* is a collection of *geo-spatial objects*. Each object has a location and may have additional spatial and non-spatial attributes. Height and shape are examples of spatial attributes. Address and name are examples of non-spatial attributes. We assume that locations are points and objects are disjoint, *i.e.*, different objects have different locations. For objects that are represented by a polygonal shape and do not have a specified point location, we consider the center of mass of the polygonal shape to be the point location. The distance between two objects is the Euclidean distance between their point locations. We denote the distance between two objects  $o_1$  and  $o_2$  by  $distance(o_1, o_2)$ . Similarly, if  $o$  is an object and  $l$  is a location, then  $distance(o, l)$  is the distance from  $o$  to  $l$ .

An uncertain geographical dataset is a pair  $(D, \varphi_c)$ , where  $D$  is a geo-spatial dataset and  $\varphi_c : D \rightarrow [0, 1]$  is a function that maps each object of  $D$  to a value between 0 and 1, called *confidence*. An *instance* of  $(D, \varphi_c)$  is a pair  $(D, \tau)$  where  $\tau : D \rightarrow \{true, false\}$  is a function that maps each object of  $D$  to a *correctness value*, which is either *true* or *false*. An uncertain dataset  $(D, \varphi_c)$  has  $2^{|D|}$  possible instances, where  $|D|$  is the number of objects in  $D$ . We consider the confidence of an objects as an indication of how likely it is for the object to be correct, *i.e.*, to be mapped to *true* by  $\tau$ . To each instance  $I = (D, \tau)$ , we assign a probability  $P(I)$  according to the confidence values of the objects:  $P((D, \tau)) = [\prod_{\{o_i | \tau(o_i)=true\}} \varphi_c(o_i)] \cdot [\prod_{\{o_i | \tau(o_i)=false\}} (1 - \varphi_c(o_i))]$ . When computing a route over an uncertain dataset, the actual instance is not known. Hence, the probabilities of possible instances should be taken into account.

Usually, users know only  $D$  and  $\varphi_c$  when querying or using uncertain data. However, when developing algorithms for uncertain data, it is important to test them on data for which  $\tau$  is known in order to determine the quality of the results of each algorithm. Thus, the datasets in our experiments included full information about  $\tau$ .

**Shortest  $k$ -Route.** Consider a dataset  $D$  with  $n$  objects  $o_1, \dots, o_n$ . A *complete route* over  $D$  is a sequence  $\rho = o_{i_1}, \dots, o_{i_n}$  where  $i_1, \dots, i_n$  is some permutation of  $1, \dots, n$ . The complete route  $\rho$  provides an order for traversing the objects of  $D$ . Now, suppose that we are given an instance  $I = (D, \tau)$ , which includes  $\tau$  in addition to  $D$ . Consider a traversal that starts at some given point  $s$  and visits the objects according to  $\rho$ . For each object  $o$ , we can count the number of correct objects and the distance until we get to  $o$ . Formally, we denote by  $correct_\rho(o_{i_j})$  the number of correct objects among  $o_{i_1}, \dots, o_{i_j}$ . That is,

$$correct_\rho(o_{i_j}) = |\{o_{i_l} \mid 1 \leq l \leq j \text{ and } \tau(o_{i_l}) = true\}|.$$

Also, we denote by  $distance_\rho(s, o_{i_j})$  the distance of the path that starts at  $s$  and leads to  $o_{i_j}$  according to  $\rho$ . That is,

$$distance_\rho(s, o_{i_j}) = distance(s, o_{i_1}) + \sum_{l=1}^{j-1} distance(o_{i_l}, o_{i_{l+1}}).$$

Given an instance  $I = (D, \tau)$  and a complete route  $\rho = o_{i_1}, \dots, o_{i_n}$  over  $D$ , a  *$k$ -route* is the shortest subsequence  $o_{i_1}, \dots, o_{i_j}$  such that  $correct_\rho(o_{i_j}) = k$ ; however, if such a subsequence does not exist (*i.e.*,  $correct_\rho(o_{i_n}) < k$ ), then the  $k$ -route is  $\rho$  itself. Intuitively, a  $k$ -route is a traversal that stops at the  $k$ -th correct object. We denote by  $k$ -*distance* $(s, \rho, I)$  the distance of the  $k$ -route  $o_{i_1}, \dots, o_{i_j}$  when starting at  $s$ , that is,  $k$ -*distance* $(s, \rho, I) = distance_\rho(s, o_{i_j})$ .



For an uncertain dataset, there can be many possible instances having  $k$ -routes with different lengths. Thus, we consider an expected length rather than an exact length. Given an uncertain dataset  $(D, \varphi_c)$ , a starting point  $s$  and a complete route  $\rho$  over  $D$ , the *expected length* of a  $k$ -route is

$$\Sigma_I \text{ is an instance of } (D, \varphi_c) [P(I) \cdot k\text{-distance}(s, \rho, I)].$$

The shortest  $k$ -route over an uncertain dataset  $(D, \varphi_c)$  is a complete route  $\rho$  that has an expected length smaller or equal to the expected length of any other  $k$ -route over  $(D, \varphi_c)$ . Since computing the shortest  $k$ -route is computationally hard, our goal in this work is to provide polynomial-time algorithms for computing a short  $k$ -route.

**Assessing the Quality of the Result.** In this work, we present three algorithms to the problem of finding a short  $k$ -route. In order to assess the quality of the results of these algorithms, we compare the expected length of the  $k$ -routes that the different algorithms compute. An algorithm  $A_1$  is considered better than algorithm  $A_2$  with respect to an uncertain dataset  $(D, \varphi_c)$  and a starting point  $s$ , if the expected length of the  $k$ -route produced by  $A_1$  is shorter than the expected length of the  $k$ -route produced by  $A_2$ . Given a digital map that contains  $D$ , algorithm  $A_1$  is better than  $A_2$  for  $(D, \varphi_c)$  if the number of points  $s$  (of the map) for which  $A_1$  is better than  $A_2$  is greater than the number of points  $s$  for which  $A_2$  is better than  $A_1$ .

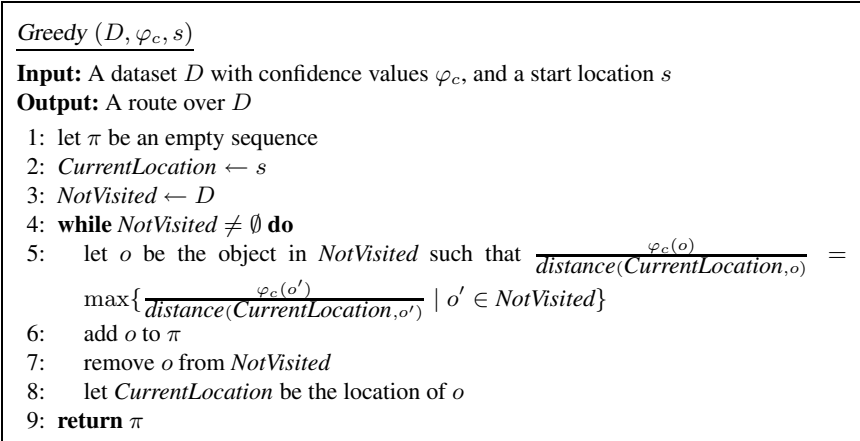
### 3 Algorithms

In this section, we present three novel algorithms for finding a short  $k$ -route. We use the following notation when presenting the algorithms. We denote by  $(D, \varphi_c)$  the given uncertain dataset and by  $o_1, \dots, o_n$  the objects of  $D$ . We denote by  $s$  the location where the traversal should start. The result of the algorithms is a sequence  $o_{i_1}, \dots, o_{i_n}$  that defines a complete route.

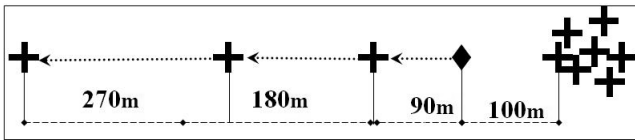
#### 3.1 The Greedy Algorithm

In the greedy algorithm, a route is constructed iteratively. Intuitively, in each iteration, the algorithm adds (to the sequence) the object that has the best ratio of confidence to distance among the objects that have not yet been added in previous iterations. The algorithm is presented in Fig. 1. Note that when choosing which object to add, while constructing the route, objects with high confidence are preferred over objects with low confidence and near objects are preferred over far objects.

The greedy algorithm is simple and efficient. No preprocessing is required and it has  $O(|D|^2)$  time complexity. It usually performs well (*i.e.*, provides a short  $k$ -route) in the following two cases. First, when  $k$  is very small. In particular, this is true for  $k = 1$ . Secondly, when the objects of  $D$  are uniformly distributed and there is no correlation between confidence values and locations. Intuitively, in such cases, there is no preferred direction for the first leg of the traversal (which starts at  $s$ ). Hence, the initial direction chosen by the greedy algorithm is as good as any other direction, and the produced route will have an expected distance close to the optimal.



**Fig. 1.** The greedy algorithm



**Fig. 2.** An example where the greedy algorithm does not perform well. The starting point is marked by a diamond. Objects are marked by crosses.

When  $k$  is large and the distribution of either the objects or their confidences is not homogeneous, the greedy algorithm is not likely to provide good results. The following example illustrates a problematic behavior of the greedy algorithm.

*Example 3.* Fig. 2 shows a dataset that has a cluster of objects on the right side, and three objects with growing distances between them on the left side. Suppose that all three objects have the same confidence value. Given the starting location marked by a diamond, the route computed by the greedy algorithm will first go to the three objects on the left instead of going to the cluster on the right. For  $k = 4$ , for instance, it is better to start the route by going to objects in the cluster on the right side.

From Example 3, we can learn that the greedy algorithm is not an approximation algorithm to the shortest  $k$ -route problem. That is, for any given positive constant  $c$ , it is possible to construct an example in which the greedy algorithm will return a  $k$ -route whose expected length is greater than the expected length of the shortest  $k$ -route multiplied by  $c$ . Constructing such an example is done by generating a dataset similar to the one in Example 3, choosing a large enough  $k$ , and appropriately adding more objects (with growing distances between them) on the left side of the starting location and adding objects to the cluster on the right side.

AAG ( $D, \varphi_c, s$ )

**Parameter:** An accuracy parameter  $\epsilon$

**Input:** A dataset  $D$  with confidence values  $\varphi_c$ , and a start location  $s$

**Output:** A route over  $D$

- 1: generate a weighted graph  $G$  from  $D$  and  $\varphi_c$
- 2: generate a transition matrix  $P$  from  $G$
- 3: create a uniform distribution  $X_1 = (\frac{1}{n}, \dots, \frac{1}{n})$
- 4:  $t \leftarrow 1$
- 5: **while**  $\|P^{t+1}X_1 - P^tX_1\| \geq \epsilon$  **do**
- 6:    $t \leftarrow t + 1$
- 7: create from  $P^tX_1$  a function  $X^s$  that provides an aa-value to each node
- 8:  $\pi \leftarrow \text{Greedy}(D, X^s, s)$
- 9: **return**  $\pi$

**Fig. 3.** The Adjacency-Aware Greedy algorithm

While the greedy algorithm is not an approximation algorithm to the shortest  $k$ -route problem, it is an approximation algorithm for TSP [15]. This shows that in some aspects, the shortest  $k$ -route problem is inherently different from TSP.

### 3.2 The Adjacency-Aware Greedy Algorithm

Dealing with clusters of objects is important in many real-world scenarios. For example, in many cities, hotels are grouped near airports or tourist sites. Restaurants are usually located in the city center, near tourist sites and in the business district. Similarly, other utilities, such as shops or municipal buildings, are usually grouped together rather than being uniformly dispersed all over the city.

When a given dataset contains clusters of objects, a good heuristic is to give precedence to points that are in a cluster over points that are not in a cluster. This, however, is not done by the greedy algorithm, as shown in Example 3. The *Adjacency-Aware Greedy Algorithm* (AAG) improves the greedy algorithm by preferring objects that are surrounded by many near objects, especially if the near objects have high confidence values. This is done by means of assigning *adjacency-aware values* (abbr. *aa-values*) to objects as follows.

The aa-value given to an object should be based not only on the distances of the other objects and their confidence values, but also on their configuration. For example, we should prefer an object that has a neighboring cluster of four objects, within a distance of 100 meters, over an object that has four neighbors, all of them at a distance of 100 meters but in four different directions.

To compute the aa-values, we represent the dataset as a graph with weighted edges. We use the weights to compute, for each object, an aa-value that is the probability of reaching that object in a random walk on the graph. The weight of an edge  $(o_1, o_2)$  represents the probability of moving from  $o_1$  to  $o_2$  and is determined by the distance between the two objects and their confidence values. In a random walk on the graph, an object with many near neighbors has a higher probability to be visited than an object

with fewer near neighbors. Furthermore, an increase in the aa-value of a node raises the aa-values of its neighbors for the following reason. If a node  $o$  has a higher probability to be visited in a random walk, then there is an increased likelihood of visiting the near neighbors of  $o$ . Hence, the aa-values of objects are affected by the configuration of the dataset.

Now, we formally define the weighted graph and show how to compute the probability of reaching a node by a random walk on this graph. Given the uncertain dataset  $(D, \varphi_c)$ , we generate a weighted graph  $G = (V, E, w)$ , where the set of nodes  $V$  consists of all the objects in  $D$ , the set of edges  $E$  is  $D \times D$ , i.e., there is an edge in  $G$  between every two nodes, and  $w$  is a function that maps each edge  $e = (o_1, o_2)$  of  $E$ , where  $o_1 \neq o_2$ , to the weight  $w(e) = \frac{\varphi_c(o_2)}{\text{distance}(o_1, o_2)}$ . For each object  $o$ , we define  $w((o, o)) = 0$ . A random walk over  $G$  is a stochastic process that chooses the next node to visit as follows. If we are at some node  $v$ , we randomly choose an outgoing edge of  $v$ . The probability of choosing an edge is proportional to its weight. The random walk creates a sequence  $v_1, v_2, \dots, v_t, \dots$  of nodes. Since the walk is random, the node  $v_t$  that is visited after  $t$  steps can be any node of  $G$ —each node with a different probability. We denote by  $X_t$  the probability distribution over  $V$  of being at each node after  $t$  steps. We represent  $X_t$  as a vector of probabilities of length  $|D|$ . That is,  $X_t[i]$  is the probability to be at node  $o_i$  after  $t$  steps.

The random walk is a memoryless process, that is, each step depends only on the last state. In other words, the probability of choosing an outgoing edge for making the next step is independent of the path that led to the current node. Hence, it is a Markov chain, which means that the random walk can be described using an  $n \times n$  transition matrix  $P$ , such that  $X_{t+1} = PX_t$  holds for every  $t$  (note that  $n$  is the number of objects in  $D$ ). We denote by  $P_{ij}$  the element in the  $i$ th column and the  $j$ th row of  $P$ . The element  $P_{ij}$  is the probability to move from node  $o_i$  to node  $o_j$ . Since the choice of edges is according to their weights, we define  $P$  as follows.

$$P_{ij} = \frac{w(o_i, o_j)}{\sum_{j'=1}^n w(o_i, o_{j'})}$$

Note that  $\sum_{i=1}^n P_{ij} = 1$  holds for every row  $j$ .

The transition matrix  $P$  defines an irreducible and aperiodic Markov chain. (Intuitively, irreducible means that from each node there is a non-zero probability to reach any other node, since the graph is connected; aperiodic means that for each node, 1 is the greatest common divisor of the lengths of all paths from this node to itself, since the graph is not bipartite.) So, given an initial uniform distribution  $X_1 = (\frac{1}{n}, \dots, \frac{1}{n})$ , we have that  $P^t X_1 \rightarrow X^s$  as  $t \rightarrow \infty$ , where  $X^s$  is a stationary distribution, that is,  $PX^s = X^s$ . For each  $i$ , the distribution  $X^s$  gives the probability to be at  $o_i$  in a random walk on  $G$ .

The AAG algorithm of Fig. 3 computes the stationary distribution  $X^s$  and then applies the greedy algorithm where  $X^s$  replaces  $\varphi_c$ . Computing  $X^s$  can be done as a pre-processing step. Thus, given a user request with a specific location, the time complexity of computing a route is the same as the time complexity of the greedy algorithm.

Our experiments show that the AAG algorithm improves the greedy algorithm. However, AAG has the disadvantage that the probability distribution  $X^s$  must be computed

$k$ -EG ( $D, \varphi_c, s$ )

**Parameter:** The number  $k$  of correct objects to visit

**Input:** A dataset  $D$  with confidence values  $\varphi_c$ , and a start location  $s$

**Output:** A route over  $D$

```

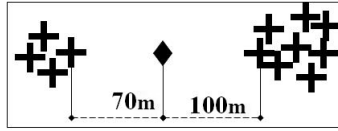
1:  $\mathcal{K} \leftarrow \emptyset$ 
2:  $\mathcal{S} \leftarrow \{\{o\} \mid o \in D\}$ 
3: while  $\mathcal{S} \neq \emptyset$  do
4:   for each set  $S$  in  $\mathcal{S}$  do
5:     if  $k \leq \sum_{o \in S} \varphi_c(o)$  then
6:       add  $S$  to  $\mathcal{K}$ 
7:       remove  $S$  from  $\mathcal{S}$ 
8:     else
9:        $minLength \leftarrow \infty$ 
10:      for each  $o \in D - S$  do
11:        let  $l_{S,o}$  be the length of the route created by a greedy algorithm for the
        starting point  $s$  and the objects  $S \cup \{o\}$ , based on merely distances,
        without using confidence values (at each iteration the greedy adds to
        the path the nearest object to the current location among the objects not
        added so far)
12:        if  $l_{S,o} < minLength$  then
13:           $objToAdd \leftarrow o$ 
14:           $minLength \leftarrow l_{S,o}$ 
15:        add  $objToAdd$  to  $S$ 
16:       $minLength \leftarrow \infty$ 
17:       $chosenSet \leftarrow \emptyset$ 
18:      for each  $S$  in  $\mathcal{K}$  do
19:        let  $l_S$  be the length of the route created by a greedy algorithm for the starting
        point  $s$  and the objects of  $S$ , according to distance and without using the confi-
        dence values
20:        if  $l_S < minLength$  then
21:           $chosenSet \leftarrow S$ 
22:           $minLength \leftarrow l_S$ 
23: let  $\pi$  be the route that starts at  $s$  and is generated by a greedy algorithm using only
        distances, over the objects of  $chosenSet$ 
24: complete  $\pi$  to include all the objects of  $D$  using the greedy algorithm as in Fig. 5
        (when choosing which object to add use the ratio of distance and confidence)
25: return  $\pi$ 

```

**Fig. 4.** The  $k$ -Expectancy Grouping algorithm

before computing a route, and hence AAG is less efficient than the greedy algorithm for datasets that change frequently. AAG also suffers from the following two problems.

1. AAG ignores  $k$  when computing the route. For instance, consider the case that is depicted in Fig. 5, assuming that all the objects have the same confidence value. There is a small cluster on the left side of the starting point and a larger cluster on the right side of the starting point. The smaller cluster is closer to the starting point



**Fig. 5.** An example where the AAG algorithm does not perform well. The starting point is marked by a diamond. Objects are marked by crosses.

than the larger cluster. For large values of  $k$ , it is better to go to the bigger cluster first. But for small values of  $k$ , going to the nearer (and smaller) cluster may be a better approach. In AAG, however, the same path is returned for all values of  $k$ .

2. A second problem is that by going directly to points in a cluster, there may be points on the way to the cluster, such that visiting them would not increase the distance of the route and yet, in the AAG method, such points are not always visited.

Our third method solves the above problems.

### 3.3 The $k$ -Expectancy Grouping Algorithm

We now present the third method, namely, the  $k$ -Expectancy Grouping ( $k$ -EG) algorithm. Differently from the previous methods, the route generated by this algorithm depends not only on the dataset and the starting point, but also on the value of  $k$ . The  $k$ -EG algorithm consists of two steps. The first creates sets of objects such that the expected number of correct objects in each one is  $k$ . The second step applies the greedy algorithm to each one of these sets, and chooses the set for which the greedy algorithm generates the shortest route.

The  $k$ -EG algorithm is shown in Fig. 4. In the first part of the algorithm, sets of objects are generated and inserted into  $\mathcal{K}$ . The sets in  $\mathcal{K}$  are constructed so that the sum of confidence values, of the objects in each set, is greater than  $k$ . This means that for the sets in  $\mathcal{K}$ , the average number of correct objects is at least  $k$ . Initially,  $\mathcal{K}$  is empty.

The algorithm uses  $\mathcal{S}$  to store sets that are eventually moved to  $\mathcal{K}$ . Initially, for each object  $o$  in  $D$ , the set  $\{o\}$  is in  $\mathcal{S}$ . Then, we iteratively extend the sets in  $\mathcal{S}$  by adding one object at a time, as described below. When a set has (for the first time) a confidence sum that is at least  $k$ , it is moved to  $\mathcal{K}$ . In order to extend a set  $S$  of  $\mathcal{S}$  by one object, we examine all the objects  $o$  of  $D$  that are not yet in  $S$ . For each object  $o$ , we compute a route that starts at  $s$  and traverses the objects of  $S \cup \{o\}$ . This route is computed by a greedy algorithm that uses ordinary distances (i.e., it is essentially the same algorithm as in Fig. 1, except that all the confidence values are equal to 1). The object  $o$  for which the constructed route is the shortest is the one that is added to  $\mathcal{S}$ .

After constructing the sets (Lines 1–15), we choose the one that has the shortest route (Lines 16–22). Then, a route is created from the chosen set by applying the greedy algorithm with ordinary distances. After traversing all the objects of the chosen set, we continue the route by visiting all the remaining objects of  $D$ , but now we apply the greedy algorithm that uses the ratio of the confidence to the distance.

In general,  $k$ -EG has  $O(n^5)$  time complexity, where  $n$  is the number of objects in  $D$ . To see why this is true, note that initially there are  $n$  sets in  $\mathcal{S}$ . Since the number of sets

in  $\mathcal{S}$  does not grow, there are at most  $n$  sets in  $\mathcal{S}$  during the entire run of the algorithm. Also, each set contains at most  $n$  objects. Every set can be extended at most  $n$  times, each time by choosing an object from a set of at most  $n$  possible objects. So, there are at most  $n^2$  times of considering whether to add a certain object to a certain set, which means no more than  $n^3$  times of computing a route using a greedy algorithm, for all the  $n$  sets. Since for each set  $S$  the greedy algorithm has an  $O(|S|^2)$  running time, the total time is  $O(n^5)$ .

In practice, the sets in  $\mathcal{S}$  are expected to have a size that is much smaller than  $n$ . It is reasonable to assume that in practical cases, the sets of  $\mathcal{S}$  (and hence, also the sets in  $\mathcal{K}$ ) have an  $O(k)$  size. If we consider, for instance, the case where all the objects in  $D$  have confidence values greater than 0.5, then every set in  $\mathcal{S}$  has at most  $2k$  objects. Under the assumption that sets in  $\mathcal{S}$  have an  $O(k)$  size, the running time of the algorithm is  $O(n^2k^3)$ . When  $k$  is constant, we actually get an  $O(n^2)$  running time.

## 4 Experiments

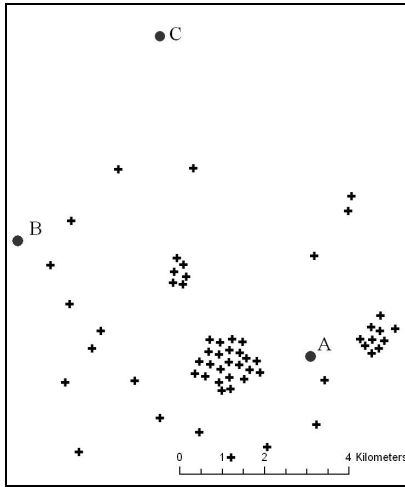
In this section, we describe the results of extensive experiments on both real-world data and synthetically-generated data. The goal of our experiments was to compare the three methods presented in Section 3 over data with varying levels of object spread and different distributions of confidence values.

### 4.1 Tests on Synthetic Data

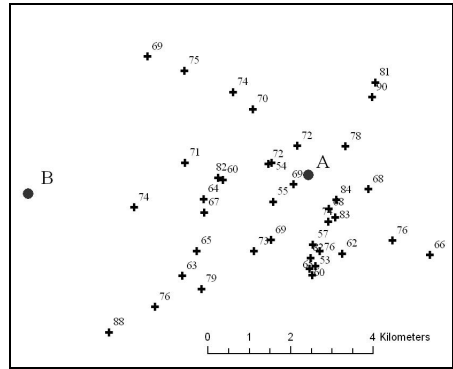
We used synthetic datasets to test the differences between our algorithms. One of the synthetic datasets on which we conducted experiments is depicted in Fig. 6. In this figure, objects are marked by crosses. Potential starting points are marked by circles and have a letter (A, B or C) next to the circle. The confidence values were chosen randomly according to a Gaussian distribution (normal distribution) with mean 0.7 and standard deviation 0.1. We do not show the confidence values in Fig. 6 because in some parts of the figure, objects are too dense for writing visible numbers next to them.

For estimating the expected distance of a route  $\rho$  over some given dataset  $(D, \varphi_c)$ , when testing the quality of some algorithm, we generated 100 instances of  $(D, \varphi_c)$  and computed the average distance of a  $k$ -route over these instances. That is, for every given dataset  $(D, \varphi_c)$ , we generated 100 instances  $(D, \tau_1), \dots, (D, \tau_{100})$  where each  $\tau_i$  was the result of randomly choosing truth values  $\tau_i(o_1), \dots, \tau_i(o_n)$ , such that  $\varphi_c(o_j)$  and  $1 - \varphi_c(o_j)$  were the probabilities of choosing  $\tau_i(o_j)$  to be *true* and *false*, respectively. We then computed the distances  $d_1, \dots, d_{100}$ , where  $d_i$  is the length of the route from the starting point to the  $k$ th correct object when traversing  $(D, \tau_i)$  according to  $\rho$ . We consider the average  $(\sum_{i=1}^{100} d_i)/100$  as the expected distance of  $\rho$  over  $(D, \varphi_c)$ .

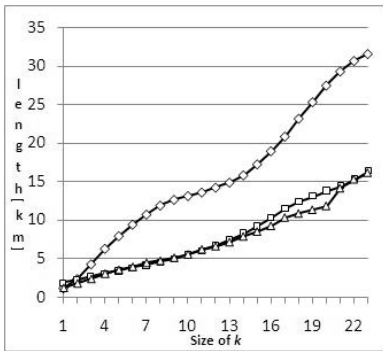
Fig. 8 shows the results of our algorithms when computing a route over the dataset of Fig. 6 where A is the starting point. The graph in this figure shows the expected  $k$ -distance, of the routes computed by the algorithms, as a function of  $k$ . The results of the greedy algorithm are presented by diamonds. For AAG, the results are depicted by squares, and for  $k$ -EG, the results are depicted by triangles. The graph shows that for small  $k$  values ( $k = 1$  or  $k = 2$ ), all three algorithms provide a route with a similar expected distance. For larger  $k$  values, the greedy algorithm is much worse than AAG and



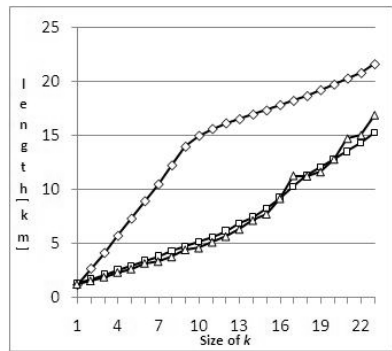
**Fig. 6.** A synthetic dataset



**Fig. 7.** A dataset of hotels in Soho, Manhattan



**Fig. 8.** Results on the dataset of Fig. 6 starting at point A

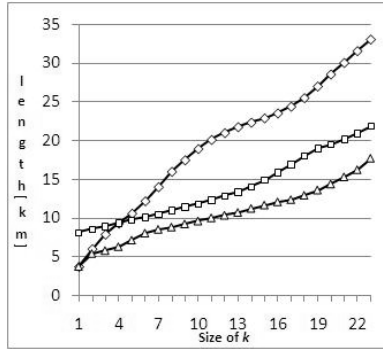


**Fig. 9.** Results on the dataset of Fig. 6 starting at point B

$k$ -EG. For instance, when  $k = 7$ , the route of greedy algorithm has an expected length that is greater than 10 kilometers while AAG and  $k$ -EG provide routes with expected lengths of less than 5 kilometers. The differences are because AAG and  $k$ -EG generate a route that goes directly to a near cluster while the route generated by the greedy algorithm does not go directly to a cluster. For the starting point B, the differences in the quality of the results, between the greedy and the other two algorithms, are even larger, because it takes longer for the route of the greedy algorithm to get to a cluster.

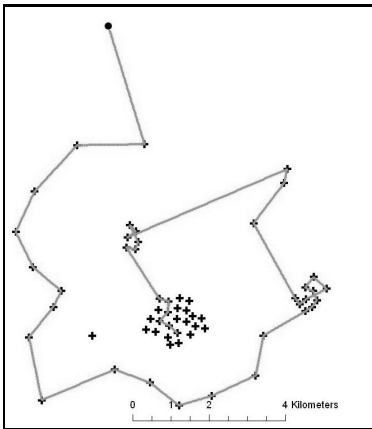
Fig. 10 shows the results of our algorithms when computing a route over the dataset of Fig. 6 using C as the starting point. In this case, there is a difference between the results provided by AAG and those of  $k$ -EG. In order to understand the behavior of the different algorithms in this case, we present the routes that are computed. The greedy



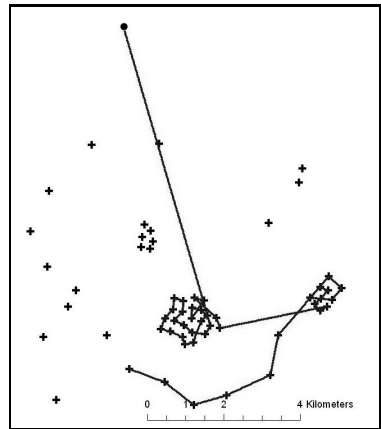


**Fig. 10.** Results on the dataset of Fig. 6 starting at point C

algorithm returns the route that is depicted in Fig. 11. AAG returns the route in Fig. 12. The route that  $k$ -EG returns for  $k = 7$  is presented in Fig. 13. In these figures, it can be seen that the route computed by the greedy algorithm reaches a cluster after a long travel. AAG reaches a cluster directly and thus is better than the greedy algorithm for large  $k$  values. The main problems with the route that AAG computes is that it goes directly to a cluster and skips objects that are on the way to the cluster. Going through these objects increases the likelihood of reaching  $k$  correct object sooner without lengthening the route. Thus, for this case,  $k$ -EG provides a better route than AAG.

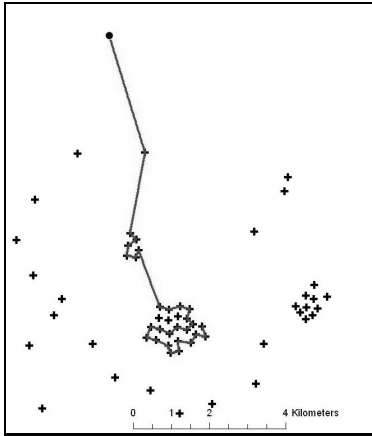


**Fig. 11.** The route by the greedy algorithm on the dataset of Fig. 6 starting at point C

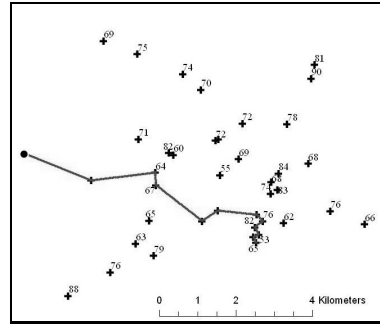


**Fig. 12.** The route by AAG on the dataset of Fig. 6 starting at point C

We conducted several additional tests on synthetic datasets. In these tests, we had datasets with a few large clusters, datasets with several small clusters and datasets with no clusters at all. Our experiments confirmed that in the presence of clusters, the greedy



**Fig. 13.** The route by  $k$ -EG on the dataset of Fig. 6 starting at point C, for  $k = 7$



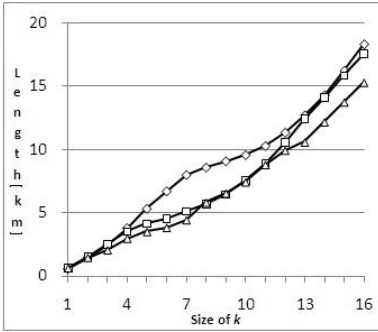
**Fig. 14.** The route by  $k$ -EG on the dataset of Fig. 7 starting at point B, for  $k = 7$

algorithm is much worse than the other two algorithms, and they showed that  $k$ -EG provides the best results in almost all cases.

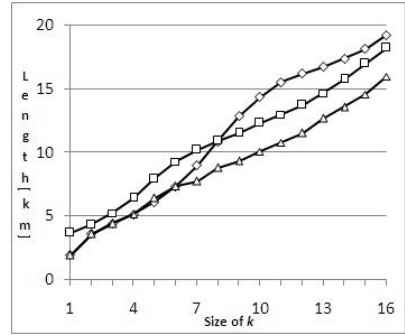
## 4.2 Tests on Real-World Data

We tested our algorithms on several real-world datasets to which we added confidence values. A dataset of hotels in Soho, Manhattan, is depicted in Fig. 7. The objects were taken from a map of New-York City and the confidence values were added randomly according to a Gaussian distribution with mean 0.7 and standard deviation 0.1. The results of our algorithms on this dataset are depicted in Fig. 15 and Fig. 16 for the starting points A and B, respectively. In this test, once again, the greedy algorithm provides the worst route and  $k$ -EG provides the best route, for almost all cases. The routes computed by the greedy, AAG and  $k$ -EG algorithms are depicted in Figures 17, 18 and 14, respectively.

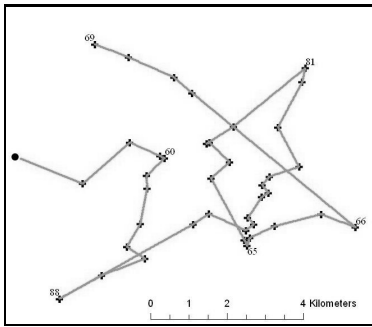
In  $k$ -EG, a route is chosen from a set of possible routes. Intuitively, this reduces the number of cases where the algorithm produces an extremely bad route. To show it, we conducted experiments over three real-world datasets that are very different one from the other, using two distinct confidence distributions. One dataset that we used is of embassies in Tel-Aviv. In this dataset, almost all the objects are in two clusters that are quite far one from the other. A second dataset is of gas stations in the area of Tel-Aviv. This dataset contains three large clusters (dense urban areas) but also many objects that do not belong to a cluster. A third dataset that we used is of points of interest where objects are dispersed without any visible cluster. For each one of these datasets, we chose confidence values randomly. First, according to a uniform distribution in the range 0 to 1, and secondly, according to a Gaussian distribution with mean 0.7 and standard deviation 0.1. For each case, we chose a starting location.



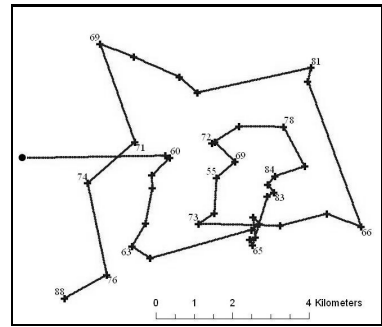
**Fig. 15.** Results on real-world dataset, starting at point A



**Fig. 16.** Results on real-world dataset, starting at point B



**Fig. 17.** The route by the greedy algorithm on the dataset of Fig. 7 starting at point B



**Fig. 18.** The route by AAG on the dataset of Fig. 7 starting at point B

Over each dataset, we summarized for AAG and  $k$ -EG the quality of the result with respect to the result of the greedy algorithm. To do so, we computed for  $k = 2, \dots, 10$  the ratio of the distance of the route produced by the tested algorithm (AAG or  $k$ -EG) to the distance of the route produced by the greedy algorithm. We show the minimal and the maximal ratios for these cases in Fig. 19.

The graph in Fig. 19 shows that AAG sometimes generates a route that is much worse than that of the greedy algorithm. This is due to the fact that in the presence of clusters, the route generated by AAG goes directly to a cluster even when all the clusters are far from the starting point. This approach can be expensive, especially for small  $k$  values. In the presence of clusters, both AAG and  $k$ -EG sometimes produce a route that is much better than the route produced by the greedy algorithm. Not surprisingly, when there are no clusters, the differences between the algorithms are smaller. Note that we get similar results for different distributions of confidence values, but an increase in the variance of confidence values leads to an increase in the difference between the smallest and the largest ratios.

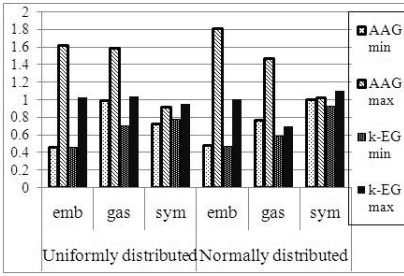


Fig. 19. The results of the algorithms summed up for several real-world sources

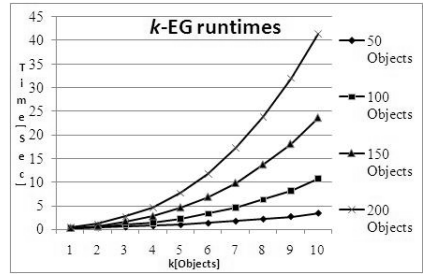


Fig. 20. The runtimes (in seconds) of  $k$ -EG as a function of  $k$ , on dataset of different sizes

### 4.3 Running Times

We now consider the time it takes to compute a route using our algorithms. To give running-time estimations, we measured the computation of a route on datasets of different sizes. When measuring the times, we used a PC with a Core 2 Duo, 2.13 GHz, processor (E6400) and 2GB of main memory. In Table 1 we show the time it takes to compute, using the greedy and AAG algorithms, a route over four datasets with 50, 100, 150 and 200 objects. For AAG, we show both the time it takes to compute adjacency-aware values, in the preprocessing part of the method, and the time it takes to compute a route after the preprocessing has been completed. For  $k$ -EG, we present in Fig. 20 the time for computing a route as a function of  $k$ . Table 1 and Fig. 20 show that the greedy algorithm is the most efficient among the three algorithms while  $k$ -EG is less efficient than the other two methods. AAG is less efficient than the greedy algorithm when including the preprocessing time, but without the preprocessing time, AAG is as efficient as the greedy algorithm.

Table 1. The time for computing a route over datasets of different sizes

	50 objects	100 objects	150 objects	200 objects
Greedy	<0.01 sec	0.02 sec	0.02 sec	0.02 sec
AAG preprocessing	0.02 sec	0.03 sec	0.08 sec	0.13 sec
AAG compute route	<0.01 sec	<0.01 sec	0.02 sec	0.02 sec

## 5 Related Work

With the ongoing advances in the areas of wireless communication and positioning technologies, it has become possible to provide mobile, location-based services. These services may track the movements and requests of their customers in multidimensional data warehouses, and later use this information for answering complex queries [10]. Data models for location-based services have been developed and implemented in recent years. An R-tree-based technique for indexing data about the current positions of objects in highly dynamic databases has been proposed by Saltenis and Jensen [18]. An

efficient search for specific information over multiple collections has been described by Goodchild and Zhou [9], who have also reported on several conceptual designs for a searching process that is based on collection-level metadata (CLM). Miller and Shaw [12] have described the use of GIS-T data models and different aspects of path finding in geospatial systems for transportation purposes.

Manipulating uncertain and probabilistic data has received a lot of attention recently. Several papers deal with managing probabilistic and uncertain data, and propose models for representing the data [2,5,8,11]. In some papers, the problem of querying probabilistic data is considered and various techniques for efficient evaluation of queries over probabilistic data are proposed [6,7,14,16,23]. The above papers are concerned with probabilistic data in general, and not with spatial data. For probabilistic spatial data, the problem of computing a join of spatial polygonal-shaped objects with imprecise locations is investigated in [13]. Computing nearest-neighbor on probabilistic spatial databases is discussed in [22]. Probabilistic spatial data has also been considered in the context of dealing with moving objects [18,19,20]. All these problems are different from the one discussed in this paper, namely, finding the shortest  $k$ -route.

## 6 Conclusion

In this work, we introduced the problem of finding the shortest  $k$ -route over uncertain geo-spatial datasets. Since the problem is computationally hard, we presented three heuristic algorithms for computing a short  $k$ -route, and illustrated the differences between these algorithms. We compared the algorithms using extensive experiments over synthetic and real-world data. Our experiments show that in most cases,  $k$ -EG provides the best route (*i.e.*, provides a route that is expected to lead to  $k$  correct objects within a shorter distance) and the greedy algorithm provides the worst route. However, for these algorithms, there is a tradeoff between the quality of the results and the efficiency of the algorithm. The greedy algorithm is the most efficient and  $k$ -EG is the least efficient among the three. As future work, we intend to develop optimization techniques to improve the efficiency of  $k$ -EG.

## References

1. Andritsos, P., Fuxman, A., Miller, R.J.: Clean answers over dirty databases: A probabilistic approach. In: Proceedings of the 22 International Conference on Data Engineering (2006)
2. Barbara, D., Garcia-Molina, H., Poter, D.: The management of probabilistic data. IEEE Transaction on Knowledge and Data Engineering 4(5), 487–502 (1992)
3. Beeri, C., Doytsher, Y., Kanza, Y., Safra, E., Sagiv, Y.: Finding corresponding objects when integrating several geo-spatial datasets. In: ACM-GIS, Bremen, Germany, pp. 87–96. ACM Press, New York (2005)
4. Beeri, C., Kanza, Y., Safra, E., Sagiv, Y.: Object fusion in geographic information systems. In: VLDB, pp. 816–827 (2004)
5. Cavallo, R., Pittarelli, M.: The theory of probabilistic databases. In: Proceedings of 13th International Conference on Very Large Data Bases (1987)
6. Cheng, R., Kalashnikov, D., Parbhakar, S.: Evaluating probabilistic queries over imprecise data. In: Proc. of ACM SIGMOD International Conference on Management of Data, San Diego (CA, USA), ACM Press, New York (2003)

7. Dalvi, N.N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: Proceedings of the 30th International Conference on Very Large Data Bases (2004)
8. Fuhr, N.: A probabilistic framework for vague queries and imprecise information in databases. In: Proc. of the 16th International Conference on Very Large Data Bases (1990)
9. Goodchild, M.F., Zhou, J.: Finding geographic information: Collection-level metadata. *Geoinformatica* 7(2), 95–112 (2003)
10. Jensen, C.S., Kligys, A., Pedersen, T.B., Timko, I.: Multidimensional data modeling for location-based services. *The VLDB Journal* 13(1), 1–21 (2004)
11. Lakshmanan, L.V.S., Leone, N., Ross, R., Subrahmanian, V.S.: Probview: A flexible probabilistic database system. *ACM Trans. on Database Systems* 22(3), 419–469 (1997)
12. Miller, H.J., Shih-Lung, S.: *Geographic Information Systems for Transportation: Principles and Applications (Spatial Information Systems)*. Oxford University Press, Oxford (2001)
13. Ni, J., Ravishankar, C.V., Bhanu, B.: Probabilistic spatial database operations. In: Proc. of the 8th International Symposium on Advances in Spatial and Temporal Databases (2003)
14. Pittarelli, M.: An algebra for probabilistic databases. *IEEE Transactions on Knowledge and Data Engineering* 6(2), 293–303 (1994)
15. Rosenkrantz, D.J., Stearns, R.E., Lewis II, P.M.: An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing* 6, 563–581 (1977)
16. Ross, R., Subrahmanian, V.S., Grant, J.: Aggregate operators in probabilistic databases. *Journal of the ACM* 52(1), 54–101 (2005)
17. Safra, E., Kanza, Y., Sagiv, Y., Doytsher, Y.: Integrating data from maps on the world-wide web. In: Proceedings of the 6th International Symposium on Web and Wireless Geographical Information Systems, pp. 180–191 (2006)
18. Saltenis, S., Jensen, C.S.: Indexing of moving objects for location-based services. In: Proceedings of the 18th International Conference on Data Engineering, Washington DC (USA) (2002)
19. Trajcevski, G., Wolfson, O., Hinrichs, K., Chamberlain, S.: Managing uncertainty in moving objects databases. *ACM Transactions on Database Systems* 29(3), 463–507 (2004)
20. Trajcevski, G., Wolfson, O., Zhang, F., Chamberlain, S.: The geometry of uncertainty in moving objects databases. In: Proceedings of the 8th International Conference on Extending Database Technology (2002)
21. Verrantaus, K., Markkula, J., Garmash, A., Terziyan, Y.V.: Developing GIS-supported location-based services. In: Proceedings of the 1st International Conference on Web Geographical Information Systems, pp. 423–432 (2001)
22. Zhang, S.: A nearest neighborhood algebra for probabilistic databases. *Intelligent Data Analysis* 4(1), 29–49 (2000)
23. Zimányi, E.: Query evaluation in probabilistic relational databases. *Theoretical Computer Science* 171(1-2), 179–219 (1997)

# Querying Objects Modeled by Arbitrary Probability Distributions

Christian Böhm, Peter Kunath, Alexey Pryakhin, and Matthias Schubert

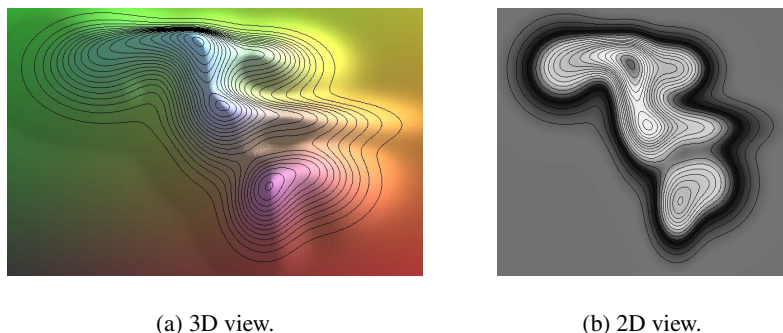
Institute for Computer Science, Ludwig-Maximilians Universität München  
{boehm, kunath, pryakhin, schubert}@dbs.ifi.lmu.de  
<http://www.dbs.ifi.lmu.de>

**Abstract.** In many modern applications such as biometric identification systems, sensor networks, medical imaging, geology, and multimedia databases, the data objects are not described exactly. Therefore, recent solutions propose to model data objects by probability density functions(pdf). Since a pdf describing an uncertain object is often not explicitly known, approximation techniques like Gaussian mixture models(GMM) need to be employed. In this paper, we introduce a method for efficiently indexing and querying GMMs allowing fast object retrieval for arbitrary shaped pdf. We consider probability ranking queries which are very important for probabilistic similarity search. Our method stores the components and weighting functions of each GMM in an index structure. During query processing the mixture models are dynamically reconstructed whenever necessary. In an extensive experimental evaluation, we demonstrate that GMMs yield a compact and descriptive representation of video clips. Additionally, we show that our new query algorithm outperforms competitive approaches when answering the given probabilistic queries on a database of GMMs comprising about 100.000 single Gaussians.

## 1 Introduction

In recent years, a large variety of database applications has emerged where it is beneficial to consider the uncertainty of the given data. The object uncertainty might be caused by the inexactness of feature measurement like in biometrical or biological databases. Furthermore, object uncertainty is often introduced to obtain a compact description of very large or complex objects. For example, in sensor networks, it is not feasible to transmit an exact value at all points of time. Therefore, several points of time are aggregated into a single uncertain description. In order to model this uncertainty, the data objects are described by probability density functions (pdf) over a given feature space (cf. [1,2,3,4,5,6]).

In most applications, there is no explicitly known pdf describing the uncertain data object. Instead, the density function is given by a sample of feature values or in the case of data compression, the set of feature values being compressed. Thus, approximation techniques have to be employed in order to find an explicit density function. In other words, we need to describe each object as a function instead of set of sample points. A large number of database applications in all fields of science and industry approximate the underlying data distributions by a *Gaussian mixture model (GMM)*. The idea of a

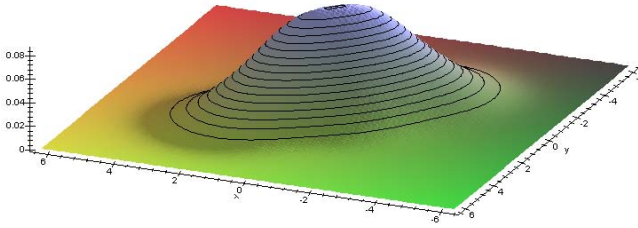


**Fig. 1.** Approximation of an arbitrary complex bivariate probability density by a mixture of Gaussians with independent attributes

GMM is to model a complex density function by a weighted sum of several Gaussian distributions. If a GMM is not exact enough, the quality can be arbitrarily increased by adding further Gaussians (cf. Figure 1). The popularity of GMMs can be explained by the following characteristics. A GMM has the ability to approximate an arbitrary statistical data distribution very closely [7,8]. An example for such an approximation can be seen in Figure 1. In particular, GMMs are capable of modeling arbitrarily shaped correlations in data (cf. [8]) by a mixture containing distributions with independent attributes (cf. Figure 2). Moreover, GMMs can be efficiently derived by a variety of mathematical methods from any given sample set. The widespread use of these methods contributes to prevalent employment of GMMs in several applications.

In the following, we survey some of example application areas employing GMMs more closely. The first example for using objects represented by GMMs is managing multimedia data. An ordinary movie of about 90 minutes contains about 140.000 single images, called frames. Storing all these images for content-based video retrieval would require large storage capacities and an enormous computational effort for comparing videos. To avoid this problem, the multimedia community often employs summarization techniques representing videos as mixture models [9]. Thus, a video clip is described by a GMM over the feature space representing single frames. Storing the videos as a GMM dramatically reduces the resource consumption and still allows accurate content-based video retrieval. Moreover, the use of mixture models is justified by the demand to consider high level or semantic features (cf. [10,11,12] for details) in order to guarantee effectiveness in the retrieval of multimedia data. For instance, [13] describes the usage of face detection and recognition techniques in order to calculate a compact description of video data. Another example for the use of GMMs is bioclimatic research. For instance, the authors of [14] propose the use of GMMs to describe species ranges from mapped climatic variables. Moreover, authors of [15] employ a GMM in order to describe drug dissolution profiles. They also discuss implications of the GMMs for pharmaceutical product formulation tasks. Further application areas of GMMs are sensor networks (e.g. [16,4]), audio signal analysis (e.g. [17]), economics (e.g. [18]). Additionally, the authors of [7,8] list several examples of the use of GMMs in the following application areas: medicine, psychology, geology, agriculture.





**Fig. 2.** A Gaussian mixture model representing correlated 2D data

In his paper, we propose a novel method for query processing on arbitrarily shaped pdf modeling each data object as a GMM. Our new models aims to answer identification queries of the following form: Given a query GMM and a database of GMMs, return the  $k$  database objects which might describe the same data object with the highest probability. Unlike previous approaches, we additionally consider the case that no database object describes the same object as the query. Thus, we can handle the case that it is most likely that there is no object in the database resembling the given query. To efficiently handle this type of query, we propose a method for storing GMMs in a probabilistic index structure. The principal idea of this method is to store each component (i.e., each Gaussian of a GMM and its weighting) as a separate object. To calculate the probability that a given query object corresponds to a database object, the GMMs are reconstructed during query time. Therefore, we store the information about the possible candidates in a so-called candidate table. The candidate table is located in main memory and stores two values for each object that was touched but is not yet complete. Based on these two values, it is possible to calculate a conservative approximation of the probability for a database object (see Section 4 for details). When a component belonging to an object  $o$  is retrieved from the underlying index structure the information about  $o$  (i.e., two values in the candidate table) is updated.

The main contributions of this paper are: (1) A new probabilistic model for handling uncertain objects represented by GMMs. (2) A method to allow unsuccessful search for probabilistic queries. (3) A new query algorithm for efficiently answering probabilistic ranking queries that are based on decomposing the GMMs.

The rest of the paper is organized as follows. In Section 2, we survey the related work on managing and searching databases of uncertain objects. Section 3 introduces our new uncertainty model and specifies the examined types of queries. Section 4 introduces a new method for object indexing and query processing. Our experimental results are shown in Section 5 and the paper concludes with a summary in Section 6.

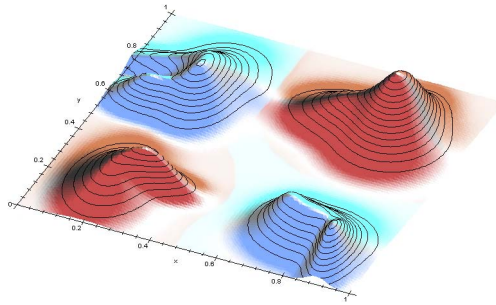
## 2 Related Work

**Statistical Modeling of Data in Sensor Networks.** In [1], a new probabilistic abstract datatype (ADT) based on a one-dimensional Gaussian is introduced. In order to handle large amounts of data, the authors store the Gaussian ADT data in an ORDBMS. For query processing, a two-stage process based on linear constraints is applied. However the GADT approach considers neither a mixture of probability density functions

(pdf) nor multivariate probability distributions. The basic idea in [4] is to build a probabilistic model from both historical and current sensor readings and to query the model instead of the actual sensor network. The authors present an architecture, called BBQ, that realizes this idea by a model based on time-varying multivariate Gaussians. Users then submit one-shot queries to a relational database to request real-time information about the network. However, the authors do not discuss the usage of index structures for efficient query processing. The authors of [19] propose to use probabilistic models for acquisitional settings such as sensor networks and Internet monitoring. Such a probabilistic model enables them to define new query types, exploit correlations when answering queries and to provide probabilistic guarantees on the correctness of the answers. While it is possible to use many different types of models, the authors explicitly describe the time-varying multivariate Gaussian model as an example. However, the authors do not consider an index structure for query processing.

**Spatial Uncertainty.** Authors of [2,3,5,20] deal with an uncertainty model for spatially uncertain objects and propose queries which are specified by intervals in the query space. In this setting a query retrieves uncertain objects w.r.t. the likelihood that the uncertain object is indeed placed in the given query interval. Let us consider the contributions more exactly. In [2] a new uncertainty model is introduced and several new types of queries are described that allow the handling of inexact data. [5] introduced the U-Tree for indexing uncertain 2D objects. Authors of [20] adapt the Gauss-tree proposed in [6] to spatial uncertainty model and discuss *probabilistic ranking queries*. The authors of [21] address the evaluation of probabilistic queries over uncertain data in constantly-evolving environments and discuss among others the so-called *probabilistic nearest neighbor query* that retrieves the set of objects being closest to a query object. All of these contributions employ the so-called interval uncertainty model and pose queries based on intervals in the data space. Thus, the mentioned approaches rather deal with spatial uncertainty (i.e., the location of an object is considered to be uncertain). Besides the mentioned methods for indexing spatially uncertain objects, [22] introduces *existential uncertainty*. The idea of this approach is that the existence of each data object is uncertain.

**Uncertainty in Object Identification.** In [6], the Gauss-tree is introduced which is an index structure for managing large amounts of Gaussian distribution functions. The proposed system aims at efficiently answering so-called identification queries. Additionally, [6] proposed *probabilistic identification queries* which are based on a Bayesian setting (i.e., given a query pdf, retrieve those pdfs in the database that correspond to the query pdf with the highest probability). An example for this setting is: Given a facial image represented by a normal distribution, retrieve the person in the database whose face corresponds to the query image with the highest probability. However, the methods in [6] underly the limitation that each object can only be described by a single, axis-parallel Gaussian. In this paper, we approximate arbitrary probability distributions that are approximated by GMMs. Furthermore, unlike [6] our method facilitates unsuccessful search which is an important feature for several application areas (e.g., the content-based retrieval of multimedia data).



**Fig. 3.** Two uncertain objects (i.e., (dark) red object and (light) blue object) modeled by GMMs

**Indexing Techniques for Uncertain Objects.** The indexing methods introduced in [3,5,6] have serious problems that strongly limit their use for applications with objects modeled by GMMs. The first limitation of the introduced uncertainty models is that each object is represented as a single multivariate distribution. Though this type of pdf is rather common, it is for many of the above mentioned applications too simplifying. Figure 3 illustrates two uncertain objects modeled by GMMs (object A in blue and object B in red) in order to exemplify the problem. Each probabilistic object consists of two distinct sets of Gaussians that are placed on a diagonal in oppositional edges of feature space. The indexing techniques described in [3,5,6] would approximate each object with a node  $N$  that ranges over whole feature space (i.e., from 0 to 1 in each dimension). Therefore, the selectivity of query processing algorithms is very low (i.e., we should access all available nodes of an index structure while performing a probabilistic query). Additionally, several of the existing methods (e.g., [3,6]) are based on distributions that do not consider any correlation between the dimensions of the underlying feature space which is a rather problematic assumption even for application were Gaussians are rather well-suited. Last, but not least, the models derive probabilities based on the assumption that the object which is searched for is always stored in the database. If the query object is unknown, the model does not necessarily calculate a small probability because the probability is relative to the total probability that the query belongs to at least one object in the database. Recently, [23] introduced a method and a corresponding index structure modeling pdfs using piecewise-linear approximations. This new approach also employs linear functions as the U-Tree but is more exact in its approximation.

**GMMs in Multimedia Retrieval.** In our experimental evaluation, we test our general approach of indexing arbitrary probability functions on a database of video clips being represented by GMMs. To demonstrate that this method is competitive with existing approaches to video retrieval, we will sketch some relevant work in this area. In [9], a summarization technique is presented which is based on GMMs. However, the authors do not propose any technique for speeding up the search on these summarizations. Furthermore, the named approach yields a specialized solution for video retrieval and is thus not concerned with the efficient search in general probabilistic data sets. The authors of [24] propose an approach for obtaining a compact representation of videos that computes the optimal representatives by minimizing the Hausdorff distance between

the original video and its representation. If the Euclidian metric is used,  $k$ -means can be applied to summarize video clips [25] by a set of centroids. A randomized technique for summarizing videos, called video signature, is proposed in [26]. A video sequence in the database is described by selecting a number of its frames closest to a set of random vectors. The authors of [26] also propose a specialized distance function on the derived summarization vectors. However, to the best of our knowledge, none of these techniques uses an index structure to accelerate query processing.

### 3 Probabilistic Similarity Search Using a Mixture of Gaussians

#### 3.1 Gaussian Mixture Model (GMM)

We assume that our objects are given by a probability density function (pdf) over a  $d$ -dimensional feature space  $\mathbb{R}^d$ . The pdf is defined in terms of a mixture of Gaussians, as specified in the following definition:

**Definition 1 (Gaussian Mixture Model).** *Let  $x \in \mathbb{R}^d$  be a variable from a  $d$ -dimensional feature space. A Gaussian mixture model (GMM)  $G_O$  of an object  $O$  is the following probability density function corresponding to a weighted sum of  $k_O$  Gaussian functions:*

$$G_O(x) = \sum_{1 \leq i \leq k_O} W_{O,i} \cdot N_{O,i}(x) \tag{1}$$

where  $W_{O,i}$  is a weight factor  $0 \leq W_{O,i} \leq 1$  with

$$\sum_{1 \leq i \leq k_O} W_{O,i} = 1 \tag{2}$$

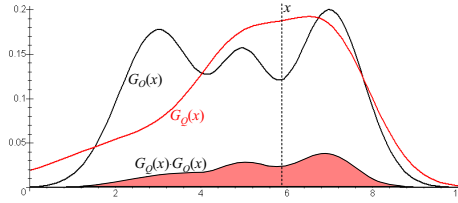
and  $N_{O,i}(x)$  is the density of a multivariate normal distribution:

$$N_{O,i}(x) = \prod_{1 \leq l \leq d} N_{\mu_{O,i,l}, \sigma_{O,i,l}^2}(x_l) = \prod_{1 \leq l \leq d} \frac{1}{\sqrt{2\pi\sigma_{O,i,l}^2}} \cdot e^{-\frac{(x_l - \mu_{O,i,l})^2}{2\sigma_{O,i,l}^2}}. \tag{3}$$

Note that we assume the attribute independence of the *single* Gaussians involved in a GMM (i.e., axis-parallel Gaussians). This assumption does not limit the generality of our approach, because it can be shown that a mixture of axis-parallel Gaussians is able to model arbitrary pdfs including those distributions where some of the attributes are strongly correlated to each other [8,7]. As an example, consider Figure 2 where a 2D Gaussian distribution with a strong correlation is modeled by a mixture of  $k = 5$  axis-parallel Gaussians. Thus, the attribute independence assumption of the single Gaussian does not imply an attribute independence of the pdf modeled by the complete GMM.

#### 3.2 Similarity of Objects Described by GMMs

After defining a GMM for describing an uncertain object, we now turn to the question how to determine the probability of an object  $O$  to be a *hit* for a query object  $Q$ . As a



**Fig. 4.** An example of two GMM in univariate space

starting point, we assume that the query  $Q$  is provided as by an exact feature vector (i.e., a point  $x \in \mathbb{R}^d$ ). In this case, Formula 1 can be used in combination with the Bayes theorem to determine this probability:

$$P(O|x) = \frac{P(O) \cdot G_O(x)}{\sum_{U \in DB} P(U) \cdot G_U(x)}.$$

Since it makes sense to assume that each database object is part of the answer with the same prior probability,  $P(O) = P(U)$  for all  $O, U \in DB$  and thus, we can cancel out the prior probability from the formula. Please note that the use of probability densities instead of probabilities is common practice in Bayes classification, as can be verified in [27][28].

The situation becomes more complex if both data and query object ( $O$  and  $Q$ ) are given as a GMM. We have to find a probability measure which indicates the probability that both GMMs  $O$  and  $Q$  correspond to the same unknown *true* object  $x$  (which is a traditional vector, and is used as a stochastic variable here). In general, we know nothing about  $x$  except that it is taken from  $\mathbb{R}^d$ . For every position of  $x$  in this data space, we can determine the probability density of  $O$  and  $Q$ , respectively. Both are given by a pdf modeled as a GMM. But for every position  $x$ , we can also determine the probability density with which  $x$  belongs to both  $O$  and  $Q$ . As belonging to  $O$  and belonging to  $Q$  are independent events, this is simply the product of the two corresponding pdfs. Since the true object  $x$  can be any point of  $\mathbb{R}^d$ , we have to form a  $d$ -dimensional volume integral over all possible positions of  $x$  to determine the overall probability that  $O$  and  $Q$  correspond to the same unknown  $x$ .

A one-dimensional example is visualized in Figure 4. Two objects,  $Q$  and  $O$ , each modeled as a mixture of  $k = 3$  Gaussians are depicted. We have also indicated  $x$ , a possible position of the unknown true object. We can see from the diagram the probability density with which  $x$  belongs to  $O$  (approximately 0.12), to  $Q$  (0.18) and to both of them ( $0.12 \cdot 0.18 \simeq 0.02$ ). Since all  $x \in \mathbb{R}$  are possible, the overall probability of  $O$  and  $Q$  to belong to the same object corresponds to the integral of  $G_Q(x) \cdot G_O(x)$  where  $x = -\infty.. + \infty$  (i.e., the shaded area in Figure 4). We call this probability density  $p(Q|O)$  with

$$p(Q|O) = \iint_{-\infty}^{+\infty} G_O(x) \cdot G_Q(x) dx. \tag{4}$$

The theorem of Bayes can again be used to determine the result probability of every database object  $O$  like in the case of single-valued query objects (cf. Equation 3.2):

$$P(O|Q) = \frac{p(Q|O)}{\sum_{P \in DB} p(P|O)}. \quad (5)$$

We follow the convention to use the abbreviation  $\iint_{-\infty}^{+\infty} f(x) d\mathbf{x}$  for the so-called volume integral  $\int_{-\infty}^{+\infty} \dots \int_{-\infty}^{+\infty} f(x) dx_1 \dots dx_d$ . Note that, although only two integral signs are written, they actually stand for a  $d$ -fold integral. In the following, we show how the overall probability density  $p(Q|O)$  defined in Formula 4 can be computed analytically:

$$\begin{aligned} p(Q|O) &= \iint_{-\infty}^{+\infty} G_O(x) \cdot G_Q(x) d\mathbf{x} \\ &= \iint_{-\infty}^{+\infty} \left( \sum_{1 \leq i \leq k_O} W_{O,i} N_{O,i}(x) \right) \left( \sum_{1 \leq j \leq k_Q} W_{Q,j} N_{Q,j}(x) \right) d\mathbf{x} \\ &= \sum_{1 \leq i \leq k_O} \sum_{1 \leq j \leq k_Q} W_{O,i} \cdot W_{Q,j} \cdot \iint_{-\infty}^{+\infty} N_{O,i}(x) \cdot N_{Q,j}(x) d\mathbf{x}. \end{aligned}$$

The volume integral can be solved by the following re-arrangement of terms:

$$\iint_{-\infty}^{+\infty} N_{O,i}(x) \cdot N_{Q,j}(x) d\mathbf{x} = \iint_{-\infty}^{+\infty} \prod_{1 \leq l \leq d} N_{\mu_{O,i,l}, \sigma_{O,i,l}}(x) \cdot N_{\mu_{Q,j,l}, \sigma_{Q,j,l}}(x) d\mathbf{x}$$

because of attribute independence

$$\begin{aligned} &= \prod_{1 \leq l \leq d} \int_{-\infty}^{+\infty} \frac{e^{-\frac{(x_l - \mu_{O,i,l})^2}{2\sigma_{O,i,l}^2}} \cdot e^{-\frac{(x_l - \mu_{Q,j,l})^2}{2\sigma_{Q,j,l}^2}}}{2\pi\sigma_{O,i,l}\sigma_{Q,j,l}} dx_l \\ &= \prod_{1 \leq l \leq d} \frac{1}{\sqrt{2\pi(\sigma_{O,i,l}^2 + \sigma_{Q,j,l}^2)}} \cdot e^{-\frac{(\mu_{Q,j,l} - \mu_{O,i,l})^2}{2(\sigma_{O,i,l}^2 + \sigma_{Q,j,l}^2)}} \\ &\quad \cdot \int_{-\infty}^{+\infty} N_{\mu_{O,i,l}, \sigma_{O,i,l}}(x_l) \cdot \frac{\sigma_{O,i,l}^2 \sigma_{Q,j,l}^2}{\sigma_{O,i,l}^2 + \sigma_{Q,j,l}^2} (x_l) dx_l \\ &= \prod_{1 \leq l \leq d} \frac{1}{\sqrt{2\pi(\sigma_{O,i,l}^2 + \sigma_{Q,j,l}^2)}} \cdot e^{-\frac{(\mu_{Q,j,l} - \mu_{O,i,l})^2}{2(\sigma_{O,i,l}^2 + \sigma_{Q,j,l}^2)}} \cdot 1 \\ &= \prod_{1 \leq l \leq d} N_{\mu_{O,i,l}, \sigma_{O,i,l}^2 + \sigma_{Q,j,l}^2}(\mu_{Q,j,l}). \end{aligned}$$

Therefore, the overall probability density  $p(Q|O)$  corresponds to

$$p(Q|O) = \sum_{\substack{1 \leq i \leq k_O \\ 1 \leq j \leq k_Q}} W_{O,i} W_{Q,j} \prod_{1 \leq l \leq d} N_{\mu_{O,i,l}, \sigma_{O,i,l}^2 + \sigma_{Q,j,l}^2}(\mu_{Q,j,l}).$$

### 3.3 Handling Unknown Objects

In many applications, we have to consider the case that the query object is not necessarily stored in the database. Thus, an algorithm should allow that the result of the query is empty (i.e., it is most likely that the given query does not fit to any of the database objects). In other words, no object can be considered as similar at all. Our solution to this problem is to introduce an additional probability distribution modeling the likelihood of an unknown data object.

When applying the Bayes theorem (cf. the Equation 5), we make the implicit assumption that the object which is searched for is always stored in the database, and that all stored database objects have the same *a priori* probability of being the result of the search. Therefore, the *sum* of all result probabilities of all database objects equals 1 for every query. If a query object is at a position of extremely low density, then some of the database objects (particularly those with high overall variance) still may obtain high result probabilities, which contradicts the intuition. In terms of the Bayesian probability model, the conditional probability might become rather large if the total probability in the denominator is very small. Since a conditional probability is normalized by the total density, a small probability density might lead to a large probability if the total density is very small.

This problem can be solved by modeling a placeholder for those objects which are not stored in the database. This placeholder should have a high variance to cover the complete data space and a prior probability  $P_{PH}$  corresponding to the expected rate of unsuccessful searches. For example, if 3 out of 4 search operations are not successful (the intended object is not found in the database), then  $P_{PH} = 0.75$ , and the remaining prior probability (0.25) is shared by the database objects. The value of  $P_{PH}$  can be set at the begin of database usage to a default value. The value of  $P_{PH}$  should be adapted when collecting exact statistics about unsuccessful queries during database deployment.

The probability distribution of the placeholder can e.g. be selected to be uniformly or normally distributed. Assuming that the data distribution of the unknown objects follows that of the known objects, we can use the mean and the variance of the object set. Let us not that these should not be confused with the intra-object variance describing the uncertainty of the feature value of a *single* object:

$$\mu_{PH,l} = \frac{1}{|DB|} \sum_{O \in DB} \sum_{1 \leq i \leq k_O} W_{O,i} \cdot \mu_{O,i,l}$$

$$\sigma_{PH,l}^2 = \frac{1}{|DB| - 1} \sum_{O \in DB} \sum_{1 \leq i \leq k_O} W_{O,i} \cdot (\mu_{O,i,l} - \mu_{PH,l})^2.$$

To consider the placeholder, we have to extend our formula for calculating  $P(O|Q)$  in the following way:

$$P(O|Q) = \frac{\frac{1-P_{PH}}{|DB|} \cdot p(Q|O)}{P_{PH} \cdot P(Q|PH) + \sum_{P \in DB} \frac{1-P_{PH}}{|DB|} \cdot p(P|O)}. \quad (6)$$

### 3.4 Probabilistic Ranking Query on GMMs

A method for deciding containment in the query result is to retrieve the  $k$  most likely results. An example for this type of query is: Retrieve the 5 objects from the database having the highest probability for corresponding to the query object. We will call this type of query *probabilistic ranking query (PRQ)*. In the following we will formalize PRQs:

**Definition 2 (Probabilistic Ranking Query).** *Let  $DB$  be a database of GMMs  $M$ , let  $Q$  be a query GMM and let  $k \in \mathbb{N}$  be a natural number. Then, the answer to a probabilistic ranking query (PRQ) on  $DB$  is defined as the smallest set  $RQ_k(Q) \subseteq DB$  with at least  $k$  elements fulfilling the following condition:*

$$\forall M_a \in RQ_k(Q), \forall M_{db} \in DB \setminus RQ_k(Q) : P(M_a|Q) > P(M_{db}|Q)$$

## 4 Indexing Mixtures of Gaussians

### 4.1 General Idea

Our method is based on the idea of decomposing each GMM into its components (i.e., the single Gaussians, and index the components separately). For instance, in order to store a univariate GMM with  $W_1 = 0.3, W_2 = 0.7, \mu_1 = 0, \mu_2 = 1, \sigma_1 = 2, \sigma_2 = 3$ , we decompose the GMM in two components  $g_1$  and  $g_2$ . The first component  $g_1$  is described by  $W_1 = 0.3, \mu_1 = 0, \sigma_1 = 2$ . The second component  $g_2$  is described by  $W_2 = 0.7, \mu_2 = 1, \sigma_2 = 3$ . After decomposition, we can store two components (i.e.,  $g_1$  and  $g_2$ ) separately in an index that is appropriate for single pdfs. During query processing, the complete GMMs can be processed componentwise and thus, we can start excluding objects from the result set, even without retrieving the complete GMM.

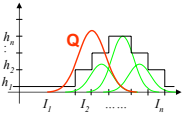
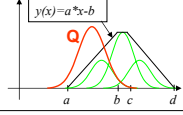
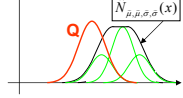
To index the components of the stored GMMs, our method can employ any hierarchically organized index structure  $\Delta$  that is capable to store single pdfs (e.g., [35][6] that are based on the R-Tree [29] principle). Since  $\Delta$  is designed to handle single pdf, we have to extend it to additionally store the weights for each component. Thus, an entry which corresponds to a GMM component  $O_i$ , consists of the parameters of the underlying Gaussian  $\mu_{O,l}$  and  $\sigma_{O,l}^2$  ( $1 \leq l \leq d$ ) and the weight of the component  $W_{O,i}$ . This way, it is possible to reconstruct the complete GMM after retrieving all of its components from the index structure  $\Delta$ . In addition to extending the entries in the leaf nodes, we have to extend the node descriptions by the maximum weight  $W_{max}^P$  of any Gaussian being stored in the corresponding subtree.

The key idea of our proposed query algorithms is that it is possible to calculate the probability  $P(O|Q)$  componentwise. Thus, if we can calculate a conservative approximation of the components of  $P(O|Q)$ ,

$$comp(O, i, Q, j) = W_{O,i} W_{Q,j} \prod_{1 \leq l \leq d} N_{\mu_{O,i,l}, \sigma_{O,i,l}^2 + \sigma_{Q,j,l}^2}(\mu_{Q,j,l})$$

it is possible to calculate conservative approximations for completely and partially retrieved GMMs during query processing.



Index Structure	Idea of Approximation (P - Directory Node)	Sketch of Formula for $N_{LB}(P, Q, j)$
Approach of [R. Cheng, Y. Xia, S. Prabhakar et al.]		$N_{LB}(P, Q, j) = \sum_{k=1}^n \int_{I_{k-1}}^{I_k} (N_{Q,j}(x) \cdot h_k) dx$
U-tree		$N_{LB}(P, Q, j) = \int_a^b (N_{Q,j}(x) \cdot y(x)) dx + \int_b^c \dots + \dots$
Gauss-tree		$N_{LB}(P, Q, j) = \int_{-\infty}^{+\infty} (N_{Q,j}(x) \cdot \hat{N}_{\mu, \mu, \sigma, \sigma}(x)) dx$

**Fig. 5.** Idea of the approximation and sketch of the conservative approximation calculation ( $N_{LB}(P, Q, j)$ ) for different index structures that handle single pdfs

Therefore, we define a conservative approximation for any component  $Q_i$  and a given page  $P$  in the index structure  $\Delta$  which maximizes  $comp(O, i, Q, j)$  over all  $O_i$  that are stored in the subtree corresponding to  $P$ :

$$\begin{aligned}
 comp_{max}^P(Q, j) &= W_{max}^P W_{Q,j} \prod_{1 \leq l \leq d} N_{LB}(P, Q, j) \\
 &\geq \max_{O_i \in P} W_{O,i} W_{Q,j} \prod_{1 \leq l \leq d} N_{\mu_{O,i,l}, \sigma_{O,i,l}^2 + \sigma_{Q,j,l}^2}(\mu_{Q,j,l})
 \end{aligned}$$

where  $N_{LB}(P, Q, j)$  is lower bound or maximum of probability density that can be achieved in a node or subtree of the underlying index structure and  $W_{max}^P$  is the maximum of the weights of all components stored in  $P$ . In particular,  $N_{LB}(P, Q, j)$  can be calculated by:

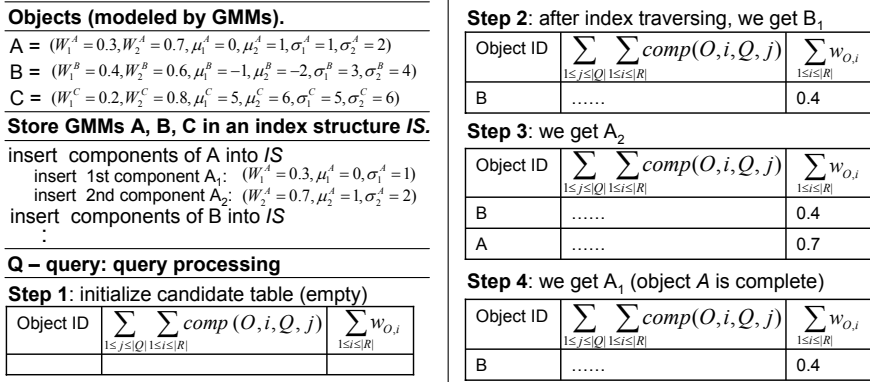
- evaluation of one or several entries in so-called *ratio table* w.r.t.  $x$ -bounds if employing the indexing technique described in [3],
- using the so-called *U-Catalog* and the function  $e.MBR(.)$  that allows pruning subtrees if employing the U-tree [5],
- using the density of the *hull function*  $\hat{N}...$  if employing the Gauss-tree [6].

Figure 5 depicts the general idea of the computation of the conservative approximation  $N_{LB}(P, Q, j)$  for the above mentioned index structures.

Furthermore, we define  $comp_{max}^P(Q)$  approximating the probability for the complete query GMM  $Q$  and an arbitrary component stored in  $P$  as:

$$comp_{max}^P(Q) = |Q| \cdot \max_{1 \leq j \leq |Q|} comp_{max}^P(Q, j)$$

Thus, we have found a way to estimate the maximum influence of any GMM component stored in page  $P$  on the probability  $P(O|Q)$  for any object  $O$  in the database



**Fig. 6.** General idea of the candidate table used in our query algorithms on GMMs

that has not yet been retrieved completely. For query processing,  $comp_{max}^P(Q)$  is useful for several tasks. First of all, we can use  $comp_{max}^P(Q)$  to estimate the maximum probability  $P(O'|Q)$  for the GMM  $O'$  which is stored in  $P$  and there is no component of  $O'$  that has been retrieved during query processing yet.

$$P(O'|Q) = k_{max} \cdot comp_{max}^P(Q)$$

where  $k_{max}$  denotes the maximum number of components in any GMM in the database. Furthermore, we can approximate the maximal probability of a partially retrieved GMM  $O$  to match  $Q$ . This is important for pruning candidate GMMs as early as possible:

$$P(O|Q) = \sum_{1 \leq j \leq |Q|} \sum_{1 \leq i \leq |R|} comp(O, i, Q, j) + (1.0 - \sum_{1 \leq i \leq |R|} w_{O,i}) \cdot comp_{max}^P(Q)$$

where  $R$  is the set of already retrieved components of  $O$  and  $P$  is the page having the maximum value of  $comp_{max}^P(Q)$ . Note, that the use of  $1.0 - \sum_{1 \leq i \leq |R|} w_{O,i}$  for all remaining weights is justified by Formula 2. The estimation of the sum of all densities is necessary to apply Formula 6. We can sum up the densities or their conservative approximations as described above during query processing. Thus, it is possible to determine a conservative approximation of the complete density of a candidate GMM at all times.

Before we explain our probabilistic query processing on GMMs in detail, we take a closer look at the data structure which manages the result candidates. This candidate table can either be located in main memory or on secondary storage, depending on the size of a particular dataset. Every component of a GMM which is fetched from the underlying index structure is stored in this table. An object is removed from the table under two conditions. Either all components of a particular object are in the candidate table (cf. Figure 5) or the object is pruned based on the conservative approximation of the object probability.

```

ALGORITHM PRQ(Query Q, integer  $k$ )
BEGIN
   $hits := \text{new PriorityQueue}(\text{ascending})$ 
   $drops := \text{new List}()$ 
   $candidates := \text{new List}()$ 
   $APL := \text{new PriorityQueue}(\text{descending})$ 
   $APL.insert(\text{root}, 1 - P_{PH})$ 
  REPEAT
     $currNode := APL.removeFirst()$ 
    IF  $currNode.isDirectoryNode() = 'TRUE'$  THEN
      FOR EACH  $node \in currNode$  DO
         $APL.insert(node, comp_{max}^{node}(Q))$ 
      END FOR
    IF  $currNode.isDataNode() = 'TRUE'$  THEN
      FOR EACH  $c \in currNode$  DO
        IF  $drops.contains(c.ObjectID) = 'TRUE'$  THEN
          CONTINUE
        END IF
         $candidates.update(c.ObjectID, c, Q)$ 
         $entry := candidates.get(c.ObjectID)$ 
        IF  $entry.isComplete() = 'TRUE'$  THEN
           $prob := entry.probability(Q)$ 
          IF  $prob \geq hits.topProbability$  THEN
            IF  $hits.Size() = k$  THEN
               $hits.removeFirst()$ 
            END IF
             $hits.add(c.ObjectID, prob)$ 
          END IF
           $candidates.delete(c.ObjectID)$ 
        ELSE
          IF  $entry.approximation(Q) \leq$ 
             $hits.topProbability$  THEN
             $drops.add(c.ObjectID)$ 
             $candidates.delete(c.ObjectID)$ 
          END IF
        END IF
      END FOR
    END IF
  UNTIL  $(candidates.Size() > 0$ 
    or  $APL.topProbability > hits.topProbability$ 
    and  $APL.Size() > 0)$ 
  report  $hits$ ;
END

```

**Fig. 7.** Pseudo code: Probabilistic Ranking Query on GMMs

In the following we will introduce our algorithm for processing PRQs. For this type of query, the minimum probability for a result depends on the object having the  $k$ -highest probability for corresponding to the query GMM  $Q$ . Therefore, we employ two priority queues: The first priority queue is used for traversing the probabilistic index structure storing the components of the GMMs. We will refer to this queue as the traversal queue. The second priority keeps those  $k$  GMMs which currently have the largest probabilities for corresponding to  $Q$ . We will sort this second priority queue in ascending order and refer to it as the result queue. The pseudo code for the algorithm is displayed in Figure 7. We start by ordering the descendant nodes of the root page w.r.t.  $comp_{max}^p(Q)$ . Afterwards we enter the main loop of the algorithm and remove the top element of the traversal queue. If this element is a node, we load its child nodes. If these child nodes are nodes themselves, we determine  $comp_{max}^p(Q)$  and update the traversal queue. If the child nodes are components of GMMs, we check the candidate table for

a corresponding GMM  $M$  and insert a new descriptor, in the case that there is not already a descriptor for  $M$ . Afterwards, we can update the candidate table as mentioned before. If a GMM  $M$  has been read completely, we can delete it from the candidate table and compare its probability  $P(M|Q)$  to the probability of the top element of the result queue (i.e., the GMM encountered so far having the  $k$  highest probabilities). If the probability of  $M$  is higher than that of the top element, we need to add  $M$  to the queue. However, to make sure that we do not retrieve more than  $k$  elements, we have to check the size of the result queue. If there are already  $k$  elements, we have to remove the previous top element before inserting  $M$ . In the case, that the entry in the candidate table does not contain the complete information about  $M$  yet, we still can calculate a probability estimation and compare it to the top element of the result queue. If  $P(M|Q)$  is smaller than the  $k$  highest probability in the result queue, we can guarantee that  $M$  is not a potential result. Thus,  $M$  is deleted from the candidate table and stored in our list for excluded GMMs. The algorithm terminates if the top of the traversal queue provides a lower value than the top of the result queue and the candidate table is empty. Please note that it is not necessary to calculate any complex integral functions during query processing. As shown in Section 3.2 it is sufficient to compute a weighted product of probability densities during query execution.

## 5 Experimental Evaluation

To demonstrate the effectiveness and efficiency of our new approach, we implemented the proposed methods in Java 1.5. All experiments were performed on a workstation equipped with a 2.2 GHz Opteron CPU and 8GB RAM. In order to store components of GMMs, we implemented an X-Tree [30] variant that uses split and insert methods as proposed in [6], and the technique for calculation of conservative approximations on Gaussians (i.e., for calculation of  $N_{LB}$  in Formula 7 of Section 4) as described in [6].

**Testbed.** In order to demonstrate, that our new uncertainty model yields an competitive solution for content-based object retrieval in multimedia databases, we tested our uncertainty model based on GMMs on a dataset of 1,050 music video clips. The average length of a video is 4 minutes 14 seconds. To transform the videos into GMMs, we first of all extracted the set of all frames contained in a video and transformed each frame into a color histogram. Thus, each video was represented by a set of 3,000 up to 10,000 single images. To model these set of images as a GMM, we applied Expectation Maximization clustering. Afterwards each video was represented by a GMM consisting of 100 Gaussians. Let us note that some of the videos are described by a smaller number of Gaussians because the clustering generated empty clusters without any weight in the GMM. Thus, the testbed consists of about 100,000 16-dimensional Gaussians which correspond to approximately 150 GB of video data. To demonstrate that our new uncertainty model is competitive to other methods for content-based video retrieval, we additionally generated a database representing each video as a set of color histograms. However, since using thousands of feature vectors for representing a video clip of 3 to 5 minutes is not feasible, we had to reduce the number of employed feature vectors considerably. Thus, each video is described by selecting every 50th frame in chronological order. To query a database describing a video as set of feature vectors, we employ

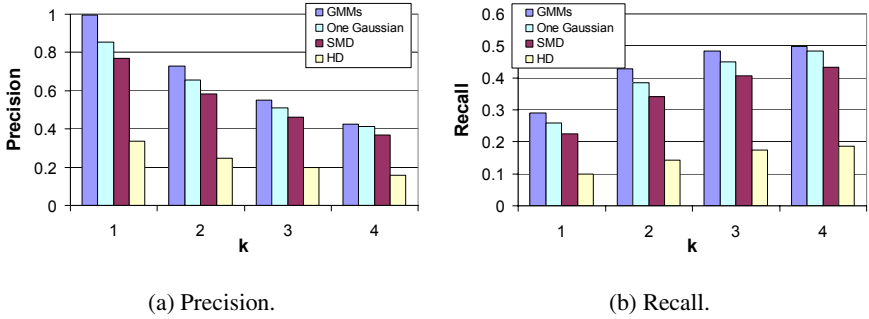


Fig. 8. Precision and recall for PRQ on GMMs and comparison partners

well known distance functions for set-valued objects like the sum of minimum distances (SMD) and the Hausdorff distance (HD) [31]. Furthermore, we generated a third database representing each data object as a single Gaussian, instead of a GMM.

**Effectiveness.** We compare the results of our new method to the mentioned comparison partners for video retrieval w.r.t. precision and recall. Therefore, we posed ranking queries, subsequently retrieving the 4 objects in the database that most likely match the query object. The queries consist of 40 videos for which there exist multiple versions in the database corresponding to different versions or having a different recording quality. Thus, we can measure precision and recall for these query videos. For each size of the result set ( $k = 1, \dots, 4$ ), we measured precision and recall. Figure 8 displays the precision and the recall achieved by all 4 compared methods. Our new approach for modeling uncertain objects as GMMs outperformed all other methods w.r.t. precision as well as recall. For example, the GMM based model displayed a more than 20 % better precision for  $k = 1$  than the best of its comparison partners. Thus, we can state that modeling a video as a GMM instead of a single Gaussian significantly increased the quality of object representation and therefore, the precision and the recall of the result sets improved as well. Furthermore, the result indicates that modeling videos as GMMs is superior to modeling videos as sets of feature vectors.

We additionally tested the capability, of our new method to cope with query objects that are not stored in the database. The result indicates a probability for unsuccessful queries that is less than 6%. Since such a small probability value is not observed for any result of a successful search, we assume an unsuccessful search for result probabilities of less than 10 %.

**Efficiency.** After demonstrating the usefulness of our new approach for uncertain objects, we now examine the efficiency of our new method to index GMMs. To the best of our knowledge, the method proposed in this paper is the first approach for efficiently managing large sets of GMMs. Thus, we compare our method to the basic approach that no index is available and the result set has to be determined by a sequential scan over the database. Since measuring real disc accesses is often not significant due to the existence of disc caches, we measure the IO performance in this section by counting IO operations and afterwards add up the cost for the counted disk accesses based

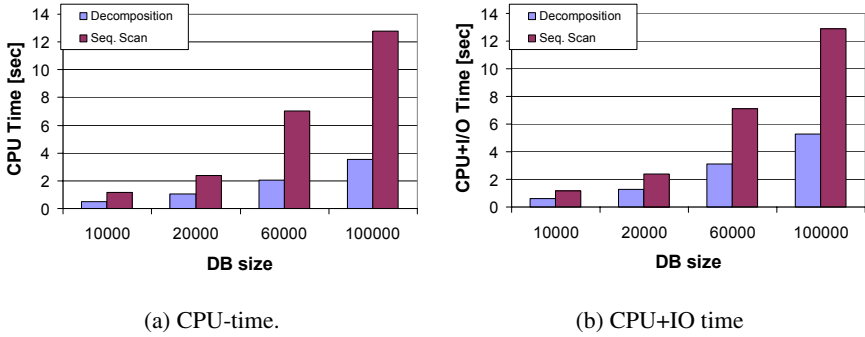


Fig. 9. Efficiency for PRQ ( $k=3$ ) on synthetic data for varying database size

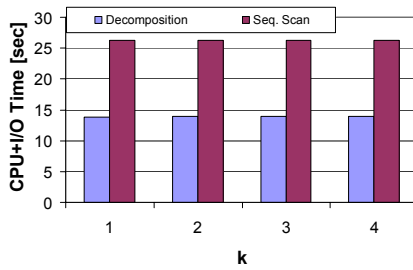


Fig. 10. CPU+IO time for PRQ on the video dataset

on a transfer rate of 50 MB/s and an access time of 6 ms. We evaluated the efficiency of our new approach on the real world video data set. The result is displayed in Figure 10. For all query parameters our new PRQ method was capable to speed up query processing to a factor of approximately 2. Furthermore, the query time was comparably stable for the given query parameters, e.g. the query time did not significantly increase for larger values of  $k$ . Since the video data set is rather small, we generated an artificial data set of up to 100,000 GMMs to examine the scalability of our approach. These GMMs consisted of up to 10 Gaussians over an 2D data space. The results are displayed in Figure 9. In this figure, we distinguish between the CPU query time (cf. Figure 9(a)) and the complete query time (cf. Figure 9(b)). The speedup of our new approach increased with the size of the database. Thus, using our approach is more beneficial for larger data sets.

## 6 Conclusion

In this paper, a new uncertainty model is proposed which represents uncertain objects by Gaussian mixture models (GMM). A GMM can approximate arbitrary multivariate probability distribution and thus, GMMs are applicable in a wide variety of new applications like multimedia retrieval, sensor networks, medicine, geology. To use GMMs for

query processing, we demonstrate that the probability that a query GMM corresponds to a database GMM can be calculated analytically. Furthermore, our model is capable of handling unsuccessful queries (i.e., if the database does not contain a similar data object the results will have a very low result probability). Thus, we can recognize that there is most likely no corresponding object in the database. After introducing the uncertainty model, we examine probabilistic ranking queries as an important query type. To speed up these queries, we propose a new indexing technique which is based on object decomposition. The key idea of this method is to store each component of each GMM separately in an index structure for single pdfs like [3,5,6]. During query processing, we can prune certain GMMs based on their already retrieved components. Additionally, the result GMMs are reconstructed while searching the database. Thus, our new method is capable of efficiently retrieving the GMMs in the database that resemble a query GMM with maximum likelihood. In our experimental evaluation, we demonstrate the effectiveness of our new approach for content-based multimedia retrieval on a data set of 1,050 video clips. Additionally, we show that the new query algorithm employing object decomposition yields a significant speed up compared to sequential query processing on a synthetic data set and the named data set of video clips.

## References

1. Faradjian, A., Gehrke, J., Bonnet, P.: GADT: A Probability Space ADT For Representing and Querying the Physical World. In: Proc. 18th Int. Conf. on Data Engineering (ICDE'02), San Jose, CA, USA p. 201 (2002)
2. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluating Probabilistic Queries over Imprecise Data. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'03), San Diego, CA, USA pp. 551–562 (2003)
3. Cheng, R., Xia, Y., Prabhakar, S., Shah, R., Vitter, J.S.: Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data. In: Proc. 30th Int. Conf. on Very Large Data Bases (VLDB'04), Toronto, Canada pp. 876–887 (2004)
4. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J., Hong, W.: Model-driven data acquisition in sensor networks. In: Proc. 30th Int. Conf. on Very Large Data Bases (VLDB'04), Toronto, Canada (2004)
5. Tao, Y., Cheng, R., Xiao, X., Ngai, W.K., Kao, B., Prabhakar, S.: Indexing Multi-Dimensional Uncertain Data with Arbitrary Probability Density Functions. In: Proc. 30th Int. Conf. on Very Large Data Bases (VLDB'05), Trondheim, Norway, pp. 922–933. (2005)
6. Böhm, C., Pryakhin, A., Schubert, M.: The Gauss-Tree: Efficient Object Identification of Probabilistic Feature Vectors. In: Proc. 22nd Int. Conf. on Data Engineering (ICDE'06), Atlanta, GA, US, p. 9 (2006)
7. Titterton, D.M., Smith, A.F.M., Makov, U.E.: Statistical analysis of finite mixture distribution. Wiley, New York (1985)
8. Lindsay, B.G.: Mixture models: Theory, geometry, and applications (1995)
9. Greenspan, H., Goldberger, J., Mayer, A.: A probabilistic framework for spatio-temporal video representation & indexing. In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds.) ECCV 2002. LNCS, vol. 2350, pp. 461–475. Springer, Heidelberg (2002)
10. Yang, M., Ahuja, N.: Gaussian mixture model for human skin color and its application in image and video databases. In: Proc. of the Conf. on Storage and Retrieval for Image and Video Databases (SPIE 99), vol. 3656, pp. 458–466. Springer, Heidelberg (1999)

11. Chen, S.-C., Kashyap, R.L., Ghafoor, A.: *Semantic Models for Multimedia Database Searching and Browsing*. Kluwer Academic Publishers, Dordrecht (2002)
12. Srinivasan, U., Nepal, N.: *Managing Multimedia Semantics*. IRM Press (2005)
13. Deb, S.: *Video Data Management and Information Retrieval*. Idea Group Publishing (2005)
14. Gavin, D.G., Hu, F.S.: Bioclimatic modelling using gaussian mixture distributions and multiscale segmentation. *Global Ecology and Biogeography* 14, 491 (2005)
15. Lim, P., Quek, S., Peh, K.: Application of the gaussian mixture model to drug dissolution profiles prediction. *Neural Comput. Appl.* 14(4), 345–352 (2005)
16. Zajdel, W., Kröse, B.: Gaussian mixture model for multi-sensor tracking. In: *Proc. of the 15th Dutch-Belgian Artificial Intelligence Conference (BNAIC'03)*, pp. 371–378 (2003)
17. Reynolds, D.A., Quatieri, T.F., Dunn, R.B.: Speaker verification using adapted gaussian mixture models. *Digital Signal Processing* 10(1), 19–41 (2000)
18. Yoo, S.-H.: Application of a mixture model to approximate bottled water consumption distribution. *Applied Economics Letters* 10(3), 181–184 (2003)
19. Deshpande, A., Guestrin, C., Madden, S.R.: Using Probabilistic Models for Data Management in Acquisitional Environments. In: *Proc. CIDR* (2005)
20. Böhm, C., Pryakhin, A., Schubert, M.: Probabilistic Ranking Queries on Gaussians. In: *Proc. of the 18th Int. Conf. on Scientific and Statistical Database Management (SSDBM'06)*, pp. 169–178 (2006)
21. Cheng, R., Kalashnikov, D.V., Prabhakar, S.: Evaluation of Probabilistic Queries over Imprecise Data in Constantly-Evolving Environments 32(1), 104–130 (2007)
22. Dai, X., Yiu, M.L., Mamoulis, N., Tao, Y., Vaitis, M.: Probabilistic Spatial Queries on Existentially Uncertain Data. In: *Bauzer Medeiros, C., Egenhofer, M.J., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633*, pp. 400–417. Springer, Heidelberg (2005)
23. Ljosa, V., Singh, A.K.: APLA: Indexing arbitrary probability distributions. In: *Proc. of the 23rd Int. Conf. on Data Engineering (ICDE 2007)* (2007)
24. Chang, H.S., Sull, S., Lee, S.U.: Efficient Video Indexing Scheme for Content-Based Retrieval. In: *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, pp. 1269–1279. IEEE Computer Society Press, Los Alamitos (1999)
25. Zhuang, Y., Rui, Y., Huang, T.S., Mehrotra, S.: Adaptive key frame extraction using unsupervised clustering. In: *ICIP* (1), pp. 866–870 (1998)
26. Cheung, S.S., Zakhor, A.: Efficient video similarity measurement with video signature. In: *IEEE International Conference on Image Processing (ICIP 02)*, vol. 1, pp. 621–624. IEEE Computer Society Press, Los Alamitos (2002)
27. Han, J., M., K.: *Data Mining: Concepts and Techniques*. Morgan Kaufmann, San Francisco (2006)
28. Witten, I.H., E., F.: *Data Mining. Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco (2005)
29. Guttman, A.: R-trees: A Dynamic Index Structure for Spatial Searching. In: *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 47–57. ACM Press, New York (1984)
30. Berchtold, S., Keim, D.A., Kriegel, H.P.: The X-Tree: An Index Structure for High-Dimensional Data. In: *Proc. 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, Bombay, India, pp. 28–39 (1996)
31. Eiter, T., Mannila, H.: Distance measures for point sets and their computation. *Acta Informatica* 34(2), 103–133 (1997)



# Invisible Graffiti on Your Buildings: Blind and Squaring-Proof Watermarking of Geographical Databases\*

Julien Lafaye<sup>1</sup>, Jean Béguec<sup>1,3</sup>, David Gross-Amblard<sup>1,2</sup>, and Anne Ruas<sup>3</sup>

<sup>1</sup> Laboratoire CEDRIC, Spécialité Informatique – CC 432, Conservatoire national des arts & métiers, 292 rue Saint Martin, 75141 PARIS Cedex 3, France

<sup>2</sup> Laboratoire LE2I, Université de Bourgogne, Faculté des Sciences Mirande, Aile de l'ingénieur, BP 47870 21078 DIJON Cedex, France

<sup>3</sup> Laboratoire COGIT, Institut Géographique National (IGN), 2/4 Avenue Pasteur 94 165 SAINT MANDE Cedex, France

**Abstract.** Due to the ease of digital copy, watermarking is crucial to protect the intellectual property of rights owners. We propose an effective watermarking method for vectorial geographical databases, with the focus on the buildings layer. Embedded watermarks survive common geographical filters, including the essential squaring and simplification transformations, as well as deliberate removal attempts, e.g. by noise addition, cropping or over-watermarking. Robustness against the squaring transformation is not addressed by existing approaches. The impact on the quality of the datasets, defined as a composition of point accuracy and angular quality, is assessed through an extensive series of experiments. Our method is based on a quantization of the distance between the centroid of the building and its extremal vertex according to its orientation.

## 1 Introduction

Geographical Information Systems (GIS) have existed for more than 40 years but their application domain is much wider nowadays, ranging from environmental surveillance by country agencies to localization-aware services for individual mobile users. This phenomenon is stressed for the general public by the increasing availability of GPS devices (e.g. car navigation) and the recent development of Google Earth and GeoPortail [1]. Most geographical applications rely on an underlying vectorial spatial database (points, polylines and polygons).

Gathering such accurate information is an onerous task for the data owner. Hence, huge and detailed vectorial databases carry a high scientific and/or economical value. Due to the ease of reproduction of digital media, unauthorized copy and use threaten geographical data providers. Protecting the intellectual property (IP) of rights owner is a requirement.

---

\* Work supported by the ACI Sécurité & Informatique TADORNE grant (2004-2007).

On the legal side, data providers restrict the way buyers are allowed to use their data. On the technical side, robust watermarking is a known technique for IP protection. It consists of hiding a copyright mark within the dataset. Embedded marks must be robust against removal attempts to be useful. In this paper, we propose a robust watermarking method for polygonal datasets.

To embed the watermark, the data has to be altered. What might sound as a drawback is common to most watermarking methods [10]. There is a trade-off between watermark robustness and data alteration: the more alterations are allowed, the more robust the embedded watermark is. So, defining precisely what makes the value of a dataset is a prerequisite for watermarking.

Some applications do not rely only on spatial accuracy (i.e. the distance between a point in the real world and this point in the dataset). For example, spatial accuracy is not crucial for tourist city maps designers who apply strong transformations to road polylines and building polygons in order to increase legibility. Some others focus on objects like forests, cliffs and shallows for which precise borders can be difficult to define. But most applications rely on accurate data for automatic operations (e.g. service proximity search, GPS navigation, spatial analysis of risks, etc.). Accuracy can even be mandatory, e.g. for reefs locations on IHO/SHOM boat maps [22]. Finally, accurate datasets must conform with some standard reference system for interoperability purposes (e.g. the World Geodetic System – WGS84, which is the GPS reference system). So any watermarking method must respect data accuracy.

Beside accuracy, real world requirements entail specific constraints within the dataset. For example, it turns out that most of the vectorial content of geographical databases consists of building polygons (80% on the professional dataset used in the experiments). Building polygons are under the scope of the squaring constraint: data is systematically corrected so that buildings with right angles are mapped to polygons with right angles in the dataset. This *squaring* operation, available in any GIS, is systematically performed by data owners *and* data users, and increases the angular quality of the dataset. It is also very invasive since potentially each point of the dataset is moved. Experiments show that it also tends to increase data accuracy. But surprisingly, its impact has never been taken into account by existing watermarking proposals, e.g. [18,21]:

- On the owner side, watermarking is likely to turn squared shapes into skewed ones, reducing the data quality;
- On the user side, squaring is likely to wipe the watermark of the owner, lowering its security. This natural transformation, along with other common filters, can be interpreted as an attack on the watermark.

To take these effects into account, we model the quality of a dataset by means of (1) its accuracy and also (2) its angular quality. This choice is motivated likewise by a recently published survey [16], where it is deeply discussed that quality encompasses accuracy and must take into account shapes and topology.

In this paper, we propose an effective method for building watermarking that is robust against geographical transformations (including squaring and simplification) and attacks by malicious users. As far as we know, this is the first

method which takes into account the essential squaring transformation. It provides a high level of security while controlling the impact on the quality of the dataset (point accuracy and angular quality) and not introducing topological errors (overlapping polygons). An extended version of our method can even resist the MBR attack, that replaces each building by its minimum bounding rectangle. The scheme is blind: the original dataset is not required for detection, definitely an important feature for huge datasets.

A classical skeleton [2] of databases watermarking algorithms is to create a secret dependency between (1) a robust identifier of the data and (2) one of its characteristics, e.g. between the primary key of a tuple and one of its numerical attributes. Revealing this dependency acts as a proof of ownership. In our approach, we get rid of the primary key by constructing a robust identifier for each building using a well chosen portion of the highest significant bits of the coordinates of its centroid. Then, we rely on the observation that buildings have an intrinsic orientation and that most of their edges are parallel or perpendicular to this orientation. To hide a watermark bit, we expand or shrink buildings along their orientation. The expansion ratio is deterministically chosen among a set of quantized values according to the robust identifier of the polygon, the secret key of the owner and the bit to be embedded. By embedding the watermark within the shapes of a building rather than within the coordinates of its vertices, we achieve robustness of watermarks against squaring. Our scheme is also robust against other transformations we present later in the paper. Any malicious attacker has to tremendously reduce accuracy and/or angular quality of the dataset to erase the watermark.

*Outline.* After a description of watermarking basics, a simple model for buildings databases and a definition for data quality are presented in Section 2. Our watermarking procedure is described in Section 3. Correction, efficiency and robustness of the method are assessed in Section 4, through an extensive series of experiments. Related work is exposed in Section 5 and Section 6 concludes.

## 2 Preliminaries

### 2.1 Quality of Geographical Data

A point  $p = (x, y)$  is defined by its 2-dimension coordinates  $(x, y)$  in some reference system  $R_0$ . A simple polygon  $P = (p_1, \dots, p_n, p_{n+1} = p_1)$  is described by the list of its points. Two polygons taken from a real dataset are shown on Fig. 1(a). A geographical database instance is defined by  $(R, DB)$  where  $R$  is a reference system and  $DB = \{P_i\}$ ,  $i \in \{1, \dots, N\}$  is a set of  $N$  polygons. It is always provided with some reference system otherwise it is of no use for automatic operations.

We do not rely on the order of polygons within the dataset, nor on the order of points within a polygon. Furthermore, there is no primary key identifying these polygons. Polygons are supposed non-overlapping as in many geographical applications.

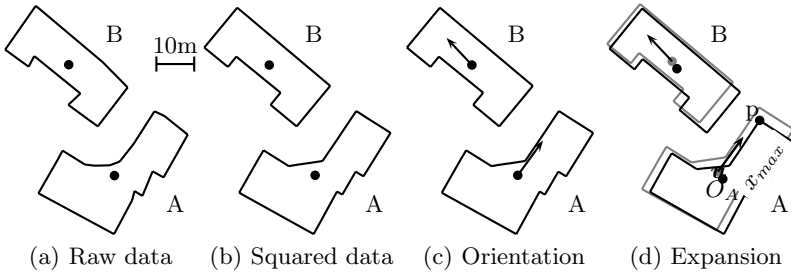


Fig. 1. Buildings polygons

The (economical) value of a dataset  $(R, DB)$  is correlated with its *mean accuracy*, its *maximum accuracy* and its *angular quality*. The *mean accuracy* is the mean value of the distance between a point of a building and its corresponding point in the dataset; the *maximum accuracy* is the maximum value of the distance between a point of a building and its corresponding point in the dataset. The *angular quality* [3] is defined as the opposite of its angular energy. The energy of an angle is a continuous piecewise quadratic function whose minima are reached for multiples of  $\pi/4$ . The angular energy of a polygon is the sum of the energies of its angles. The intuition is that angles of real-world buildings are mostly right, or at least multiples of  $\pi/4$ . So, regular buildings have lower energy levels.

### 2.2 Watermarking

A watermarking procedure is defined as a pair of algorithms  $(\mathcal{W}, \mathcal{D})$ , where  $\mathcal{W}$  is the watermarking algorithm, and  $\mathcal{D}$  is the detection algorithm. Algorithm  $\mathcal{W}$  takes as inputs a dataset  $(R, DB)$ , a secret key  $\mathcal{K}$  and some tuning parameters, and produces a watermarked dataset  $(R, DB_{\mathcal{K}})$ . The aim of the detector is, given a suspect dataset  $(R', DB')$  and the secret key  $\mathcal{K}$ , to decide whether this dataset holds a watermark or not. A watermarking procedure is said to be *blind* if the original dataset is not needed by the detector  $\mathcal{D}$ . It is said to be *robust* if it detects marks in altered watermarked datasets.

It is well known that any robust watermarking method must alter the data. Hence, there is a trade-off between the allowed alteration, i.e. the allowed impact on the quality, and the robustness of the algorithm.

### 2.3 Geographical Filters and Attacks

Geographical datasets are likely to undergo transformations by legitimate or malicious users. A broad collection of such transformations is presented below. They can be divided into correction filters (SQ, DP), legibility improvements (ETR, MBR, CE) and malicious attacks (GN, OW, CA). Nevertheless, this taxonomy is not fixed as a malicious user might apply correction filters and a legitimate one might crop a large dataset to keep only the part useful to him. A robust enough watermarking algorithm should resist all of them:

**Squaring (SQ).** For each polygon, its vertices are moved so that its angular energy is lowered. The strength of the squaring is controlled by the maximum allowed alteration  $d$  on coordinates.

**Douglas-Peucker simplification (DP).** The Douglas-Peucker simplification algorithm [4] is a polyline simplification algorithm. It works by removing the vertices of polygons that draw small artifacts on the edges of this polygon. Its strength is controlled by a threshold distance  $d$ . The higher  $d$ , the larger the removed artifacts are.

**Cropping (CA).** Polygons not contained within a given rectangularly shaped region of the dataset are discarded.

**Gaussian noise (GN).** A random noise is added to each point of the database. The distribution of the noise has mean 0 and a variable deviation  $d$ .

**Over-watermarking (OW).** Applying the watermarking algorithm with a different key on an already marked dataset.

**Enlarge to rectangle (ETR).** This filter replaces buildings by their bounding rectangle. Two modes are available. The first one replaces each building with a rectangle having the same surface. The second one takes as input a target scale and replaces the buildings that are too small (for a legally fixed threshold value) to be legible on a map at that scale.

**Change elongation (CE).** Applies a fixed ratio elongation along their orientation on all buildings of the dataset.

**Minimum bounding rectangle (MBR).** Replaces each building by its minimal bounding rectangle.

## 3 Building Watermarking

### 3.1 Outline of the Algorithm

We build a robust identifier  $id_i$  for each polygon  $P_i$  by using the highest significant bits of the coordinates of its centroid, expressed in the predefined reference system  $R_0$ . This identifier is robust since it is invariant through the modifications of vertex coordinates, involving only least significant bits. High amplitude modifications are likely to break the identifiers but also to lead to visible shapes alterations and/or polygon overlappings. Furthermore, if the coordinates of the polygon of the centroid are expressed in a reference system  $R'$ , different from  $R_0$ , it is easy to convert them back into  $R_0$ . Indeed, no geographical data comes without a reference system.

In order to hide a bit of information in polygon  $P_i$ , we expand or shrink it along its orientation. This orientation is computed relatively to the centroid (see Fig. 1(c)), and represents the majority weighted angle among edges directions. We present its computation in Section 3.3. For a rectangular shape, this orientation is parallel to the longest edge. Choosing to expand along this orientation offers several advantages. First, we observed that most edges of a polygon are parallel or perpendicular to this orientation. For example, there are 3 directions in polygon  $A$  (Fig. 1(b)): SW-NE, SE-NW and W-E. The main direction, i.e. the orientation is clearly SW-NE since the longest edges are heading this direction.

Other directions are perpendicular or make a  $\pi/4$  angle with the orientation. When a polygon is expanded along its orientation, geometrical relations between directions do not change. Second, an expansion along the orientation can still be detected if the polygon is rotated.

It remains to compute the expansion factor to apply, and to choose which polygons are going to be altered. These operations must be done so that any attacker, aware of the watermarking method, is unable to guess on which polygons they were actually applied. A classical method to achieve this [2] is the following: use the concatenation of the given identifier  $id_i$  of a polygon and the secret key  $\mathcal{K}$  of the owner to seed a pseudo-random number generator (PRNG). Use pseudo-random drawings from the generator to determine whether the current polygon is modified and, eventually, with which expansion factor. The sequence of numbers produced by the generator is predictable if and only if  $id_i \cdot \mathcal{K}$  is known. It appears purely random to anyone who does not possess this seed (an attacker may easily compute  $id_i$ , but  $\mathcal{K}$  remains unknown).

In the following, we detail the three consecutive steps of our algorithm: (1) computation of polygon identifiers and orientations, (2) computation of expansion factors and (3) watermarking by expansion.

*Example 1.* An example of our watermarking method applied on polygons  $A$  and  $B$  is shown on Fig. 1(d). Original shapes are shown in black and watermarked ones in gray. First, we compute the centroid of  $A$  and  $B$ , obtaining  $O_A = (293, 155)$  and  $O_B = (171, 447)$ . To form unique identifiers  $id_A$  and  $id_B$ , we concatenate the two highest significant digits of each coordinate, obtaining  $id_A = 2915$  and  $id_B = 1744$ . Choosing these two digits is correct under the hypothesis that any reasonable alteration is below 10 meters and that the typical distance between any two buildings is more than 10 meters (this example considers decimal base while our algorithm considers binary base). Second, based on the pseudo-random choices of a generator seeded with  $id_A$  and the secret key  $\mathcal{K}$ , we decide that  $A$  must be watermarked with a mark bit 0. We compute the main orientation  $\mathbf{u}$  of  $A$  and find the vertex  $p$  such that  $\mathbf{u} \cdot Op$  is maximal. Let  $x_{max}$  denote this value. Finally, we expand the building along its main orientation so that  $x_{max}$  becomes a predefined value  $x_{max}^0$ , encoding bit 0. Polygon  $B$  is processed identically. Remark that  $A$  has been expanded whereas polygon  $B$  has been shrunked, and that most angles are invariant under this transformation.

### 3.2 Computing Robust Identifiers

As a robust identifier, we use the highest significant bits of the centroid of the polygon following existing works on relational databases, e.g. [2]. The centroid is the center of mass of the polygon and is easily computed. Centroids of polygons  $A$  and  $B$  are represented as black dots on Fig. 1(a) and 1(b).

We need to ensure that the chosen highest significant bits are significant enough. Suppose that the  $h$ -th bit is the least highest significant bit. On the one hand,  $h$  must be high enough to that small modifications of the polygon do not change the identifier. On the other hand,  $h$  must be small enough so that two adjacent polygons do not share the same identifier. For space reasons, we omit

the discussion on a proper choice of  $h$ . It is carried on in an extended technical report [12].

The identifier of a polygon  $P$  is computed by pruning in the binary representations of its  $x$  and  $y$  coordinates the bits that represent powers of two at most  $h - 1$  and concatenating them. We denote by  $hsb(O, h)$  this operation.

$$id = hsb(O, h) = concat(hsb(x_O, h), hsb(y_O, h)).$$

### 3.3 Computing Polygon Orientation

We define the main orientation  $\mathbf{u}$  of a polygon as the maximum weighted orientation of its edges. For instance, if only  $e_1$  and  $e_2$  have orientation  $\alpha$ , then the weight of angle  $\alpha$  is the sum of the lengths of  $e_1$  and  $e_2$ . The problem is that parallel walls in the real world are not necessarily mapped to parallel edges in the dataset. So, we need to sum the lengths of edges that are almost parallel. We define  $\varepsilon$  the tolerance angle, that is  $e_1$  and  $e_2$  are considered as having the same orientation if their orientations  $\alpha_1$  and  $\alpha_2$  are such that  $|\alpha_1 - \alpha_2| < \varepsilon$ . To efficiently compute the orientation, we defined a bucket-based classifying algorithm based on the observation that there is often only a small number of different orientations per polygon. The algorithm consists of the following three steps. First, we create a set of  $k$  empty buckets, provided we choose  $k$  such that  $\pi/k < \varepsilon$ . In bucket  $i$ , we put all edges having an orientation between  $(i - 1) \cdot \frac{\pi}{k}$  and  $i \cdot \frac{\pi}{k}$ . Hence, in buckets  $i$  and  $i + 1$  we have all edges that are almost equal to  $i \cdot \pi/k$ . Then, we aggregate these small buckets into bigger ones by merging two buckets if there is no empty bucket between them. The main orientation of a building is computed as the mean value of the bucket having the highest cost (the cost of a bucket being defined as the sum of the lengths of the edges in that bucket). It can happen that three or more buckets need to be aggregated, leading to consider orientations as equal when their difference is greater than angle tolerance. This is very unlikely. Indeed, we observed that on buildings, there is only a few directions per polygon (2, 3 in most cases) which are clearly separated. A similar approach was followed in [5] with the main difference that the method proposed in [5] requires to compute the weight of all  $\pi/k$  orientations and select the one with the highest weight. Our method is more efficient but may be less accurate in a restricted number of situations.

*Example 2.* We illustrate the orientation computation algorithm on polygon A of Fig. 1(b). The number of classes is set up to 10. We got the following repartition of edges:  $\{b_1 : 3, b_4 : 8, b_9 : 6\}$  and the corresponding weights:  $\{b_1 : 9.02, b_4 : 62.4, b_9 : 45.2\}$ . The highest cost bucket is bucket 4, i.e. the orientation is between  $3\pi/10$  and  $4\pi/10 = 2\pi/5$ . The computation of the weighted mean angle of bucket 4 gives 0.96 rad.

### 3.4 Expansion as a Bit Embedding Method

In this subsection we show how to embed a single watermark bit  $b$  into a polygon  $P$ . To ensure that the watermark is robust enough, we alter the longest distance

$x_{max}$  (see Fig. 1(d)) along the orientation  $\mathbf{u}$  from the centroid  $O$  to a vertex  $p$  (for a rectangular polygon, this length is half the length of the longest edge). We name *main length* this longest distance and *quantize* it so that it becomes a multiple of a secret  $d$  coding a bit 0 or 1. But only altering the coordinates of  $p$  is not sufficient because it may lower angular quality (right angles may be flattened by this transformation). Hence, we choose to alter all lengths along the orientation  $\mathbf{u}$  so that most angles are preserved. Defining by  $\mathbf{v}$  the unary vector such that  $(0, \mathbf{u}, \mathbf{v})$  is a direct orthonormal basis, watermarking is done as follows:

- compute the x coordinate  $x_i$  of each point  $p_i$  of the polygon in  $(0, \mathbf{u}, \mathbf{v})$ ;
- compute the main length  $x_{max} = \max_i \{|x_i|\}$ ;
- expand all points coordinates along direction  $\mathbf{u}$  so that  $x_{max}$  is quantized to one of the values  $\{x_{max}^0, x_{max}^1\}$  coding a watermark bit 0 or 1. Quantization is detailed below.

Given a quantization step  $d$ , we define 0-quantizers (resp. 1-quantizers) as  $q_0^k = k.d$  (resp.  $q_1^k = k.d + d/2$ ),  $k \in \mathbb{Z}$ . Intuitively, 0-quantizers (resp. 1-quantizers) are used to code a bit 0 (resp. a bit 1). To quantize the value  $x_{max}$  using the  $i$ -quantizers ( $i \in \{0, 1\}$ ), we look for  $k_0$  such that  $|q_{k_0}^i - x_{max}|$  is minimal. More precisely, this is achieved with the following steps (quantization on 0-quantizers is presented): compute  $k_r = x_{max}/d$ ; round  $k_r$  to the closest integer  $k_0$  and define the quantized version of  $x_{max}$  as  $x'_{max} = k_0.d$ . To use 1-quantizers, one should choose  $k_r = x_{max}/d - 1/2$  and  $x'_{max} = k_0.d + d/2$ . The quantization process is illustrated on Fig. 2.

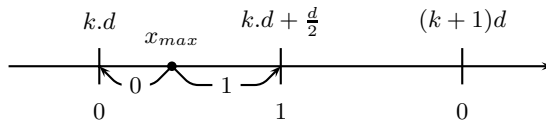


Fig. 2. Encoding 0 or 1 into the main length  $x_{max}$  using quantization

The expansion coefficient of the polygon is defined as  $\sigma = x'_{max}/x_{max}$ . We transform each point  $p = x.\mathbf{u} + y.\mathbf{v}$  in the original polygon into a point  $p' = \sigma.x.\mathbf{u} + y.\mathbf{v}$  in the watermarked polygon.

The expansion is such that the maximum distortion on a vertex of a polygon is at most  $d/2$ . Remark that this distortion can be reached only for the vertices that are the farthest to the centroid along  $\mathbf{u}$ . On the average, and for these points, the actual distortion is  $d/4$ .

### 3.5 Watermarking Algorithm

*Watermarking.* The complete algorithm is presented in Alg. 1. Let  $1/\gamma$  be the target ratio of watermarked polygons. It is a parameter of the algorithm. For each polygon of the dataset, we compute its robust identifier  $id$ . Then, we seed



a pseudo-random number generator  $G$  (PRNG) with  $\mathcal{K}.id$ . If the first integer produced by  $G$  modulo  $\gamma$  is 0, we embed a bit in the polygon. The bit is chosen according to the next binary value produced by  $G$  and embedded using the previously described expansion method.

*Variable Step Quantization.* We do not use a single quantization step  $d$  but a quantization interval  $[d_{min}, d_{max}]$ . Indeed, if  $d$  is the same for the whole dataset, main lengths of all watermarked polygons will be multiples of  $d$ . This could be easily detected and used by an attacker to alter the watermark [12].

*Discussion.* Using this method, the watermark is spread almost uniformly over the dataset. This process being controlled by a secret key, it is impossible to find the exact locations of expanded polygons, assuming the PRNG is secure. To alter the watermark, an attacker has to alter much more polygons than the watermarking process did if he wants to be sure to affect all watermarked polygons. The choice of watermarking parameters  $\gamma, d_{min}$  and  $d_{max}$  depends on the specific usage of the dataset. They cannot be fixed arbitrarily for all applications but the following rules are always valid:

- there is an unavoidable trade-off between quality alteration ( $\gamma \uparrow, d_{min} \downarrow, d_{max} \downarrow$ ) and robustness of the watermark ( $\gamma \downarrow, d_{min} \uparrow, d_{max} \uparrow$ ). Experiments presented in Section 4 give indications on how to choose optimal values;
- if the accuracy (maximum distance between a point in the dataset and in the real world) of the unwatermarked database is  $\beta_1$ , then the accuracy of the watermarked one is  $\beta_1 + d_{max}/2$ . If the watermarked dataset is sold under the agreement of an accuracy  $\beta_2$ , then  $d_{max}$  must be chosen so that  $d_{max} < 2(\beta_2 - \beta_1)$ ;
- the allowed alteration on the building, i.e.  $d_{max}$  must be higher than the typing accuracy of the dataset. Below this value, alterations can be considered as noise and rounded by a malicious user without altering the quality of the dataset at all. For instance, a 1 millimeter alteration is meaningless in a dataset of accuracy 1 meter.

### 3.6 Handling Data Constraints

The bit embedding method using expansion does not take into account topological relationships between buildings. We voluntarily chose to ignore them during bit embedding and to detect errors and cancel modifications when needed (function `testCollisions`). Such a strategy is valid as soon as few errors occur. By choosing  $d_{max} = 4$  meters, the alteration on each point of a polygon is at most 2 meters. Usually, even in urban areas, polygons are spaced by a distance superior to 2 meters. Indeed, with this value, we got only one case of overlapping, even in the worst setting, i.e. when  $\gamma = 1$ . This validates the detect-and-cancel strategy. Such a post-watermarking filtering enables to handle any kind of errors which can occur sparsely during the watermarking process.

**Algorithm 1:** Watermarking algorithm

---

```

Input: secret key  $\mathcal{K}$ , watermarking ratio  $1/\gamma$ ,  $h$ , quantization step interval
           $D = [d_{min}, d_{max}]$ 
Data:  $(R_0, DB)$ : original dataset
Output:  $(R_0, DB_{\mathcal{K}})$ : watermarked dataset
foreach building  $P$  in  $DB$  do
   $O \leftarrow \text{centroid}(P)$ ;
   $id \leftarrow \text{hsb}(O, h)$ ;                               /* robust identifier  $id$  */
   $\text{seed}(G, \mathcal{K} \cdot id)$ ;                             /* seed the PRNG  $G$  with  $\mathcal{K} \cdot id$  */
  if  $\text{nextInteger}(G) \bmod \gamma = 0$  then
    // Watermark this building
     $u \leftarrow \text{orientation}(P)$ ;                       /* orientation */
     $x_{max} \leftarrow \max\{p \in P | \mathbf{Op} \cdot \mathbf{u}\}$ ;    /* main length */
     $d \leftarrow d_{min} + \text{nextFloat}(G) \cdot (d_{max} - d_{min})$ ; /* quantization step */
     $b \leftarrow \text{nextInteger}(G) \bmod 2$ ;                /* watermark bit  $b$  */
     $x'_{max} \leftarrow \text{quantize}(x_{max}, d, b)$ ;        /* quantize  $x_{max}$  */
     $\sigma \leftarrow x'_{max}/x_{max}$ ;                    /* expansion ratio */
     $\text{expand}(P, O, u, \sigma)$ ;
    if  $\text{testCollision}()$  then
       $\text{rollback}()$ ;

```

---

### 3.7 Detection

*Outline.* Given a suspect dataset  $(R', DB')$ , we translate it into the original reference system  $R_0$ , obtaining  $(R_0, DB')$ . Then, we perform the actual detection which is very similar to the watermarking algorithm, with the essential difference that no alteration is performed. It consists of two steps: computing the ratio of matching polygons and comparing this ratio to a predefined threshold value  $\alpha$ . The values of  $d_{min}$ ,  $d_{max}$ ,  $h$ ,  $\gamma$  and  $\mathcal{K}$  used for detection must be the same as the ones used for watermarking. So they must be kept as part of the secret. For each of the polygon we seed a random generator with  $\mathcal{K}$  concatenated with its identifier. If the polygon satisfies the watermarking condition (i.e.  $\text{nextInteger}(G) \bmod \gamma = 0$ ), we compute the expected bit value  $b$  as  $\text{nextInteger}(G) \bmod 2$ . We also compute the quantization step  $d$  between  $d_{min}$  and  $d_{max}$ . Then, we decode the bit  $b'$  embedded in the main length  $x_{max}$  of the polygon and compare it with  $b$ .

*Decoding.* To decode a bit from a quantized value  $x$ , we simply check whether it is one of the 1-quantizers or one of the 0-quantizers. If  $x$  is none of the  $i$ -quantizers, we compute the closest quantized value  $x'_1$  in 1-quantizers and the closest quantized value  $x'_0$  in 0-quantizers. We compare the distance  $d_0 = |x'_0 - x|$  and  $d_1 = |x'_1 - x|$ . If  $d_0 < d_1$ , we decode a bit 0; if  $d_0 > d_1$ , we decode a bit 1. If  $d_0 = d_1$  no bit can be decoded. Note that a quantized value, with step  $d$ , can be altered up to  $d/4$  without leading to a decoding error. Quantization has been chosen because it enables to optimize the trade-off between average distortion (here,  $d/2$ ) and the minimum alteration leading to a decoding error (here,  $d/4$ ).

If the expected bit  $b$  and the decoded bit  $b'$  are the same, we say that the polygon matches. We maintain two counters,  $m$  (match) and  $t$  (total). The first one is incremented each time a polygon satisfying the watermarking condition is found. The second one is incremented each time this polygon matches. Hence, the detection ratio  $m/t$  is the ratio of matching polygons.

It is easy to see that on a third party dataset, the probability that each polygon matches is  $1/2$ . Therefore, the ratio  $m/t$  is compared to its expected value  $1/2$  to decide whether the mark of the owner is present in the document or not. Practically, a detection threshold  $\alpha$  must be set to bound the detection area. We detect a mark when  $|m/t - 1/2| \geq \alpha$ . The relevance of the detection process highly relies on the value of  $\alpha$ . From [13], setting  $\alpha = -\log(\delta/2)/2t$  achieves a false positive occurrence probability  $f_p \leq \delta$ . We use this formula in our experiments to keep false positives occurrence probability under  $\delta_0 = 10^{-4}$ .

**Proposition 1.** (direct application of [6]) *Let  $p$  the number of polygons satisfying the watermarking conditions. If each polygon has a probability  $1/2$  to match, the probability  $\mathcal{P} = \Pr(m/t - 1/2 \geq \alpha)$  is such that  $\mathcal{P} \leq e^{-2\alpha t^2}$ .*

Then, the false positive occurrence probability defined as  $f = \mathcal{P}(|m/t - 1/2| \geq \alpha)$  is such that  $f \leq 2e^{-2\alpha t^2}$ . Choosing  $\alpha = -\log(\delta/2)/2t$  as the detection threshold permits to keep  $f$  under  $\delta$ . We use this formula in our experiments to keep false positives occurrence probability under  $\delta_0 = 10^{-4}$ .

## 4 Experiments

### 4.1 Framework

*Data.* All experiments presented in this paper were realized on buildings from the French city of Pamiers. The data is part of the **BD TOPO**® [8], a topological database product from the French National Mapping Agency (IGN), the major maps provider on the French market. The product consists of several coherent layers (hydrographic network, roads, buildings...) from which we extracted only the buildings layer. This layer is composed of 4 278 polygons (35 565 vertices), representing dense build areas as well as sparse ones. It has a contractual accuracy of 1 meter. The core watermarking method is distributed under the GPL license [11].

*Filter/Attacks.* We performed an extensive series of experiments to validate the robustness of our method. All the filters/attacks presented in Section 2.3 were tested. We do not present here all the results but only a summary; the interested reader may refer to an extended technical report [12] for more details.

*Protocol.* We consider that an attack is successful if it destroys the watermark with high probability while inducing a quality loss comparable to the one introduced by the watermarking process. In a same manner, we consider that a watermarking algorithm  $A$  is better than an algorithm  $B$  against a specific attack if it is as robust as  $B$  while inducing a quality loss significantly smaller

**Algorithm 2:** Detection algorithm

---

```

Input: secret key  $\mathcal{K}$ , watermarking ratio  $1/\gamma$ ,  $h$ , quantization step interval
           $D = [d_{min}, d_{max}]$ , max. false positive occurrence probability  $f_p$ 
Data:  $(R', DB')$ , a suspect dataset
Output: MARK or NO_MARK
foreach building  $P$  in  $DB$  do
   $O \leftarrow \text{centroid}(P)$ ;
   $id \leftarrow \text{hsb}(O, h)$ ;
   $\text{seed}(G, \mathcal{K} \cdot id)$ ;
  if  $\text{nextInteger}(G) \bmod \gamma = 0$  then
     $t++$ ; /* increment total count */
     $\mathbf{u} \leftarrow \text{orientation}(P)$ ;
     $x_{max} \leftarrow \max\{p \in P | \mathbf{Op} \cdot \mathbf{u}\}$ ;
     $d \leftarrow d_{min} + \text{nextFloat}(G) \cdot (d_{max} - d_{min})$ ;
     $b \leftarrow \text{nextInteger}(G) \bmod 2$ ; /* expected bit  $b$  */;
     $x'_0 \leftarrow \text{quantize}(x_{max}, d, 0)$ ; /* closest 0-quantizer */
     $x'_1 \leftarrow \text{quantize}(x_{max}, d, 1)$ ; /* closest 1-quantizer */
    if  $|x_{max} - x'_0| > |x_{max} - x'_1|$  then
       $b' \leftarrow 1$ ; /* found bit is  $b' = 1$  */
    else
       $b' \leftarrow 0$ ; /* found bit is  $b' = 0$  */
    if  $b = b'$  then  $m++$ ; /* increment match count */
   $\alpha \leftarrow \text{threshold}(f_p, t)$ ;
  if  $|m/t - 1/2| > \alpha$  then return MARK; else return NO_MARK;

```

---

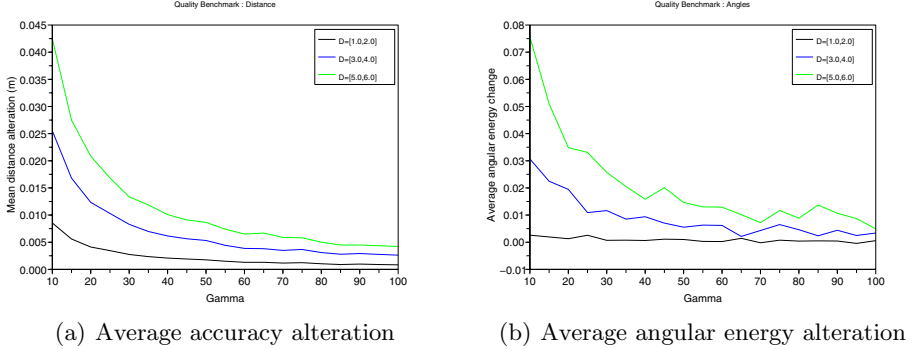
than  $B$ . To emphasize the benefits of watermarking by expansion, we put side by side a random noise based method and ours and compare them in terms of robustness/distortion trade-offs. So, we begin by quantifying the quality losses of both schemes.

## 4.2 Impact of Watermarking on Quality

To evaluate the impact of our watermarking algorithm on the Pamiers dataset, we applied it with different watermarking ratios  $1/\gamma$ , with  $\gamma \in \{10, 15, \dots, 100\}$ , and different quantization ranges  $[d_{min}, d_{max}] \in \{[1, 2], [3, 4], [5, 6]\}$ . These ranges start from data accuracy (1 meter) to the maximum reasonable alteration (6 meters).

*Average Accuracy Alteration.* The impact on the accuracy is displayed on Fig. 3(a). Alteration increases when quantization steps increase and when  $\gamma$  decreases. We observe that the average alteration of accuracy is proportional to  $(d_{min} + d_{max})/\gamma$ . For instance, when  $[d_{min}, d_{max}] = [3, 4]$ , a good approximation of the average alteration of accuracy is  $0.07 \cdot (d_{min} + d_{max})/\gamma$ . The ratio  $1/\gamma$  is not surprising since on the average,  $1/\gamma$  polygons are watermarked. Furthermore, the expected alteration for the farthest vertex from the centroid is  $(d_{min} + d_{max})/4$ .

The alteration of all the points from a watermarked polygon are proportional to the alteration of this particular vertex. These dependencies can be used by the watermarker to choose the parameters of the marking algorithm: if  $d_{max}$  is obtained as the maximum allowed alteration, and if the target alteration is fixed, one can easily compute  $\gamma$ .



**Fig. 3.** Impact of watermarking

*Angular Quality.* We verify that the variation of angular quality introduced by our method is negligible on Fig. 3(b). Even for the highest quantization steps,  $[d_{min}, d_{max}] = [5, 6]$ , the highest angular energy variation is at most  $+0.08$ . As a comparison, a weak gaussian noise (deviation  $d = 0.2m$ ) increases the angular energy by  $+6.19$ .

### 4.3 Random-Noise Based Watermarking

The random noise scheme is based on the insertion of a pseudo-random noise within the data. Even if it is not a scheme found in the literature, it supersedes existing ones and mimic the behavior of others. It is similar, in most cases, to the schemes that introduce perturbation in the dataset without taking into account the shapes of the polygons [16]. The point is that if shapes are expected to be regular, a watermarking algorithm working by altering these shapes is suboptimal in terms of robustness/distortion trade-off. Indeed, most users will correct the shapes so that the regularity aspect is brought back. This operation is very likely to remove the watermark.

The algorithm consists of looping over all the vertices of the dataset. The  $x$  and  $y$  coordinates of the vertices are watermarked independently. For each coordinate, a PRNG is seeded with its highest significant bits. Only the least significant bits are altered. Their positions and values are determined according to the drawings of the PRNG. This method can be seen as a straightforward extension of the existing AKH [2] scheme in which most significant bits of the coordinates are used as primary keys and least significant bits as alterable attributes.

The quality loss introduced by such a random noise scheme is controlled by three parameters: the watermarking ratio  $1/\gamma$ , the least highest significant bit  $lspow2$  and the number  $\xi = 2$  of alterable powers of two. For instance, if  $lspow2 = 2$  and  $\xi = 2$ , the maximum distortion on a coordinate is  $2^{lspow2-1} = 2$  and the minimum distortion is  $2^{lspow2-\xi} = 1$ . Using higher values of  $\xi$  enables for extra embedding bandwidth but lead to distortions that are removed by any rounding of the coordinates. In what follows, we compare our scheme (with  $d_{min} = 3$  and  $d_{max} = 4$ ) with two random noise schemes parameterized with two sets of parameters. For the first one, RNW1,  $lspow2 = -1, \xi = 1$ ; for the second one, RNW2,  $lspow2 = 2, \xi = 2$ . These values were chosen for the following reasons: RNW1 achieves an average accuracy alteration (compare Fig. 3(a) and Fig. 4(a) - both curves decrease as  $1/\gamma$  and have a mean accuracy alteration of 0.025 for  $\gamma = 10$ ) very close to the one observed using our method whereas RNW2 has a maximum alteration on each vertex equals to ours.

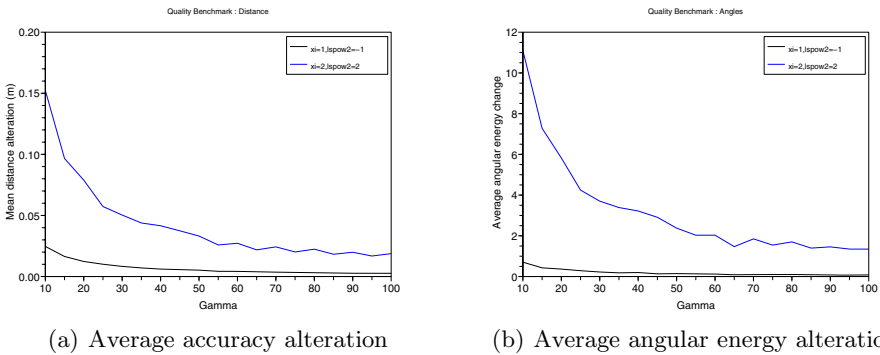


Fig. 4. Impact of random noise watermarking

### 4.4 Discussion

Table 1 sums up robustness experiments realized against geographical filters presented in Section 2.3. Robustness results for our method are presented in the WM column. Whether a method achieves robustness is indicated with the appropriate Yes/No answer. When a method is robust as far as enough polygons are watermarked, the threshold watermarking ratio under which the method has been experimentally proved robust is also indicated. Remark that robustness might also be achieved for higher values of  $\gamma$ . In the second and third columns are shown the average quality loss introduced by the attacks. These must be put into balance with the distortion introduced by watermarking algorithms (see Section 4.2 and 4.3).

As we expected, our method resist all kinds of squaring, including the most destructive ones. On the contrary, watermarks embedded using random-noise

based algorithms are washed out. Our method shows also robustness against the majority of attacks. The fact that GN( $d=1m$ ) and DP( $d=5m$ ) erase the watermark must be put into balance with the quality losses these attacks introduce. They tremendously reduce the quality of the dataset. Furthermore, it behaves well against change elongation filters since the length of polygons must be increased or decreased by more than 10% to ensure watermark removal. But this remains the weak point of our method since the embedding area is directly affected.

Compared to random-based schemes, our algorithm performs better in the majority of cases. Whereas RWN1 and RWN2 are more robust against the change elongation filter, they have a significantly higher impact on quality compared to our method. In that sense, we are confident to say that our algorithm for building watermarking achieves a very good distortion/robustness trade-off.

In the last column of the table, we combined our scheme and RNW2 into an *expand-and-translate* scheme. In a first time, we apply our scheme and then apply RNW2 on the centroid of the polygon (left invariant by the first step). We obtain a modification of the coordinates of the centroid which is used to translate the polygon. Obviously, this combined scheme introduces a larger quality loss but it achieves robustness for all the filters/attacks used in the paper. We think that this combined scheme is a relevant practical scheme which adds an extra robustness wall to RNW2 used alone by altering the shapes of the polygons.

**Table 1.** Robustness of watermarking

Filter	Precision alt.	Angular energy	WM	RNW1	RNW2	Combined
SQ ( $d=1m$ )	0.19	-14.1	Yes	No	Yes	Yes
SQ ( $d=1.5m$ )	0.23	-16.2	Yes	No	$\gamma < 90$	Yes
SQ ( $d=2m$ )	0.27	-17.2	Yes	No	$\gamma < 80$	Yes
DP ( $d=1m$ )	N/A	- 1.3	Yes	Yes	Yes	Yes
DP ( $d=2m$ )	N/A	- 0.3	Yes	Yes	Yes	Yes
DP ( $d=5m$ )	N/A	68.1	$\gamma < 70$	Yes	Yes	$\gamma < 100$
CA	0	0	Yes	Yes	Yes	Yes
GN ( $d=0.2m$ )	0.18	6.19	Yes	$\gamma < 30$	Yes	Yes
GN ( $d=0.6m$ )	0.53	40.00	Yes	No	$\gamma < 75$	Yes
GN ( $d=1m$ )	0.89	78.52	$\gamma < 65$	No	$\gamma < 30$	$\gamma < 75$
OW	-	-	Yes	Yes	Yes	Yes
ETR	N/A	-6.53	$\gamma < 40$	No	$\gamma < 25$	$\gamma < 50$
ETR(scale=25000)	N/A	-6.06	No	No	No	$\gamma < 40$
ETR(scale=250000)	N/A	-0.03	Yes	Yes	Yes	Yes
CE(scale=0.90)	1.06	0.16	No	No	$\gamma < 45$	Yes
CE(scale=0.95)	0.53	0.02	$\gamma < 95$	No	Yes	Yes
CE(scale=1.0)	0	0	Yes	Yes	Yes	Yes
CE(scale=1.05)	0.53	0.09	$\gamma < 95$	Yes	Yes	Yes
CE(scale=1.10)	1.06	0.27	No	No	$\gamma < 50$	Yes
MBR	N/A	-6.6	$\gamma < 75$	$\gamma < 35$	$\gamma < 50$	$\gamma < 75$

## 5 Related Work

Despite that state-of-the art is very rich on watermarking still images which can be directly applied to image maps, fewer works were carried on on watermarking vector maps. A complete study of vector maps watermarking [16] has been recently published which divides this field into three categories: spatial domain watermarking, transform domain (DCT,DWT,...) watermarking and 3D meshes adapted algorithms. Spatial domain watermarking consists of modifying directly the coordinates of the vertices of the dataset. Patch-based techniques [9,17,20,24] are based on a decomposition of the dataset into patches in which bits are embedded by vertices translation [17] and/or data distribution within a patch [9,20,24]. Such schemes are sensitive to patch decomposition that can vary when the dataset is cropped. Schemes of [9,20] create detectable nodes aggregations; [17] is not blind and [24] does not respect the shape of smooth objects (including squared buildings).

Vertices addition based [7,15,19] techniques are the best from a quality viewpoint. They consist of interpolating the existing edges of the dataset with fake points. In the context of buildings where the shapes are regular, such interpolations are easily detected and removed by simplification algorithms. In our context, these schemes are not robust enough. The least-significant bit watermarking method presented in [23] preserves accuracy but is very sensitive to vertices addition, which destroys synchronization.

In [21], a high watermarking capacity algorithm for vector maps is introduced. It is based on a previous work on 3D meshes watermarking by the same authors. It is robust against common geographical filters like Douglas-Peucker simplification algorithm. It is based on a decomposition of the database into patches and by moving points of a common patch into a subpatch to embed the watermark. Nevertheless, efficient attacks erasing the watermark without destroying the quality of the dataset can be easily planned, as the method requires known synchronization points.

Our method shares a common skeleton with the popular AHK algorithm [2]. But it differs on several aspects: it does not require primary keys, and do not use high significant bits to replace them in a straightforward manner (instead the centroid is used); the bit embedding operation is not a least significant bit embedding, which is very fragile against rounding, but a quantization method.

## 6 Conclusion

In this paper we presented a blind watermarking algorithm for polygon datasets. It is well suited to building layers of geographical datasets since watermarks are invariant through aggressive geographical filters applied by data users, including squaring and boxing. We experimentally showed that it is difficult for an attacker to erase the watermark without paying an extra quality fee, compared to watermarking. The algorithm has been implemented into an open database watermarking framework [14] and is available online [11]. We are currently working



on designing algorithms for other layers of geographical datasets. A real challenge we are faced with is to deal with the interactions between the different layers. Indeed, watermarking algorithms must be adapted to the data; there is no unique solution. Even if we know how to perform watermarking and detection on a single layer, it is challenging to orchestrate several algorithms on several layers so that resulting watermarked datasets remain consistent.

*Acknowledgements.* We would like to thank Eric Grosso from IGN for his technical support and helpful discussions.

## References

1. GéoPortail (visited 03/20/2007). <http://www.geoportail.fr>
2. Agrawal, R., Haas, P.J., Kiernan, J.: Watermarking relational data: framework, algorithms and analysis. VLDB J. 12(2), 157–169 (2003)
3. Airault, S.: De la base de données à la carte: une approche globale pour l'équarrissage de bâtiments (in french). Revue Internationale de Géomatique 6(2-3), 203–217 (1996)
4. Douglas, D., Peucker, T.: Algorithms for the reduction of the number of points required for represent a digitized line or its caricature. Canadian Cartographer 10(2), 112–122 (1973)
5. Duchêne, C., Bard, S., Barillot, X., Ruas, A., Trevisan, J., Holzapfel, F.: Quantitative and qualitative description of building orientation. In: Fifth workshop on progress in automated map generalisation, ICA, commission on map generalisation (April 2003)
6. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association 58(301), 13–30 (1963)
7. Huber, W.A.: Gis and steganography - part 3: Vector steganography (April 2002) [http://www.directionsmag.com/article.php?article\\_id=195](http://www.directionsmag.com/article.php?article_id=195)
8. Institut Géographique National. BD TOPO - descriptif technique (in french) (December 2002), [http://www.ign.fr/telechargement/MPro/produit/BD\\_TOPO/JT\\_Aggl0/DT\\_BDTopoPays\\_1\\_2.pdf](http://www.ign.fr/telechargement/MPro/produit/BD_TOPO/JT_Aggl0/DT_BDTopoPays_1_2.pdf)
9. Kang, H.I., Kim, K.I., Cho, J.U.: A vector watermarking using the generalized square mask. In: Information Technology: Coding and Computing, pp. 234–236 (April 2001)
10. Katzenbeisser, S., Petitcolas, F.A.P.: Information hiding, techniques for steganography and digital watermarking. Artech house (2000)
11. Lafaye, J.: Watermarking plugin for openjump (2007), [http://cedric.cnam.fr/~lafaye\\_j/index.php?n=Main.WaterGoatOpenJumpPlugin](http://cedric.cnam.fr/~lafaye_j/index.php?n=Main.WaterGoatOpenJumpPlugin)
12. Lafaye, J., Béguet, J., Gross-Amblard, D., Ruas, A.: Blind watermarking of geographical databases by polygon expansion. In: Technical Report hal-00137956, CNRS-CCSD HAL (March 2007), Available online <http://hal.archives-ouvertes.fr/hal-00137956>
13. Lafaye, J., Gross-Amblard, D.: Xml streams watermarking. In: 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC 2006) (2006)
14. Lafaye, J., Gross-Amblard, D., Guerrouani, M., Constantin, C.: Watermill: an optimized fingerprinting system for databases under constraints. submitted to TKDE (2007)

15. Lopez Vazquez, C.M.: Method of inserting hidden data into digital archives comprising polygons and detection methods, US Patent no. 20030208679 (November 2003)
16. Niu, X., Shao, C., Wang, X.: A survey of digital vector map watermarking. *International Journal of Innovative Computing, Information and Control*, 2(6) (December 2006)
17. Ohbuchi, R., Ueda, R., Endoh, S.: Robust watermarking of vector digital maps. In: *Multimedia and Expo, 2002. ICME '02*, vol. 1, pp. 577–580 (2002)
18. Ohbuchi, R., Ueda, H., Endoh, S.: Watermarking 2D vector maps in the mesh-spectral domain. In: *Shape Modeling International*, pp. 216–228. IEEE Computer Society Press, Los Alamitos (2003)
19. Park, K.T., Kim, K.I., Kang, H.I., Han, S.-S.: Digital geographical map watermarking using polyline interpolation. In: Chen, Y.-C., Chang, L.-W., Hsu, C.-T. (eds.) *PCM 2002. LNCS*, vol. 2532, pp. 58–65. Springer, Heidelberg (2002)
20. Sakamoto, M., Matsuura, Y., Takashima, Y.: A scheme of digital watermarking for geographical map data. In: *Symposium on cryptography and information security*, Okinawa, Japan (January 2000)
21. Schulz, G., Voigt, M.: A high capacity watermarking system for digital maps. In: *MM&Sec '04: Proceedings of the 2004 workshop on Multimedia and security*, Magdeburg, Germany, pp. 180–186. ACM Press, New York, NY, USA (2004)
22. The International Hydrographic Organization (IHO). *Specifications for Chart Content and Display Aspects of ECDIS*. IHO, 5th edn. (December 2001)
23. Voigt, M., Busch, C.: Watermarking 2d-vector data for geographical information systems. In: *SPIE, Security and Watermarking of Multimedia Content*, vol. 4675, pp. 621–628 (2002)
24. Voigt, M., Busch, C.: Feature-based watermarking of 2d vector data. In: *SPIE, Security and Watermarking of Multimedia Content*, vol. 5020, pp. 359–366 (June 2003)

# Transformation of Continuous Aggregation Join Queries over Data Streams

Tri Minh Tran and Byung Suk Lee

Department of Computer Science, University of Vermont, Burlington VT 05405, USA  
{ttran, bslee}@cems.uvm.edu

**Abstract.** We address continuously processing an *aggregation join* query over data streams. Queries of this type involve both join and aggregation operations, with windows specified on join input streams. To our knowledge, the existing researches address join query optimization and aggregation query optimization as *separate* problems. Our observation, however, is that by putting them within the *same* scope of query optimization we can generate more efficient query execution plans. This is through more versatile *query transformations*, the key idea of which is to perform aggregation before join so join execution time may be reduced. This idea itself is not new (already proposed in the database area), but developing the query transformation rules faces a completely new set of challenges. In this paper, we first propose a query processing model of an aggregation join query with two key stream operators: (1) aggregation set update, which produces an aggregation set of tuples (one tuple per group) and updates it incrementally as new tuples arrive, and (2) aggregation set join, i.e., join between a stream and an aggregation set of tuples. Then, we introduce the concrete query transformation rules specialized to work with streams. The rules are far more compact and yet more general than the rules proposed in the database area. Then, we present a query processing algorithm generic to all alternative query execution plans that can be generated through the transformations, and study the performances of alternative query execution plans through extensive experiments.

## 1 Introduction

In this paper, we consider the problem of processing continuous *aggregation join* queries over data streams. These queries involve both join and aggregation operations. (The aggregation may be a grouped aggregation.) Many aggregation join queries are *window-based* because joins are blocking operators (i.e., needing a finite set of tuples). A window, which restricts the number of tuples processed, is a common technique proposed in many existing researches [1,2,3,4,5,6,7,8,9,10].

Window-based aggregation join queries (called simply “aggregation join queries” in this paper) are needed in various data stream applications. For example, in a telephone call tracking application [11], a telephony company may want to keep track of the monthly total calling time on international calls made from each telephone number (i.e., subscriber) within a specific area code [11]. As another example, in a network traffic management application [10], a network administrator may want to monitor packet data flow through links between different networks [10]. For another example, in an online

auction system which has continuous streams of auction items registered, members (i.e., account holders) signing in and bids made may be monitored to build some statistics of auction activities. Below, let us take a look at an example query for this application.

*Example 1.* In an online auction application, we may have a continuous query running on two data streams Bid(ts, auctionID, bidderID, bidPrice) and Auction(ts, auctionID, sellerID, startPrice)<sup>1</sup> and one relation Person(personID, name, state). Users may want to know, for each auction created up to now by a seller from Vermont, the total number of bids made in the last one hour. In this case, the query involves a three-way join (involving two stream windows and one relation) and a grouped aggregation, grouped by auctionID. The query can be expressed as an aggregation join query as follows.

```
SELECT A.auctionID, COUNT(B.*)
FROM Auction AS A [WINDOW UNTIL NOW],
      Bid AS B [WINDOW 1 HOUR],
      Person AS P
WHERE A.auctionID = B.auctionID
      AND A.sellerID = P.personID
      AND P.state= "VT"
GROUP BY P.auctionID;
```

□

Naturally, efficient processing of these aggregation join queries is very important. One premise in this paper is that, the queries can be processed more efficiently if the optimizations of join and aggregation are *handled as one problem*. Most of the existing researches address them as separate problems: for example, joins in [3, 2, 1, 13, 14] and aggregations in [15, 11, 16, 17, 18]. Two other existing researches [19, 15] address the problem of efficiently processing aggregation join queries as one, but not as an optimization problem per se. Furthermore, their methods can only provide approximate answers using sketching techniques [15] and discrete cosine transform [19], respectively; thus, they cannot be applied to our problem since they are not window-based and cannot handle grouped aggregations.

The premise mentioned above opens a door to generating a heuristically more efficient query execution plan (QEP) through *query transformations*, and this is the focus of this paper. In the initial QEP of an aggregation join query, joins are performed first and then aggregation follows. The key idea of query transformation in this paper is to perform an aggregation before join – in other words, push aggregation down to a join input in a query execution tree. This transformation generally reduces the join input cardinality and results in a more efficient QEP, although this may not be always guaranteed. In this paper we call the initial QEP a *late aggregation plan (LAP)* and the transformed QEP an *early aggregation plan (EAP)*, and call the pushed-down aggregation operator an *early aggregation operator*.

Similar query transformation mechanism has been proposed in [20] and [21]. Their mechanism, however, is for *database* aggregation join queries. Due to the streaming nature of data, stream queries are fundamentally different from database queries. First, tuples arrive continuously and hence the query output must be updated *continuously* as

<sup>1</sup> Based on the schema used by Babu et al. [12].

well. Second, in many cases arriving tuples must be processed on-line and this requires that the query must be processed *incrementally* as soon as tuples arrive. These differences make the transformation rules for database aggregation join queries inapplicable to stream aggregation join queries.

In order to handle this problem, we introduce two key stream operators for query processing: an *aggregation set update* (AS update) and an *aggregation set join* (AS join). An AS update operator is used to update aggregate values incrementally as new tuples arrive on the input. This operator works the same way as the group-by operator mentioned in [9]. An AS join operator is used to perform a join between a new tuple arriving at one stream and the output of an early aggregation operator (called an *aggregation set*) at another stream. Note its distinction from a window join which uses a window of tuples instead of an aggregation set. To our knowledge, the AS join operator is a new operator introduced the first time through this paper. An AS update is preceded by a *window* join in an LAP, whereas preceded by an AS join in an EAP.

There is a side effect of using the AS join operator. As mentioned earlier, we consider a window-based join in this paper. A window join is processed as multiple one-way window joins – that is, each new tuple arriving in one stream is matched with tuples in the windows of the other streams. By performing early aggregations in an EAP, one or more of these one-way window joins in an LAP is replaced by one-way AS joins in an EAP. This results in *different* join output schemas depending on which window joins are replaced, because the join output schema of a one-way AS join is different from that of a window join or another one-way AS join. This side effect can be easily treated by retaining a late aggregation operator on the query output even after placing an early aggregation before joins. As a coincidental side benefit, this approach does not require any constraint between streams, unlike the database case in which either a foreign key join [20] or a functional dependency [21] is required.

In this paper we first formalize the notions of the aggregation set (AS) and the two associated operators, AS update and AS join. Then, we propose compact query transformation rules based on the approaches mentioned above, that is, supporting AS update and AS join operators and retaining a late aggregation operator. Additionally, we present a generic algorithm for executing all alternative QEPs (i.e., LAP and EAPs). Note that the algorithm works just as well for a stream-*relation* join as a stream-stream join, since a relation can be viewed as a window with no update of tuples. (We have also algebraically proven the equivalence of the generated QEPs, but we omit the details in this paper due to space limit. Interested readers are referred to [22].) Then, through experiments we study the efficiencies of alternative QEPs for varying key parameters (e.g., window size, stream rate, number of groups, join selectivity factor).

To our knowledge, this is the first work addressing query transformation on aggregation join query over data streams. Main contributions include a formal query processing model that are suitable for an aggregation join query and transformation rules that are compact and yet general enough not to assume any constraint among input streams.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 discusses some preliminary concepts. Section 4 describes the query processing model and the key operators. Section 5 proposes the query transformation rules and

presents the generic query processing algorithm. Section 6 presents the experiments and the results. Section 7 concludes the paper.

## 2 Related Work

We discuss related work in two areas: (1) processing join queries and aggregation queries in data streams and (2) handling early aggregations in the *database*.

### Processing Join and Aggregation in Data Streams

As far as we know, all of the existing data stream query processing systems – such as Aurora [23], STREAM [24], TelegraphCQ [25], NiagaraCQ [26], Stream Mill [27], Nile [28], Tribeca [29] and GigaScope [30] – process aggregation join queries by handling join and aggregation *separately*. Specifically, in all these systems an aggregation join query is always processed by first processing the join and then passing the output of the join onto the aggregation operator. In Aurora [23] and STREAM [24], query optimizers use query transformations by reordering filter operators (i.e., selection) and join operators in a QEP to generate equivalent QEPs. Their reordering, however, is not applicable between a join operator and an aggregation operator. In the other systems query optimizers do not even use query transformation at all. Thus, to our knowledge, our work is the first to allow reordering the join and aggregation operators.

Aside from these comprehensive data stream management systems, join processing and aggregation processing have been researched quite extensively. A large number of them focus on *window join* processing [1, 2, 13, 14, 3, 31, 8, 32]. In [1], Kang et al. propose sliding window two-way join algorithms and develop a unit-time cost model that estimates the execution time of the join algorithms. Golab et al. [2] extend Kang et al.'s work to multi-way window join algorithms and propose join ordering heuristics to reduce the cost. Viglas et al. [13] propose a pipelined multi-way window join called MJoin. An MJoin assigns a join order for each input stream and generates join output without maintaining intermediate results. In contrast to MJoin, an XJoin proposed in [14] is a multi-way join executed in a tree of two-way joins and maintains a fully-materialized join results for each intermediate two-way join. Some other researches [3, 31, 33] address *approximate* window join processing in the case of limited system resources. None of these window join researches considers an aggregation following a join operator.

For *window aggregation* processing, Li et al. [4] propose a generic window concept and present an efficient window aggregation technique which computes the aggregate values in one pass. The key idea is to assign to each tuple a range of the identifiers of windows to which it belongs. Zhang et al. [34] address the problem of processing multiple aggregation queries that differ only in grouping attributes. Some other researches address computing approximate answers to an aggregation query using sampling [35, 36], wavelets [18, 16], histograms [11, 17], and sketching [15, 37].

As mentioned in Introduction, two researches [19, 15] address the problem of processing the same type of query as ours. Their approaches, however, are to use approximation techniques using sketching [15] and discrete cosine transform [19]. Moreover, they do not consider window-based and grouped aggregations in their problems.

### Handling Early Aggregation in the Database

The early aggregation idea stems from the idea proposed for database queries [20, 38, 21]. The key idea of performing an early aggregation is to reduce the number of tuples participating in subsequent joins. In [20] the authors present query transformation rules for three cases depending on which relation the grouping, aggregation, and join attributes belong to. The authors also introduce a new operator called *aggregate join* that performs a join between one relation and the output of an early aggregation on the other relation. Yan et al. [38, 21] consider more general transformation cases in which the grouping attributes and the aggregation attributes may belong to more than one relation. In addition, instead of introducing a new operator as in [20], they use a “query rewriting” technique which involves reordering the join and aggregation operators and inserting an additional projection operator to compute the aggregate value of the reordered operators. As already mentioned, our work is fundamentally different from their works, as we deal with unbounded continuous data streams, not bounded finite set of tuples (i.e., relations).

## 3 Preliminaries

In this section, we present some key concepts needed to understand the rest of the paper.

**Data Streams.** We consider a data stream  $S$ , of an ordered sequence of tuples. Each tuple in the stream has the schema  $S(TS, X_1, X_2, \dots, X_d)$ , where  $TS$  is a timestamp attribute and  $X_1, X_2, \dots, X_d$  are non-timestamp attributes. We denote a tuple of the above schema as  $s(ts, x_1, x_2, \dots, x_d)$ , where  $ts$  is the value of  $TS$  and  $x_i$  is the value of  $X_i$  for each  $i = 1, 2, \dots, d$ . (We use an upper-case letter to denote an attribute and a lower-case letter to denote the value of an attribute.) We assume that the tuples arrive in the order of timestamp; handling out-of-order tuples is beyond the scope of this paper.

**Windows.** Three types of window are considered in our processing model. They are classified as in [4] depending on how the tuples in the window are updated: *sliding* window, *tumbling* window, and *landmark* window. A sliding window partitions an incoming stream into overlapping blocks, and a tumbling window does that into disjoint consecutive blocks. A landmark window accumulates all tuples that have arrived since the start of the query.

**Definition 1 (Window).** A window  $W_S$  of size  $T$  on stream  $S$  at time  $t$  is defined as a set of tuples whose timestamps are in the range of  $[t - T, t]$ . That is,  $W_S(t) = \{s_i \mid t - T \leq s_i.ts < t\}$ .  $\square$

**Definition 2 (Window increments and decrements).** Given a window  $W_S(t_1)$  at time  $t_1$ , a *window increment*, denoted as  $W_S^+(t_1, t_2)$ , is the set of tuples added to the window during a time interval  $[t_1, t_2]$ , and a *window decrement*,  $W_S^-(t_1, t_2)$ , is the set of tuples removed from the window during the same time interval.  $\square$

Given a window  $W_S(t_1)$  at time  $t_1$ , and a window increment  $W_S^+(t_1, t_2)$  and decrement  $W_S^-(t_1, t_2)$  between  $t_1$  and  $t_2$ , the window  $W_S(t_2)$  at time  $t_2$  is computed as:

$$W_S(t_2) = W_S(t_1) \cup W_S^+(t_1, t_2) - W_S^-(t_1, t_2)$$

Given the above definitions of window increments and decrements, the tumbling window and the landmark window can be considered as special cases of the sliding window. Figure 1 illustrates the three window types with their corresponding increments and decrements.

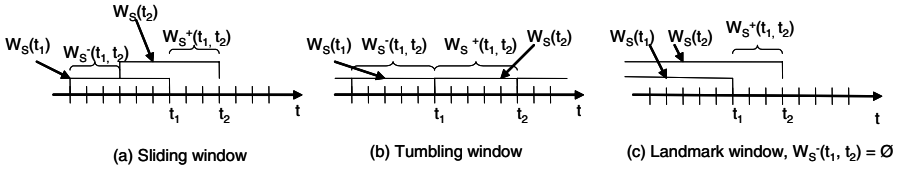


Fig. 1. Windows of different types ( $t_1 < t_2$ )

**Window Joins.** A two-way window join [11] between two streams  $S_1$  and  $S_2$  with windows  $W_{S_1}$  and  $W_{S_2}$ , respectively, is computed as follows. For each new tuple  $s_1$  in a window increment of  $S_1$ ,  $s_1$  is inserted into  $W_{S_1}$  and any expired tuples are removed from  $W_{S_1}$ . Then,  $W_{S_2}$  is probed for matching tuples of  $s_1$  and matching tuples are appended to the join output stream. The computation is symmetric for each new tuple  $s_2$  in a window increment of  $S_2$ . Generalized from this, in a multi-way join among  $m$  ( $m > 2$ ) streams, for each new tuple  $s_k$  in a window increment of  $S_k$ , matching tuples are found from the other  $m - 1$  windows and then appended to the output stream. We assume that the join computation is fast enough to finish before the other  $m - 1$  windows are updated.

## 4 Query Processing Model

In this section we present a model for continuous and incremental processing of aggregation join queries. Key components of the model are the *aggregation set*, *aggregation set update (AS update)* operator and the *aggregation set join (AS join)* operator. This model provides a basis for the query transformation rules and the query processing algorithm presented in Section 5.

The concepts of aggregation set and AS update operator are the same as the concepts of window aggregate and group-by operator mentioned in [9]. These concepts are refined and presented formally in this paper using the notions of window increment and window decrement. The AS join is a combination of the window join defined in Section 3 and the “aggregate join” proposed for database aggregation join queries in [20].

### Aggregation Sets

Aggregation of the tuples in a *window* produces a set of tuples, one tuple for each group. We call this set of tuples an *aggregation set (AS)*.

**Definition 3 (Aggregation set).** Consider a set of tuples in a window at time  $t$ , denoted as  $W_S(t)$ . Additionally, consider an aggregation operator, denoted as  ${}_G\mathcal{A}_{F(A)}(W_S(t))$  where  $G \equiv (G_1, \dots, G_p)$  is a list of grouping attributes,  $A$  is an aggregation attribute, and  $F$  is an aggregation function on  $A$ . Then, an *aggregation set* is defined as a set of tuples  $\{(g_1, \dots, g_p, v)\}$  where  $g_i$  is a value of  $G_i$  ( $i = 1, 2, \dots, p$ ) and  $v$  is an aggregate value computed as  $F(A)$  for the group  $(g_1, \dots, g_p)$  over  $W_S(t)$ . We denote the



schema of an aggregation set as  $AS(G, F(A))$ ; here,  $F(A)$  denotes an attribute whose value is  $v$ .  $\square$

### Aggregation Set Update

An aggregation set update operator is used to update the AS as the window content changes. This is done *incrementally* without re-evaluating the whole window content.

**Definition 4 (Aggregation set update).** Consider an aggregation set  $AS \equiv {}_G\mathcal{A}_{F(A)}(W_S^+(t_1))$  at time  $t_1$ , a window increment  $W_S^+(t_1, t_2)$  and a window decrement  $W_S^-(t_1, t_2)$  at time  $t_2 (> t_1)$ . Then, an *AS update* operation, denoted by  ${}_G\mathcal{U}_{F(A)}(AS, W_S^+(t_1, t_2), W_S^-(t_1, t_2))$ , returns an updated aggregation set  $AS'$  resulting from the following updates on  $AS$ :

- For each tuple  $s$  in  $W_S^+(t_1, t_2)$ , if there exists a tuple  $l$  in  $AS$  such that  $l.G = s.G$  (i.e.,  $s$  belongs to a group in  $AS$ ) then update the aggregate value  $l.F(A)$  as follows: if  $F$  is COUNT then increase  $l.F(A)$  by one; if  $F$  is SUM then increase  $l.F(A)$  by  $s.A$ ; if  $F$  is MIN and  $s.A < l.F(A)$  or  $F$  is MAX and  $s.A > l.F(A)$  then set  $l.F(A)$  to  $s.A$ , otherwise no change; (if  $F$  is AVG then compute  $l.F(A)$  by maintaining both COUNT and SUM). If there does not exist such a tuple  $l$  in  $AS$ , then insert a new tuple  $l'$  with  $l'.G$  set to  $s.G$  and  $l'.F(A)$  set to 1 if  $F$  is COUNT or to  $s.A$  if  $F$  is in {SUM, AVG, MIN, MAX}.
- For each tuple  $r$  in  $W_S^-(t_1, t_2)$ , find a tuple  $l$  in  $AS$  such that  $l.G = r.G$  (i.e.,  $r$  belongs to a group in  $AS$ ), and then update the aggregate value  $l.F(A)$  as follows: if  $F$  is COUNT then decrease  $l.F(A)$  by one; if  $F$  is SUM then decrease  $l.F(A)$  by  $s.A$ ; if  $F$  is MIN or MAX and  $r.A = l.F(A)$  then recompute  $l.F(A)$  from the set  $W_S(t_1) - \{r\}$ , otherwise no change.  $\square$

As we see from the above definition, updating an aggregate value  $l.F(A)$  for each tuple  $r \in W_S^-(t_1, t_2)$  requires re-evaluating the whole window only if  $F$  is MIN or MAX and  $r.A = l.F(A)$ . Note that even this situation happens only with a sliding window and not with a tumbling or a landmark window. In the case of a tumbling window, a window decrement is discarded and a new aggregation set is generated using the new window increment only. In the case of a landmark window, there is no window decrement.

### Aggregation Set Joins

We first present the *coalescing property* [20] of an aggregation function; this property will be used to define the aggregation set join in Definition 6.

**Definition 5 (Coalescing property).** Consider an aggregation function  $F$  on an attribute  $A$ . The aggregate of  $c$  tuples that have the same value,  $a$ , of  $A$  is computed using the following function  $f(c, a)$  depending on the type of  $F$ .

$$f(c, a) = \begin{cases} a * c & \text{if } F \equiv \text{SUM} \\ c & \text{if } F \equiv \text{COUNT} \\ a & \text{if } F \in \{\text{AVG, MAX, MIN}\} \end{cases} \quad \square$$

An AS join handles a join between a stream  $S$  and an aggregation set  $AS$  and computes the aggregate value of a join output tuple using the coalescing property.

**Definition 6 (One-way aggregation set join).** Consider two streams  $S_1$  and  $S_2$  with their window  $W_{S_1}(t_1)$  and  $W_{S_2}(t_1)$ , respectively, at time  $t_1$ . Additionally, consider the window increment  $W_{S_1}^+(t_1, t_2)$  and decrement  $W_{S_1}^-(t_1, t_2)$  of  $S_1$  at time  $t_2 (> t_1)$ . Now, given an aggregation  $F(A)$  specified in the query, let the aggregation set  $AS_2(t_1)$  on stream  $S_2$  be computed as follows depending on whether  $A$  is in the schema of  $S_2$  or not.

$$AS_2(t_1) = \begin{cases} GA_{F(A)}(W_{S_2}(t_1)) & \text{if } A \text{ belongs to } S_2. \text{ (See Definition 3)} \\ GA_{COUNT(A)}(W_{S_2}(t_1)) & \text{otherwise} \end{cases}$$

Then, a one-way AS join from  $S_1$  to  $AS_2$  via join attributes  $S_1.J_1$  and  $AS_2.J_2$ , denoted as  $S_1 \stackrel{F(A)}{\bowtie}_{J_1=J_2} AS_2$ , is computed as follows.

For each tuple  $s_1$  in  $W_{S_1}^+(t_1, t_2)$  and for each tuple  $r_1$  in  $W_{S_1}^-(t_1, t_2)$ ,

1. Find matching tuples from  $AS_2(t_1)$ . (Denote each tuple as  $l$ )
2. Return a sequence of tuples where for each tuple ( $u$ ) the value of  $F(A)$  is set as follows.

$$u.F(A) = \begin{cases} l.F(A) & \text{if } A \text{ belongs to } S_2 \\ f(c, a) & \text{otherwise} \end{cases}$$

where  $a$  is the value of  $s_1.A$  (or  $r_1.A$ ),  $c$  is the number of tuples aggregated to  $l$  in  $AS_2$ , and  $f$  is the function in the definition of the coalescing property (Definition 5).  $\square$

An extension to a *multi-way* AS join is straightforward. That is, one-way AS join is repeated from each stream ( $S_k, (k \in \{1, 2, \dots, m\})$ ) to the aggregation sets  $AS_i$  on the other streams  $S_i, i \neq k$ .

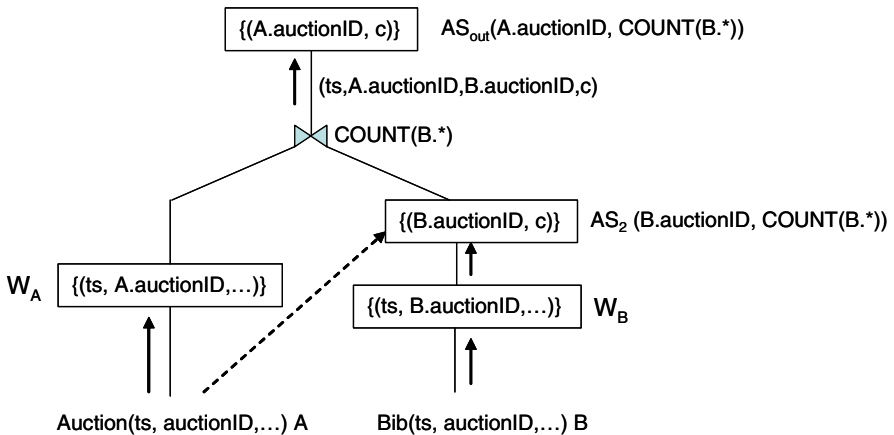


Fig. 2. An example one-way AS join

*Example 2.* Given the query in Example 1, a one-way AS join between the stream Auction  $A$  and the aggregation set  $AS_2$  on Bid  $B$  shown in Figure 2:

$$A \begin{matrix} \text{COUNT}(\ast) \\ \bowtie \\ A.auctionID=B.auctionID \end{matrix} B$$

where  $AS_2 \equiv B.auctionID.ACOUNT_{(B,\ast)}(W_B(t))$ .  $AS_2$  is then a set of tuples,  $\{(B.auctionID, c)\}$ . For each tuple  $(ts, A.auctionID, \dots)$  in  $W_A^+$ , the one-way AS join from  $A$  to  $AS_2$  produces a sequence of output tuples  $u(ts, A.auctionID, B.auctionID, c)$  where  $A.auctionID = B.auctionID$  and the aggregate value equals  $c (= f(c, a)$  in Definition 5). Similar steps are taken for each tuple in  $W_A^-$ .  $\square$

## 5 Query Transformations

In this section, we first propose transformation rules for generating EAPs. We then present a generic algorithm for executing a query execution plan (QEP), i.e., a late aggregation plan (LAP) or an early aggregation plan (EAP).

### 5.1 Transformation Rules

In this section, we propose query transformation rules developed for aggregation join queries on data streams. As mentioned in the Introduction, there are two technical problems in order to make query transformation rules work on data streams. First, the aggregation sets in a QEP should be updated incrementally and continuously, both before and after the transformation. Second, the transformation should cope with the different schemas of one-way join outputs in an EAP, as the join output schema of one-way AS join in an EAP differs according to the schema of the aggregation set generated by an early aggregation operator. Since a join output is a union of multiple one-way (AS or window) join outputs but join output schema of a one-way AS join is different from that of a one-way window- or another AS join, the different schemas of one-way join outputs hinder the union.

To handle the first problem, we use the AS update and AS join operators introduced in Section 4. Precisely, *only* the AS update operator is needed in an LAP and *both* operators are needed in an EAP. To handle the second problem, in the transformed plan we *always* keep a late aggregation (LA) operator in its original position. This LA operator guarantees that the schema of the aggregation join query output is the same even though the schemas of one-way join outputs are different. This guarantee is due to the fact that two different tuples with the same grouping attribute value are put into the same group.

In an EAP, early aggregation (EA) operators may be placed on any of the input streams. Once placed on a certain input stream, the operator generates an AS and, thus, allows for using an AS join to the AS instead of the window join to the input stream window. Determining the input streams to place EA operators on is based on the resulting EAPs' execution times as estimated using cost models<sup>2</sup>. For those EA operators inserted, their grouping attributes and aggregation functions are determined using the following *EA operator construction* rules.

<sup>2</sup> In this paper, we focus on query transformations only, cost models are presented in [22].

**Rule 1 (Grouping attribute in an EA operator)**

If the EA operator is placed on a stream that has some or all of the grouping attributes in the query, then use these and the join attributes as the grouping attributes of the EA operator. Otherwise, use only the join attributes as the grouping attributes of the EA operator.  $\square$

**Rule 2 (Aggregation function in an EA operator)**

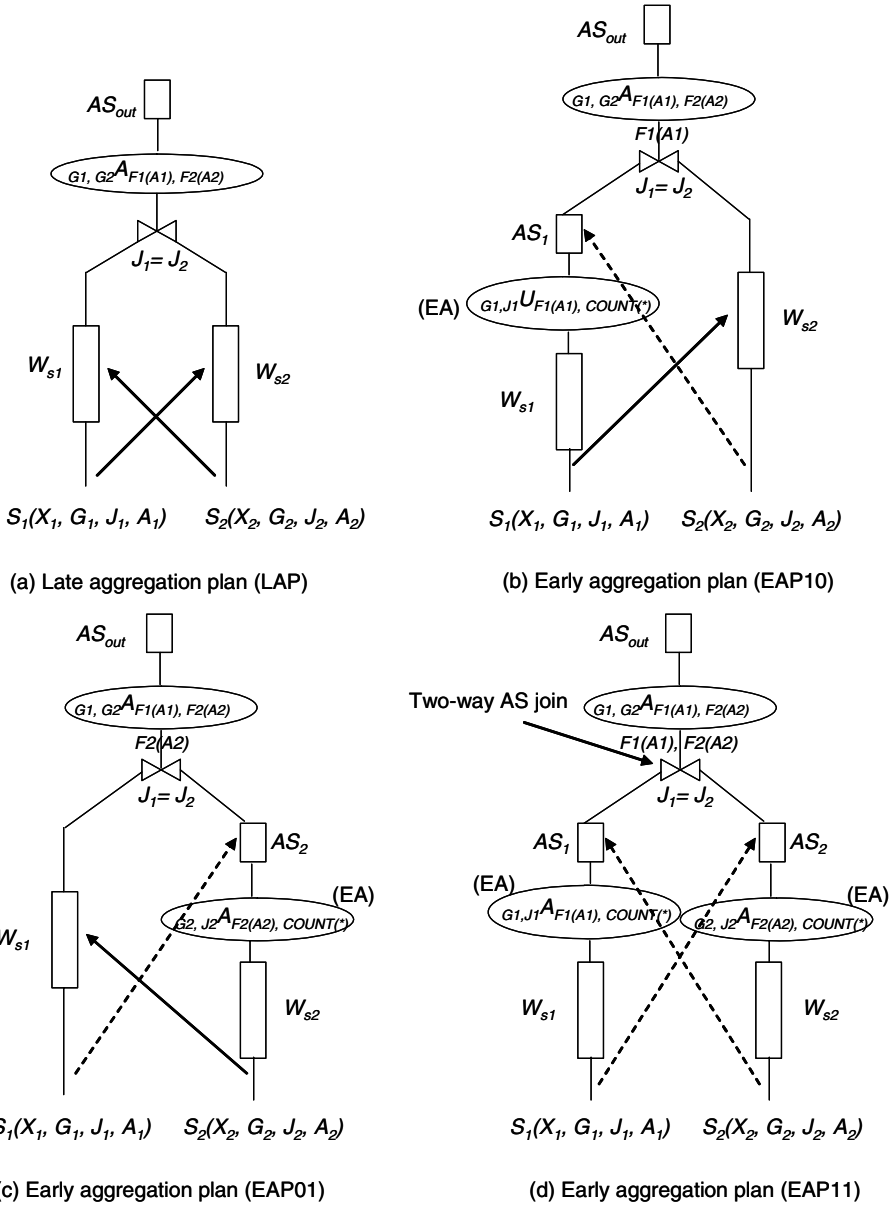
If the EA operator is placed on a stream which has all the aggregation attributes in the query, then use the aggregation function in the query as the aggregation function of the EA operator. If the stream has only some (not all) aggregation attributes in the query, then use both the aggregation function in the query and COUNT(\*) as the aggregation function of the EA operator. Otherwise, use only COUNT(\*) as the aggregation function of the EA operator.  $\square$

Note that these transformation rules are far more compact and yet more general than the transformation rules presented in the database case [20, 21]. For instance, our rules are applied to each stream without regard to other streams, whereas the database rules are applicable only if certain constraints hold among relations, such as a referential integrity [20] or a functional dependency [21]. Moreover, our rules do not depend on which streams the grouping attributes, join attributes and aggregation attributes belong to. This thus covers all the cases considered in [21].

Figure 3 illustrates transformations of an aggregation join query considering the most general case of a two-way join, i.e., both streams have grouping attributes and aggregation attributes. (This case can be reduced to special cases as considered in [20], in which only one stream has grouping (or aggregation) attributes, by setting one of the grouping (or aggregation) attributes empty.) The figure shows all four possible QEPs. The reader is asked to verify that these illustrated transformations conform to the rule for constructing an EA operator.

With the window join and AS join in place, the four QEPs in Figure 3 are equivalent. (See [22] for a proof of the equivalence.) The following example illustrates the equivalence of two QEPs shown in Figures 3a and 3c.

*Example 3 (LAP vs. EAP).* Consider the QEPs shown in Figures 3a and 3c, and assume that both aggregation function  $F_1$  and  $F_2$  are SUM. Then, the query output  $AS_{out}$  is updated in each QEP as follows. In LAP (Figure 3a), a window join is performed from  $S_1$  to  $W_{S_2}$ . Assume that, for each tuple  $s_1(x_1, g_1, j_1, a_1) \in W_{S_1}^+$ , the tuple matches  $c$  tuples,  $\{s_2(x_2, g_2, j_2, a_2), i = 1, 2, \dots, c\}$  where  $s_2.j_2 = s_1.j_1$ , in  $W_{S_2}$ . Then, the window join generates  $c$  output tuples,  $\{u(x_1, g_1, j_1, a_1, x_2, g_2, j_2, a_2) | i = 1, 2, \dots, c, j_2 = j_1\}$ . Further assume that, among these  $c$  output tuples,  $c_g$  tuples have the same value,  $g_2$ , for  $g_2$ , and hence the same value,  $(g_1, g_2)$ , for  $(g_1, g_2)$ . Then, for a tuple in  $AS(G_1, G_2, SUM(A_1), SUM(A_2))$  whose value of  $(G_1, G_2)$  equals  $(g_1, g_2)$ , the value of  $SUM(A_1)$  is increased by  $a_1 * c_g$  and the value of  $SUM(A_2)$  is increased by  $v_2 = \sum a_2, i = 1, 2, \dots, c_g$ . In the second QEP (EAP in Figure 3c), an AS join is performed from  $S_1$  to  $AS_2$ . Assume that, for each tuple  $s_1(x_1, g_1, j_1, a_1)$ , it matches one tuple,  $l_2(g_2, j_2, v_2, c_g)$  where  $j_2 = j_1$  and  $\sum a_2, i = 1, 2, \dots, c_g$ , in  $AS_2(G_2, J_2, SUM(A_2), COUNT(*))$ . Then, the AS join generates an output tuple  $u(x_1, g_1, j_1, a_1 * c_g, g_2, j_1, v_2, c_g)$  (see Definition 5 for the coalesced value  $a_1 * c_g$ ).



$G_i$ : grouping attributes;  $J_i$ : join attributes;  $A_i$ : aggregation attributes;  $X_i$ : the other attributes;  $AS_{out}, AS_1, AS_2$ : aggregation sets.

An arrow from an input stream to the window of another stream denotes a window join (see Section 3) and an arrow from an input stream to the aggregation set of the window on another stream denotes an AS join (Definition 6).

Fig. 3. Transformations of aggregation (two-way) join QEPs on data streams

**Algorithm:** QEP\_Execution

Inputs:

- $W_{S_1}, W_{S_2}, \dots, W_{S_m}$ : join windows.
- $AS_{i_1}, AS_{i_2}, \dots, AS_{i_p}$ : EA output aggregation sets ( $p \leq m$ ).
- $W_{S_k}^+$ : window increment on  $S_k$ .
- $W_{S_k}^-$ : window decrement on  $S_k$ .
- $AS_{out}$ : query output aggregation set.

Output:

- $AS_{out}$ : updated query output aggregation set.

Procedure:

Begin

For each tuple  $s_k$  in  $W_{S_k}^+$  {

1. If there exists an EA operator on  $S_k$ , then with  $s_k$  find its group in  $AS_k$  and update the aggregate value. (AS update on EA output)
2. Add  $s_k$  to  $W_{S_k}$ . (Window update)
3. With  $s_k$ , find matching tuples in either  $AS_j$  or  $W_{S_j}$  for each  $j = 1, 2, \dots, k-1, k+1, \dots, m$ , depending on whether an EA operator is placed on  $S_k$  (then  $AS_j$ ) or not (then  $W_{S_j}$ ). (Window join or AS join)
4. For each tuple produced in step 3 find its group in  $AS_{out}$  and update the aggregate value. (AS update on query output)

}

For each tuple  $r_k$  in  $W_{S_k}^-$  {

5. If there exists an EA operator on  $S_k$ , then with  $r_k$  find its group in  $AS_k$  and update the aggregate value. (AS update on EA output)
6. Remove  $r_k$  from  $W_{S_k}$ . (Window update)
7. With  $r_k$ , find matching tuples in either  $W_{S_j}$  or  $AS_j$  for each  $j = 1, 2, \dots, k-1, k+1, \dots, m$ , depending on whether an EA operator is placed on  $S_k$  (then  $AS_j$ ) or not (then  $W_{S_k}$ ). (Window join or AS join)
8. For each tuple produced in step 7 find its group in  $AS_{early}$  and update the aggregate value. (AS update on query output)

}

End

**Fig. 4.** A generic QEP-execution algorithm

This tuple is input to the AS update operator, which then makes the same update (i.e.,  $a_1 * c_g$  and  $v_2$ ) on the aggregation set  $AS$ .  $\square$

## 5.2 Generic Algorithm for Query Executions

Figure 4 outlines a high level algorithm for processing tuples with a multi-way join among  $m$  ( $m \geq 2$ ) streams  $S_1, S_2, \dots, S_m$ .<sup>3</sup> The algorithm is generic enough to cover any of the possible QEPs. It updates the output aggregation set  $AS_{out}$  for each tuple  $s_k$  in the window increment  $W_{S_k}^+$  and each tuple  $r_k$  in the window decrement  $W_{S_k}^-$ . The algorithm performs (1) AS updates on the output of an EA operator in steps 1 and 5 if there exists an EA operator on  $S_k$ , (2) window updates in steps 2 and 6, (3) either AS

<sup>3</sup> This algorithm processes tuples in pipelined fashion, but it may be queue-based as well. The query transformation works well with both types of algorithms.

joins or window joins in steps 3 and 7 depending on whether an EA operator is placed on  $S_k$ , and (4) AS updates on the query output  $AS_{out}$  in steps 4 and 8.

## 6 Performance Study

In this section, we study the performance of the proposed query transformations, with a focus on the QEP efficiencies. There are two objectives of the experiments: (1) examine the performance trends of the alternative QEPs for varying key parameter values; (2) show the cases of each alternative QEP being the most efficient one in relation to the parameter values. Section 6.1 describes the experimental setup, and Section 6.2 present the experiments conducted and their results.

### 6.1 Experimental Setup

We have built an operational prototype that implements the QEP execution algorithm (see Figure 4). The prototype has been written in Java 2 SDK 1.4.2, and runs on a Linux PC with Pentium IV 1.6GHz processor and 512MB RAM. For a join method, it supports hash join and nested loop join and for an aggregation method, it supports hash-based grouping. Additionally, it executes a join using *sliding* windows, of which tumbling and landmark windows are only special types (see Section 3).

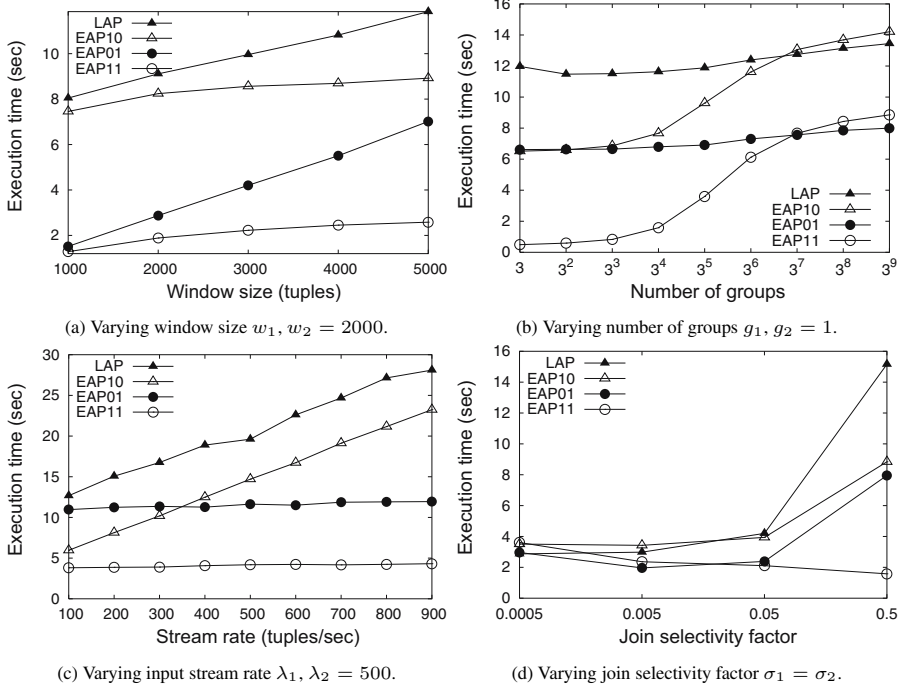
Inputs to the prototype are data streams generated using a data generator<sup>4</sup> (described below), the join arity (i.e., number of data streams) ( $m$ ), the size of each join window ( $w_1$ ,  $w_2$ ), and the QEP case number (0 for LAP, 1, 2, 3, ...,  $2^m - 1$  for EAPs). It then processes the input stream data according to the specified QEP and reports the execution time.

The data generator generates stream data sets as a sequence of tuples. Inputs to the data generator are the number of tuples in the data set, the number of attributes in the stream schema, the stream rate (i.e., number of tuples per second), the number of groups in the stream, and the number of distinct values of the join attribute. (A join selectivity factor equals the reciprocal of the number of distinct values of the join attribute.) Each tuple has a timestamp attribute, whose value is determined based on the stream rate. It also has other attributes such as join attribute, grouping attribute and aggregation attribute. Values of each of these attributes are assigned randomly with the uniform distribution. We use the string data type for grouping and join attributes and the integer data type for aggregation attribute.

### 6.2 Experiments and Results

In this section, we first investigate the efficiencies of alternative QEPs by varying streams statistics (i.e. stream rates, join selectivities, number of groups and window sizes). Then, we build showcases of different alternative QEPs being the most efficient ones. In all the experiments, the execution time of a QEP is reported per time-unit (second). For this, we measure the execution time for tuples arriving in 1000 milliseconds. We run each experiment three times, for one time-unit at each run, and compute the average execution time (in seconds).

<sup>4</sup> The data generator allows us to vary the input stream statistics so that we can evaluate the efficiencies of alternative QEPs with different input parameters.



EAP01: an EA operator on  $S_2$  only, EAP10: an EA operator on  $S_1$  only, EAP11: EA operators on both  $S_1$  and  $S_2$   
 Default setting:  $\lambda_1 = 300, \lambda_2 = 300, w_1 = 5000, w_2 = 5000, g_1 = 150, g_2 = 1, \sigma_1 = 0.1, \sigma_2 = 0.1$

**Fig. 5.** Execution times of QEPs (using two-way nested loop join)

### Experiment 1: Query Execution Costs for Varying Stream Statistics

In each set of experiments, we measure the execution time of QEPs by varying one of the four pairs of parameters: (1) window size ( $w_1, w_2$ ), (2) number of groups ( $g_1, g_2$ ), (3) stream rate ( $\lambda_1, \lambda_2$ ), and (4) join selectivity factor ( $\sigma_1, \sigma_2$ ). Furthermore, for each pair of parameters we vary only the parameters of stream  $S_1$  (i.e.,  $w_1, g_1, \lambda_1$  and  $\sigma_1$ ), since the QEPs are symmetric.

Figure 5 shows the results from the four sets of experiments. The curves in each graph represents the execution times of four alternative QEPs (i.e., LAP, EAP01, EAP10, EAP11). Due to space limit, we present the results for two-way nested-loop joins only; the results from using hash joins and three-way joins show the same trends. Interested readers are referred to [22] for the results of more comprehensive experiments.

Let us now examine the results of each set of experiments for varying each of the four parameters (i.e., window size, number of groups, stream rate, join selectivity factor). In the following discussion, we use the name of a QEP (i.e., LAP, EAP01, EAP10, EAP11) to refer to the cost of executing the QEP.

In Figure 5a, as window size  $w_1$  increases, LAP and EAP01 increase linearly. In contrast, EAP10 and EAP11 initially increase linearly but then stay constant as  $w_1$  exceeds 2000. The reason for this is as follows. In LAP and EAP01, there is no EA operator placed on  $S_1$  and, therefore, the execution time depends on  $w_1$  only. Unlike



this, in EAP10 and EAP11 which have an EA operator placed on  $S_1$ , the cost stops depending on  $w_1$  but starts depending on aggregation set size (i.e.,  $|AS_1|$ ) (which is fixed) when  $w_1$  is greater than 2000. Additionally, EAPs are always better than LAP because in the experiment, aggregation set sizes  $|AS_1|$  and  $|AS_2|$  are set smaller than window size  $w_1$  and  $w_2$ .

Figure 5b shows the results of varying the number of groups on stream  $S_1$  by a factor of 3. In this experiment, there is no grouping attribute in stream  $S_2$  and, thus,  $g_2$  equals 1. In the figures, as the number of groups  $g_1$  increases, EAP10 increases and approaches LAP and, likewise, EAP11 increases and approaches EAP01. The initial increase of EAP10 and EAP11 is caused by the increase of the aggregation set size ( $|AS_1|$ ). But, as  $g_1$  becomes large enough ( $g_1 = 3^8$ ),  $|AS_1|$  stops depending on  $g_1$  and starts depending on  $|W_1|$ . As a result, EAP10 and EAP11 lose the advantage of placing an EA operator on  $S_1$ .

Figure 5c shows the results of varying stream rate of  $S_1$  while fixing the stream rate of  $S_2$ . In the figures, as  $\lambda_1$  increases, the costs of all four QEPs increase linearly but LAP and EAP10 increase faster than EAP01 and EAP11. The reason is that the

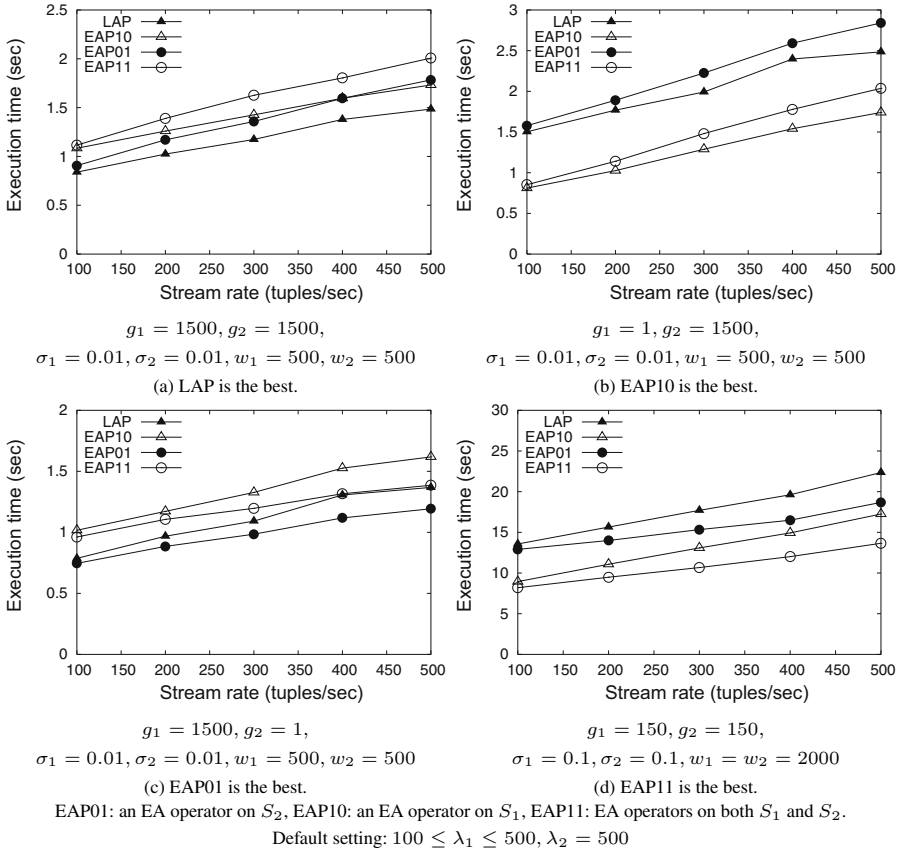


Fig. 6. Showcases of different best QEPs (using two-way nested loop join)

per-tuple processing time for each tuple from  $S_1$  in EAP01 and EAP11 is shorter than that in LAP and EAP10, as it takes shorter to find matching tuples in an aggregation set  $AS_2$  instead of  $W_2$ .

The results in Figure 5d is for the case of varying join selectivity factors  $\sigma_1$  and  $\sigma_2$  ( $\sigma_1 = \sigma_2$ ). As the join selectivity factors increase, the costs of all QEPs increase except for EAP11. The reason is that the cost of window joins in LAP, EAP01 and EAP10 depends on the join selectivity factors but this is not the case for the aggregation set join in EAP11. Moreover, as the join selectivity factors increase,  $|AS_1|$  and  $|AS_2|$  decrease, thus the cost of EAP11 decreases.

## Experiment 2: Showcases of Different Best QEPs

Intuitively, the advantage of an early aggregation is more highlighted when the number of groups ( $g_i$ ) is smaller or the join selectivity factor ( $\sigma_i$ ) is larger or the window size ( $w_i$ ) is larger. Specifically, a decrease in the number of groups leads to a decrease of an EA output aggregation set size in an EAP, thus enhancing the benefit of join reduction due to early aggregation; on the other hand, an increase in the join selectivity factor or an increase in the window size leads to an increase of join output tuples in an LAP, thus increasing the penalty of late aggregation.

Figure 6 shows the cases different QEPs are chosen as the most efficient one. The result confirms the intuition. That is, EAP11 is the best when both  $g_1$  and  $g_2$  are low, EAP10 is the best when  $g_1$  is low and  $g_2$  is high, EAP01 is the best when  $g_1$  is high and  $g_2$  is low, and LAP (or, “EAP00”) is the best when both  $g_1$  and  $g_2$  are high. In Figure 6d the scale of the graph is larger than those in the other figures (Figure 6a, b and c). This is because the execution times are much longer due to the higher join selectivity factors and larger window sizes used to generate the showcase.

## 7 Conclusion

In this paper, we focused on the problem of continuously processing an aggregation join queries on data streams using query transformations. We proposed an incremental query processing model with two key stream operators: aggregation set update and aggregation set join. Based on the processing model, we presented query transformation rules to generate an early aggregation plan equivalent to a late aggregation plan. We then developed an algorithm for executing the query execution plans. Finally, we conducted a set of experiments to study the performances of alternative QEPs.

Query transformation has been studied extensively in databases but not in data streams. To our knowledge, this is the first work addressing query transformation on aggregation join queries. Our query transformation is compact and yet generic to be applicable to each stream separately. The results of our experiments indicate that the query transformation indeed generates alternative QEPs of which the efficiencies are distinct enough to influence a stream query optimizer.

Query transformation is one step in query optimization. Thus, the future work includes to develop a comprehensive framework that integrates other components such as cost models for alternative QEPs and efficient search algorithms for finding an optimal QEP. The optimizer can run adaptively to switch to a more efficient QEP when input statistics (e.g., stream rates, number of groups, join selectivity) change significantly.

## References

1. Kang, J., Naughton, J.F., Viglas, S.D.: Evaluating window joins over unbounded streams. In: Proceedings of ICDE, Bangalore, India, pp. 341–352. IEEE Computer Society Press, Los Alamitos (2003)
2. Golab, L., Ozsu, M.T.: Processing sliding window multi-joins in continuous queries over data streams. In: Proceedings of VLDB, pp. 500–511. ACM Press, New York (2003)
3. Das, A., Gehrke, J., Riedewald, M.: Approximate join processing over data streams. In: Proceedings of ACM SIGMOD, San Diego, California, pp. 40–51. ACM Press, New York (2003)
4. Li, J., Maier, D., Tufte, K., Papadimos, V., Tucker, P.A.: Semantics and evaluation techniques for window aggregates in data streams. In: Proceedings of SIGMOD, pp. 311–322. ACM Press, New York (2005)
5. Ayad, A., Naughton, J.F.: Static optimization of conjunctive queries with sliding windows over infinite streams. In: Proceedings of ACM SIGMOD, pp. 419–430. ACM Press, New York (2004)
6. Arasu, A., Widom, J.: Resource sharing in continuous sliding-window aggregates. In: Proceedings of VLDB, pp. 336–347. Morgan Kaufmann, San Francisco (2004)
7. Arasu, A., Manku, G.S.: Approximate counts and quantiles over sliding windows. In: Proceedings of PODS, pp. 286–296. ACM Press, New York (2004)
8. Ding, L., Rundensteiner, E.A.: Evaluating window joins over punctuated streams. In: Proceedings of CIKM, pp. 98–107. ACM Press, New York (2004)
9. Ghanem, T.M., Hammad, M.A., Mokbel, M.F., Aref, W.G., Elmagarmid, A.K.: Incremental evaluation of sliding-window queries over data streams. *IEEE TKDE* 19(1), 57–72 (2007)
10. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proceedings of ACM SIGMOD, Madison, Wisconsin, pp. 1–16. ACM Press, New York (2002), doi:10.1145/543613.543615
11. Gehrke, J., Korn, F., Srivastava, D.: On computing correlated aggregates over continual data streams. *SIGMOD Record* 30(2), 13–24 (2001), doi:10.1145/376284.375665
12. Babu, S., Arasu, A., Widom, J.: CQL: A language for continuous queries over streams and relations. In: Lausen, G., Suci, D. (eds.) *DBPL 2003*. LNCS, vol. 2921, pp. 1–19. Springer, Heidelberg (2004)
13. Viglas, S., Naughton, J.F., Burger, J.: Maximizing the output rate of multi-way join queries over streaming information sources. In: Proceedings of VLDB, pp. 285–296 (2003)
14. Urhan, T., Franklin, M.J.: Xjoin: A reactively-scheduled pipelined join operator. In: *IEEE Data Engineering Bullentin*, pp. 27–33. IEEE Computer Society Press, Los Alamitos (2000)
15. Dobra, A., Garofalakis, M., Gehrke, J., Rastogi, R.: Processing complex aggregate queries over data streams. In: Proceedings of ACM SIGMOD, Madison, Wisconsin, pp. 61–72. ACM Press, New York (2002)
16. Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M.: Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In: Proceedings of VLDB, pp. 79–88. Morgan Kaufmann, San Francisco (2001)
17. Guha, S., Koudas, N.: Approximating a data stream for querying and estimation: Algorithms and performance evaluation. In: Proceedings of ICDE, pp. 567–579 (2002)
18. Vitter, J.S., Wang, M.: Approximate computation of multidimensional aggregates of sparse data using wavelets. In: Proceedings of ACM SIGMOD, pp. 193–204. ACM Press, New York (1999)
19. Jiang, Z., Luo, C., Hou, W.-C., Yan, F., Zhu, Q.: Estimating aggregate join queries over data streams using discrete cosine transform. In: Bressan, S., Küng, J., Wagner, R. (eds.) *DEXA 2006*. LNCS, vol. 4080, pp. 182–192. Springer, Heidelberg (2006)

20. Chaudhuri, S., Shim, K.: Including group-by in query optimization. In: Proceedings of VLDB, pp. 354–366. Morgan Kaufmann, San Francisco (1994)
21. Yan, W.P., Larson, P.-Å.: Eager aggregation and lazy aggregation. In: Proceedings of VLDB, pp. 345–357. Morgan Kaufmann, San Francisco (1995)
22. Tran, T.M., Lee, B.S.: Transformation of continuous aggregation join queries over data streams. Technical Report CS-07-02, Department of Computer Science, University of Vermont (2007)
23. Abadi, D.J., 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. *The VLDB Journal* 12(2), 120–139 (2003)
24. Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G.S., Olston, C., Rosenstein, J., Varma, R.: Query processing, approximation, and resource management in a data stream management system. In: Proceedings of CIDR, pp. 22–34 (2003)
25. Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S.R., Reiss, F., Shah, M.A.: TelegraphCQ: continuous dataflow processing. In: Proceedings of ACM SIGMOD, San Diego, California, pp. 668–668. ACM Press, New York (2003)
26. Chen, J., DeWitt, D.J., Tian, F., Wang, Y.: NiagaraCQ: a scalable continuous query system for internet databases. In: Proceedings of ACM SIGMOD, Dallas, Texas, United States, pp. 379–390. ACM Press, New York (2000)
27. Bai, Y., Thakkar, H., Wang, H., Luo, C., Zaniolo, C.: A data stream language and system designed for power and extensibility. In: Proceedings of CIKM, pp. 337–346 (2006)
28. Hammad, M.A., Mokbel, M.F., Ali, M.H., Aref, W.G., Catlin, A.C., Elmagarmid, A.K., Eltabakh, M., Elfeky, M.G., Ghanem, T.M., Gwadera, R., Ilyas, I.F., Marzouk, M.S., Xiong, X.: Nile: A query processing engine for data streams. In: Proceedings of ICDE, pp. 851–863. IEEE Computer Society Press, Los Alamitos (2004)
29. Sullivan, M.: Tribeca: A stream database manager for network traffic analysis. In: Proceedings of VLDB, pp. 594–606. Morgan Kaufmann, San Francisco (1996)
30. Cranor, C., Johnson, T., Spataschek, O., Shkapenyuk, V.: Gigascope: a stream database for network applications. In: Proceedings of ACM SIGMOD, San Diego, California, pp. 647–651. ACM Press, New York (2003)
31. Srivastava, U., Widom, J.: Memory-limited execution of windowed stream joins. In: Proceedings of VLDB, pp. 324–335. Morgan Kaufmann, San Francisco (2004)
32. Hammad, M.A., Aref, W.G., Elmagarmid, A.K.: Stream window join: Tracking moving objects in sensor-network databases. In: Proceedings of SSDBM, pp. 75–84 (2003)
33. Ojewole, A., Zhu, Q., Hou, W.-C.: Window join approximation over data streams with importance semantics. In: Proceedings of CIKM, pp. 112–121 (2006)
34. Zhang, R., Koudas, N., Ooi, B.C., Srivastava, D.: Multiple aggregations over data streams. In: Proceedings of ACM SIGMOD, pp. 299–310. ACM Press, New York (2005)
35. Tatbul, N., Zdonik, S.B.: Window-aware load shedding for aggregation queries over data streams. In: Proceedings of VLDB, pp. 799–810 (2006)
36. Babcock, B., Datar, M., Motwani, R.: Load shedding for aggregation queries over data streams. In: Proceedings of ICDE, p. 350. IEEE Computer Society Press, Los Alamitos (2004)
37. Considine, J., Li, F., Kollios, G., Byers, J.W.: Approximate aggregation techniques for sensor databases. In: Proceedings of ICDE, pp. 449–460. IEEE Computer Society Press, Los Alamitos (2004)
38. Yan, W.P., Larson, P.-Å.: Performing group-by before join. In: Proceedings of ICDE, pp. 89–100. IEEE Computer Society Press, Los Alamitos (1994)

# Continuous Constraint Query Evaluation for Spatiotemporal Streams

Marios Hadjieleftheriou<sup>1</sup>, Nikos Mamoulis<sup>2,\*</sup>, and Yufei Tao<sup>3,\*\*</sup>

<sup>1</sup> AT&T Labs Inc., 180 Park Avenue, Florham Park, NJ 07932  
marioh@research.att.com

<sup>2</sup> Department of Computer Science, University of Hong Kong, Pokfulam Road, Hong Kong  
nikos@cs.hku.hk

<sup>3</sup> Department of Computer Science and Engineering, Chinese University of Hong Kong, Sha Tin, New Territories, Hong Kong  
taoyf@cse.cuhk.edu.hk

**Abstract.** In this paper we study the evaluation of *continuous constraint queries* (CCQs) for spatiotemporal streams. A CCQ triggers an alert whenever a configuration of constraints between streaming events in space and time are satisfied. Consider, for instance, a server that receives updates from GPS-enabled agents that report their positions and other measurements (e.g., environmental readings). An example of CCQ is: “Alert whenever at least 5 readings closer than 5km to each other and within a time difference of 5 minutes report high pressures and low temperatures”. We model CCQs as Constraint Satisfaction Problems (CSPs) and develop solutions for their *continuous evaluation*. Our techniques (1) consider the fast arrival rate of incoming events, and (2) minimize the memory requirements, without using predefined window constraints, but by utilizing the structure of the queries. In order to show the merits of the proposed techniques, we implement a system prototype and evaluate it with real data.

## 1 Introduction

The recent advances in telecommunications have made it possible to collect unbounded streams of spatiotemporal information from various sources (GPS devices, sensors, etc.). Consider, for instance, a server which receives updates from GPS-enabled agents that continuously report their positions and other measurements (e.g., environmental readings). A real example of such a system is the Global Drifter Center [\[2\]](#) where a large number of buoys have been deployed in oceans all around the world. The buoys report various measurements at regular time-intervals in a streaming fashion while drifting in the water according to sea currents.

The large size and fast arrival rates of streaming data renders storage and off-line analysis infeasible. In addition, users are often interested in answering queries in an on-line, dynamic manner, as streaming data arrive. In this paper, we study the processing of *continuous constraint queries* (CCQs), which trigger an alert whenever a configuration

---

\* Nikos Mamoulis was supported by grant HKU 7160/05E from Hong Kong RGC.

\*\* Yufei Tao was supported by grant CUHK 1202/06 from Hong Kong RGC.

of *constraints* between streaming events in space and time are satisfied. For instance, consider the following query: “Alert whenever at least 5 readings closer than 5 km to each other and within a time difference of 5 minutes report high pressures and low temperatures”. CCQs facilitate the automatic and continuous monitoring of interesting combinations of spatiotemporal events.

There is an abundance of interesting and useful queries that can be formulated as a combination of diverse types of constraints. The constraints may capture both spatial (e.g., proximity, intersection, and containment) and temporal (e.g., during, before, and after) relationships between a large number of events, as well as to other event characteristics (e.g., measurements like velocity and temperature).

CCQs are similar to traditional triggers in database systems, in that they should be evaluated every time a new event arrives. A CCQ states that if the new event forms a specific spatiotemporal configuration (e.g., a number of abnormal thermal indications in space and time) with past events, an alert should be triggered. The important difference to traditional triggers is that the constraints are spatiotemporal. First, the methods for evaluating them are related to spatial and spatiotemporal query processing. Second, typically, there is no need to keep the whole history of events in memory, since the spatiotemporal constraints define the essential intervals in time and space, for which information needs to be maintained. For instance, for a CCQ asking for a number of abnormal thermal indications within 10 minutes in time, we need not keep events in memory older than 10 minutes, since they could not form query results with current or future data.

In this paper we present a system with the following characteristics: (1) a stream of events arrives on a central server at fast rates; (2) events are associated with spatiotemporal properties, and other, alphanumeric measurements; (3) users register CCQs, and (4) the system triggers alerts whenever a newly arriving event together with past events form a result of a CCQ.

Our main focus is on critical applications where it is essential not to dismiss any query alerts and, in addition, not to produce any false alarms. Hence, we propose techniques that produce exact query results, raising alerts if and only if a combination of events satisfies all constraints for a given query. This is accomplished by guaranteeing that all useful events (the ones that can contribute to a query answer) are stored in main memory during processing. Since storing the complete event stream is not feasible for most practical applications, we introduce algorithms for determining *event expiration times* — computed by inspecting all query constraints related to a specific event — and establishing a time-instant after which the event will not possibly satisfy any constraints and can be deleted. We introduce a simple expiration time computation algorithm, as well as a tighter technique that guarantees that every event is kept in main memory for a very short amount of time. With this approach we minimize the peak amount of main memory that is consumed by the system. Finally, to show the merits of our architecture, we implement and evaluate a prototype for the proposed system.

## 2 Problem Formulation and Definitions

This section introduces a formal problem statement and some necessary definitions to simplify our analysis. As already discussed, a stream of events arrives on a central server

where each event carries a timestamp (or a time interval), spatial (i.e., geometric) properties, and other properties of a simple type (e.g., alphanumeric). Users register queries that pose various constraints between properties of a possibly large number of events. The goal is to find in real-time tuples of event instances that have already appeared on the stream and satisfy all query constraints, in which case an alert is triggered. First, we formally define an event instance. Then, we present a formal way of expressing generic constraint queries between events. Finally, we discuss algorithms for evaluating such queries continuously, as new events arrive on the stream.

A spatiotemporal stream is a never ending sequence of events  $\mathcal{S} = \langle e_1 e_2 \cdots e_n \cdots \rangle$  where:

**Definition 1.** An event  $e$  is a tuple  $\{t, r, p\}$ ,  $e.t$  are the temporal properties of the event (e.g., a time-interval or the arrival time),  $e.r$  the geometric properties (e.g.,  $e.r \in \mathbb{R}^d$  for a  $d$ -dimensional point), and  $e.p$  a set of application dependent properties (e.g., temperature, velocity, other measurements).

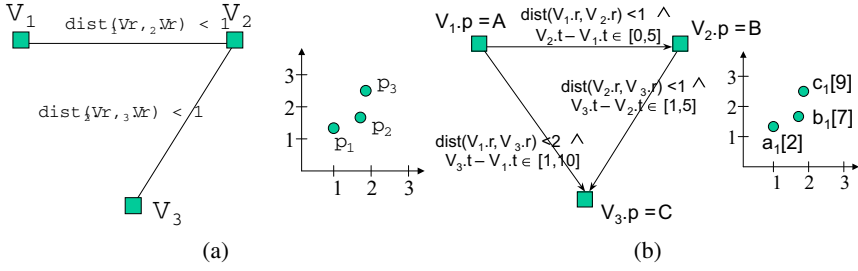
**Table 1.** Notation used throughout the paper

Symbol	Meaning
$e$	Streaming event
$t$	Time-instant or time-interval
$r$	Geometric properties
$p$	Generic properties
$\odot$	Spatial predicate
$V, \mathcal{V}$	Variable, set of variables
$D$	Variable domain
$C, \mathcal{C}$	Constraint, set of constraints
$\mathcal{G} = (\mathcal{V}, \mathcal{C})$	Constraint graph
$\{e_i, e_j\} \propto C$	Tuple $\{e_i, e_j\}$ satisfies constraint $C$

We can naturally express queries with arbitrary constraints between a large number of events as *Constraint Satisfaction Problems* (CSP) [33,31,4,16]. Constraint satisfaction is a paradigm that can capture a wide variety of applications from AI, engineering, databases, and other disciplines. A CSP is defined by a set of variables, each of which has a finite set of potential values, and a set of constraints between these variables. CSPs that contain constraints between at most two variables are called *binary*. A binary CSP is usually represented by a *Constraint Graph* (CG) where nodes correspond to variables and edges correspond to constraints. The basic goal in a CSP is to find one or all assignments of values to variables so that all constraints are satisfied. In our setting, the potential values of a given variable are event instances that have appeared on the stream, and the binary constraints are the spatial, temporal, and other possible constraints between events. More formally:

**Definition 2.** A binary Constraint Satisfaction Problem is:

1. A set of variables  $\mathcal{V} = \{V_1, \dots, V_m\}$ .
2. For each  $V_i$ , a finite domain  $D_i = \{e_1^i, \dots, e_{k_i}^i\}$  of  $k_i$  possible values.



**Fig. 1.** Examples of Constraint Graphs

3. A set of binary constraints. A constraint  $C_{i,j}$  is a relation of permitted values for the pair of variables  $\{V_i, V_j\}$ .

We say that an assignment of values  $\{V_i = e_i^i, V_j = e_k^j\}$  satisfies constraint  $C_{i,j}$  if  $\{e_i^i, e_k^j\}$  is in the relation  $C_{i,j}$ . A solution to a CSP is an assignment  $\{V_1 = e_{s_1}^1, \dots, V_m = e_{s_m}^m\}$  such that for all  $1 \leq i, j \leq m$ , constraint  $C_{i,j}$  is satisfied.

Figure 1a illustrates a CG corresponding to a simple CSP with the domain of each variable being a collection of events (whose spatial properties are 2D point locations) and constraints between variables being ‘distance within 1 space unit’. Note that the constraints in this graph are not expressed explicitly by allowed pairs of values, but implicitly with the use of spatial predicates. A solution to this CSP is the triplet  $\{V_1 = p_1, V_2 = p_2, V_3 = p_3\}$  of points shown next to the figure.

In our setting, the domain of each variable (e.g., the set of events that correspond to variable  $V_i$ ) can be implicitly defined by *unary* constraints, which correspond to selections in database languages. For example “events with measured temperature greater than 30°C”. Selection predicates can range from simple relational operators (e.g.,  $\leq$ ,  $<$ ,  $\geq$ ,  $>$ ,  $=$ ) to more advanced selection conditions (e.g., “select events with property  $A$ ”) or *metric* spatiotemporal constraints (i.e., relating *e.r* and *e.t* to fixed coordinates of space and time). In the rest, we use notation  $e_i^i \propto C_i$  to denote that event  $e_i^i$  satisfies unary constraint  $C_i$  of variable  $V_i$ .

As in our spatial CSP example, we can implicitly define binary constraints by spatial, temporal, and/or any other predicates between event properties *e.p*. Thus, in our setting, we can specialize the definition of constraints as follows:

**Definition 3.** A binary constraint  $C_{i,j}$  between variables  $\{V_i, V_j\}$  is a tuple  $\{t, \odot, p\}$ , where  $t$  is a temporal predicate (e.g., a time-interval),  $\odot$  is any binary spatial predicate (for example intersection or proximity), and  $p$  any non spatiotemporal constraint between properties *e.p*.

A binary constraint  $C_{i,j}$  is satisfied only when the assignment  $\{V_i = e_i^i, V_j = e_k^j\}$  satisfies all  $C_{i,j}.t$ ,  $C_{i,j}.\odot$ , and  $C_{i,j}.p$ . Notice that the temporal predicate semantics are application specific. In the simplest case, *e.t* can be a time-integer and  $C_{i,j}.t$  can be a time-interval containment (e.g.,  $e_i^i.t, e_k^j.t \in \mathbb{R}$  and  $C_{i,j}.t = \{e_k^j.t - e_i^i.t \in [t_{lb}, t_{ub}]\}$ )



but more general semantics can also be considered. Without loss of generality, we restrict our analysis to time-interval temporal constraints only. Also, for convenience, in the rest we use notation  $\{e_l^i, e_k^j\} \propto C_{i,j}$  to denote that assignment  $\{e_l^i, e_k^j\}$  satisfies binary constraint  $C_{i,j}$ . In addition, we allow negative temporal values as well, which can express chronological precedence between events — negative values refer to the past, while positive refer to the future. For example, time-interval  $[-3, 4]$  on edge  $(V_i, V_j)$  means that  $\{e_l^i, e_k^j\}$  satisfies the temporal constraint if  $e_k^j$  arrives 0–4 time-instants *after* or 0–3 time-instants *before*  $e_l^i$ . This generalization is useful since it helps express inferred constraints between events: Constraint  $[t_1, t_2]$  on edge  $(V_i, V_j)$  implies an *inverse* constraint  $[-t_2, -t_1]$  between  $(V_j, V_i)$ .

*Constraint inference* facilitates the derivation of complete constraint graphs where all possible constraints  $C_{i,j}$  are specified (i.e., cliques). In general, user queries may not provide such complete information (e.g., the graph of Figure 11a). Nevertheless, any partial query graph can be converted into a complete graph using spatial and temporal inference rules like inversion, composition and intersection [7,22,19]. For instance,  $dist(V_{1,r}, V_{2,r}) < 1$  and  $dist(V_{2,r}, V_{3,r}) < 1$  imply that  $dist(V_{1,r}, V_{3,r}) < 2$ , assuming that the geometric properties  $e.r$  are points. Such CG transformations are useful for several reasons: (1) for identifying if a query is unsatisfiable by discovering negative cycles; such queries can be ignored, (2) in order to tighten existing constraints and make them more selective, and (3) to simplify the proposed algorithms and in some cases improve query evaluation performance (as will become clear in later sections). The interested reader can refer to [7,22,19] for more details on temporal and spatial inference, which are beyond the scope of this paper.

Figure 11b shows a complete CG with spatiotemporal constraints (inverse edges are omitted for clarity). Nodes are annotated with unary constraints, which are selections on the non-spatial properties for simplicity. In practice  $V_{1,p} = A$  could be  $V_{1.temperature} > 30^\circ C$ . We denote events for which  $e.p = A$  by  $a_1, a_2$ , etc. Thus event  $a_1$  on the right of Figure 11b has  $a_{1,p} = A$ , event  $b_1$  has  $b_{1,p} = B$ , etc. The *continuous constraint query* (CCQ) that corresponds to the CG triggers an alert if any event  $a_i$  is close to an event  $b_j$  which arrives at most 5 time-instants after  $a_i$  (i.e.,  $C_{1,2}.t = [0, 5]$ ), and  $b_j$  is close to an event  $c_k$ , which arrives at least 1 and at most 5 time-instants after  $b_j$  (i.e.,  $C_{2,3}.t = [1, 5]$ ), and, equivalently, the inverse constraint  $C_{3,2}.t = [-5, -1]$  means that event  $b_j$  arrived at least 1 and at most 5 time-instants before  $c_k$ . These two constraints also infer that  $c_k$  must arrive at least 1 and at most 10 time-instants after  $a_i$ , illustrated by the inferred temporal constraint between variables  $V_1$  and  $V_3$ .

An extended SQL can be used to express CCQs. For instance, our example can be expressed in pseudo-SQL with spatial and temporal predicates, as follows:

```
CREATE TRIGGER collision
FOR E as V1, E as V2, E as V3
WHEN V1.p = A AND V2.p = B AND V3.p = C
AND DISTANCE(V1.r, V2.r) < 1 AND V2.t - V1.t IN [0, 5]
AND DISTANCE(V2.r, V3.r) < 1 AND V3.t - V2.t IN [1, 5]
```

The continuous constraint query evaluation problem can be formulated as follows:

**Problem Statement:** Given a number of constraint queries, continuously evaluate their satisfiability as new events appear on the stream, and trigger alerts whenever a combination of events constitutes a solution.

Previous methods on processing complex spatiotemporal queries with multiple selections and joins employ spatiotemporal indexes in combination with constraint satisfaction algorithms [24,21]. Our problem is different in that data arrive continuously (instead of being stored in a database first) and that the queries should be evaluated continuously. In other words, the variable domains of the CSPs are not static but dynamically change over time. Therefore, CCQs fall into the class of queries described in [6], requiring special evaluation methods. In addition, our problem is different than handling traditional triggers in a DBMS [13], since CCQs need not be evaluated over the whole, past time horizon; As a result, we do not need to maintain the complete event history but, instead, we can apply rules that minimize the required space (to be discussed shortly). In the next section we describe our system prototype for registering and evaluating CCQs over continuous data streams.

### 3 System Architecture

The problem formulation presented in the previous section raises a number of interesting questions. We need to develop suitable data structures and algorithms for evaluating CSPs in real-time. In addition, we need to evaluate queries incrementally every time a new event appears on the stream by restricting the domain of each variable in the given CGs, to speed up execution. Finally, we must continuously identify and delete from the system events that cannot possibly belong to future solutions, in order to minimize main memory requirements.

For simplicity and clarity we make the following assumptions: First, we consider only transitory events — the information associated with each event is valid only for a specific time-instant or time-interval and the properties *e.p* (i.e., the associated measurements) of an event remain static throughout its interval. In addition, we restrict our analysis only to streams of events with measurements that appear in chronological order. In other words, the events arrive in the same order as their temporal properties define. If this is not the case, then a large enough buffer could be used in order to rank events according to their arrival time-instants first. The size of the buffer can be determined according to the maximum expected delay of an arrival, which in most cases is a system characteristic. Finally, we assume that all registered query CGs have been transformed into complete constraint graphs. A variation of the Floyd-Warshall algorithm as it appears in [22] can be used for temporal constraints. The same algorithm can also be applied for spatial constraints, using the inference rules of [19].

#### 3.1 CCQ Evaluation

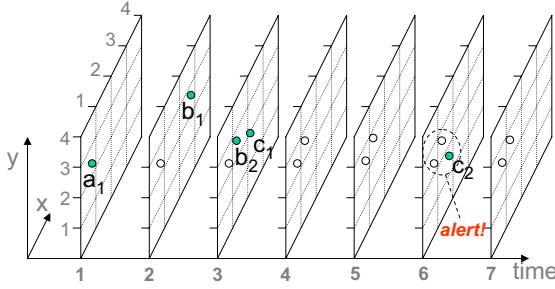
We first discuss the basic algorithm used to continuously evaluate constraint queries, in order to identify the operations that must be supported by our system. When a new event *e* arrives it might qualify for a variable domain in a registered CCQ, and therefore it can

be part of a potential solution. Thus, initially we need to evaluate the unary constraints of all variables of all registered queries, in order to identify CGs related with  $e$ . Let  $V_i$  be such a variable in a query corresponding to CG  $\mathcal{G}$  (i.e.,  $e \propto C_i$  in  $\mathcal{G}$ ).  $V_i$  is related through binary constraints to all other variables in  $\mathcal{G}$  and these constraints refer to the future, the past, or both temporal directions. For example,  $V_2$  in the CCQ of Figure 1b is connected to a past variable  $V_1$  (since  $C_{2,1}.t = [-5, 0]$ ) and a future variable  $V_3$  (since  $C_{2,3}.t = [1, 5]$ ).

Let  $\mathcal{V}_i^-$  and  $\mathcal{V}_i^+$ , be the sets of past and future variables to  $V_i$ . Note that a variable can be in both  $\mathcal{V}_i^-$  and  $\mathcal{V}_i^+$ . If the subgraph  $\mathcal{G}'$  containing  $V_i = e$  and  $\mathcal{V}_i^-$  is satisfiable,  $e$  is interesting for two reasons. First,  $e$  triggers an alert if  $V_i \cup \mathcal{V}_i^- = \mathcal{V}$  (i.e.,  $\mathcal{G}' = \mathcal{G}$ ). For example, variable  $V_3$  in the graph of Figure 1b has past variables only (i.e.,  $V_3 \cup \mathcal{V}_3^- = \mathcal{V}$ ). Thus, an event  $e$  with  $e.p = C$  may trigger an alert if there exist past events consistent to  $e$  and the subgraph  $\mathcal{G}'$ , containing  $V_1, V_2$  and their constraints. Second, if  $\mathcal{V}_i^+ \neq \emptyset$ , it is possible for  $e$  to participate in an alert in the future when variables in  $\mathcal{V}_i^+$  get values from events that are consistent with  $\mathcal{G}'$ . For example, variable  $V_2$  in the graph of Figure 1b has a future variable ( $V_3$ ). Thus, an event  $e$  with  $e.p = B$ , which makes the subgraph containing  $V_1$  and  $V_2$  satisfiable, can participate in an alert with some future event that instantiates  $V_3$ .

Thus, if  $\mathcal{G}'$  is satisfiable and  $\mathcal{V}_i^+ \neq \emptyset$ , we say that  $e$  is *useful* and we keep it in the system for potential future inclusion in an alert. The question now is *how long* is it essential to keep  $e$  for, until we decide that it cannot be part of a future solution? We assign  $e$  an *expiration* time  $e.X$ , after which  $e$  becomes obsolete and should be deleted. Intuitively, the expiration time cannot be longer than the longest interval length of temporal constraints that use  $e$ . This is a worst case upper bound, and we will see later on that it can be substantially improved.

Algorithm 1 summarizes the procedure for handling a new event  $e$  that arrives in the system. Initially, the expiration time is set to its time-instant  $e.t$ ; if the event is not found useful by the algorithm, it will be immediately deleted. All relevant CG and variables to  $e$  are then retrieved. A variable  $V_i$  in a CG  $\mathcal{G}$  is relevant if  $e \propto C_i$ , as already discussed. For each such variable, we find the set  $\mathcal{V}_i^-$  of past only variables and compute their domains  $D_i$  according to the past events stored in the system. The domain of a variable  $V_j \in \mathcal{V}_i^-$  is determined such that (1) its values are consistent with the assignment  $V_i = e$  (for this, constraint  $C_{i,j}$  is used), and (2) they are consistent with  $C_j$ , i.e., the unary constraint of  $V_j$ . If a domain of a variable is empty, we know that the assignment  $V_i = e$  for query  $\mathcal{G}$  is inconsistent. Otherwise, we solve the CSP for graph  $\mathcal{G}$  only if all variables in  $\mathcal{G}$  are consistent with  $e$  (and thus if  $V_i$  is chronologically last). If a solution is found (line 12), the algorithm generates an alert. Finally, the expiration time of  $e$  is updated accordingly, if  $V_i$  has any future variables ( $\mathcal{V}_i^+ \neq \emptyset$ ) and if the past only variables are consistent with  $V_i$ . Otherwise, the event can be deleted, since these variables can never be satisfied. Algorithm 1 sets the expiration time as the maximum required by outgoing edges to future variables (i.e., the maximum upper bound of the temporal constraints linking the current variable with future variables). If there are many queries or variables that may use  $e$  in the future, the expiration time is the maximum timestamp determined by all of them. Later, we will discuss alternative policies that result in tighter expiration times and reduce the memory requirements. In addition, we



**Fig. 2.** A continuous query evaluation example

will discuss the details for storing and indexing events, computing variable domains, solving constraint graphs, and managing event expirations.

---

**Algorithm 1.** HandleNewEvent( $e$ : event,  $Q$ : queries)

---

```

1:  $e.X := e.t$ ;
2: Get all  $(V_i, \mathcal{G})$  pairs such that  $\mathcal{G} \in Q, V_i \in \mathcal{G}, e \propto C_i$ ;
3: for each  $(V_i, \mathcal{G})$  do
4:    $V_i.sat := \text{true}$ ;
5:   for each  $V_j \in \mathcal{V}_i^-$  do
6:     use  $V_i = e, C_{i,j}$ , and  $C_j$  to compute  $D_j$ ;
7:     if  $D_j = \emptyset$  then
8:        $V_i.sat := \text{false}$ ;
9:       break; ▷ break  $V_j$  for-loop
10:  if  $V_i.sat$  and  $\mathcal{V}_i^- \cup V_i = \mathcal{G}.V$  then
11:    Solve  $\mathcal{G}$ 
12:    if  $\mathcal{G}$  is satisfiable then alert solution
13:  if  $\mathcal{V}_i^+ \neq \emptyset$  and  $\forall V_j \in \mathcal{V}_i^- / \mathcal{V}_i^+ : D_j \neq \emptyset$  then
14:     $e.X := \max\{e.X, e.t + \max_{V_j \in \mathcal{V}_i^+} (C_{i,j}.t_{ub})\}$ ;
15: if  $e.X > e.t$  then Store( $e$ ); ▷ event is useful
    
```

---

### 3.2 An example

In this section we present a simple example that clarifies the rational behind Algorithm 1. Let us assume that only the CCQ shown in Figure 1b has been registered with the system. Suppose that the arrivals on the stream follow the sequence shown in Figure 2. Event  $a_1$  (for which  $a_1.p = A$ ) arrives at time-instant 1 and  $a_1 \propto C_1$ . The event should be stored as a future candidate (i.e., it satisfies the unary constraint of query variable  $V_1$ ). An *expiration time* of 11 (due to  $C_{1,3}.t = [0, 10]$ ) is assigned to  $a_1$ , after which  $a_1$  will be deleted from main memory in order to save space.

Next, event  $b_1$  appears at time-instant 2, which is clearly related with variable  $V_2$ . Since  $V_1$  is the only past variable related with  $V_2$  we need to evaluate constraint  $C_{2,1}$ . Since  $a_1$  is the only stored event that satisfies  $C_1$  and  $\text{dist}(a_1, b_1) > 1$ ,  $C_{2,1} \odot$  is violated, and hence  $b_1$  can never participate in a query alert and can be deleted. At the next time-instant, events  $b_2$  and  $c_1$  arrive simultaneously. This time  $\text{dist}(a_1, b_2) < 1$  and

$\text{dist}(b_2, c_1) < 1$ , however,  $c_1$  violates the temporal constraint  $C_{3,2}.t = [-5, -1]$  and it can be deleted. Nevertheless,  $b_2$  can potentially belong to a future solution, hence, it needs to be retained and its expiration time becomes 8, given the current time 3 and the temporal constraint  $C_{2,3}.t = [1, 5]$ .

Finally, at time-instant 6 event  $c_2$  arrives, and since  $\text{dist}(c_2, b_2) < 1$  and  $a_1, b_2$  have not expired yet, an alert is triggered with solution tuple  $\{a_1, b_2, c_2\}$ . Since  $V_3$  has no future variables,  $c_2$  is deleted. On the other hand, events  $a_1$  and  $b_2$  need to be retained until their expiration times, since more events satisfying  $C_3$  might appear in the stream triggering additional alerts. After that time both events can be deleted.

### 3.3 A Detailed Analysis of the Proposed Framework

We now describe in detail the components of our system prototype that manages incoming events and evaluates CCQs based on Algorithm 1. Our system prototype consists of five basic components, shown schematically in Figure 3.

Queries are stored in memory-based constraint graph representations. Given a new event  $e$  on the stream, the Query Index retrieves all queries that contain at least one variable with an associated unary constraint that is satisfied by  $e$ . An important component is the Spatiotemporal Index, which is used for storing useful past event instances. Given a new event  $e$  and the spatiotemporal constraints associated with some related variable, the index is probed and all past events  $e'$  that qualify for these constraints given  $e$ , are retrieved. This will help populate all CG variable domains fast, with only a few candidates, instantly pruning a large number of unrelated events. An Expiration Time Array indexes events according to the time they should be deleted from the system and enables efficient deletion from the Spatiotemporal Index. Finally, a CSP Solver solves CGs given appropriate variable domains.

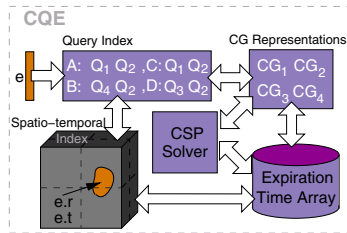


Fig. 3. A system prototype

*Constraint Graph Representation.* Given query  $Q$  with a complete directed graph  $\mathcal{G} = (\mathcal{V}, \mathcal{C})$ ,  $|\mathcal{V}| = m$ , we represent the graph using a 2-dimensional matrix  $M$  such that  $M[i, j] = C_{i,j}$ ,  $1 \leq i, j \leq m$ .

*Query Index.* Queries contain a number of unary variable predicates, corresponding to multiple properties. Given a large number of queries we must locate efficiently the ones that contain unary constraints satisfied by newly arriving events; these are the only queries that have to be considered for further processing (see line 2 of Algorithm 1).

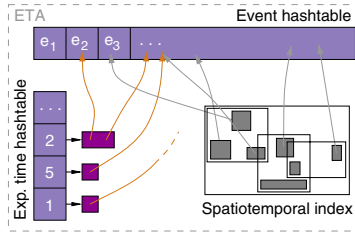
Many predicate indices have appeared in the literature [20,15,34,13,30]. Here we opt for a simpler approach with small memory requirements. In order to efficiently identify all variables  $V_i$  related to an incoming event  $e$ , we maintain a hash-table indexed by all known properties appearing in the unary predicates of registered queries. The data entries of the hash-table are pointers to the queries that have at least one variable with a unary constraint that is related to a specific property. Using the hash-table, we can locate fast all queries  $Q$  that contain a variable related to  $e$ . Then, by retrieving the unary constraint associated with that variable we can evaluate if  $e$  satisfies the constraint.

*Spatiotemporal Index.* The basic functionality of the Spatiotemporal Index is to store all useful past events that might contribute to a CG solution (i.e., an alert) in the future (see line 15 of Algorithm 1). The index acts as a filtering step that facilitates efficient evaluation of a query by substantially limiting variable domains before evaluation (lines 5-6 of Algorithm 1). A naive algorithm would consider all events indiscreetly as possible domain values. Instead, the Spatiotemporal index, given a new event  $e$ , returns only those stored events that satisfy the spatiotemporal predicates of the binary constraints associated with the past variables that are related to  $e$ . Any data or space partitioning structures can be used for that purpose, like the R-tree [10] and the Multi-Layer Grid File [28], both extended with a temporal dimension. For illustration purposes in the following discussion we assume that a 3D R-tree is used to index useful past events.

When an event  $e$  arrives, the domains of past variables that are related to  $V_i = e$  are computed (lines 5-6 of Algorithm 1), as follows. For each  $V_j \in \mathcal{V}_i^-$ , the spatiotemporal index is probed using the spatial  $e.r$  and temporal  $e.t$  properties of  $e$  in combination with predicates  $C_{j,i}.\odot$  and  $C_{j,i}.t$ . For instance, when event  $b_2$  arrives at time instant  $b_2.t = 2$  in the example of Figure 2 for constraint  $C_{2,1}$  a spatiotemporal range query with temporal extent  $[-3, 2]$  (since we are looking for events at least 0 and at most 5 instants before  $b_2.t$ ) and spatial extent a circle with radius 1 around the location of  $b_2$  is evaluated using the index. Since no events are contained in this range,  $D_1 = \emptyset$  and  $b_2$  will be found not useful and deleted from the system. The results produced by the spatiotemporal search are also filtered using the unary constraint of the variable whose domain  $D_j$  is being populated. This guarantees that the domains of all past variables become consistent with the assignment  $V_i = e$ , before solving the CSP on graph  $\mathcal{G}$ .

*Expiration Time Array.* While new events arrive on the stream, older events that have been stored in the index become obsolete. A structure is required that can index events in increasing expiration time, for easy deletion from the system. In Section 4.1 we will propose a technique that assigns tight expiration times to events. This technique calls for efficient operations for updating the expiration time of an event to a new value. We term this structure the Expiration Time Array.

In order to satisfy these requirements we use the following architecture (shown in Figure 4). We store all useful events in main memory and associate them with unique identifiers. In addition, we build a hash-table on the unique ids. The spatiotemporal index stores direct pointers to the events in the hash table. The Expiration Time Array is another hash-table with key being the expiration time  $t$  of the events, and data entries being arrays that contain pointers to events that have expiration time equal to  $t$ .



**Fig. 4.** The Expiration Time Array

Assume that a new event arrives on the stream and is inserted in the index. After the completion of the insertion operation the event is inserted in the Expiration Time Array according to the computed expiration time  $t$ . The appropriate hash table entry is located (or a new one is created if needed) and a pointer to the entry is inserted. The cost of this operation is dominated by the cost of an insertion to the index.

The Expiration Time Array contains possibly one array of pointers per future time-instant. Notice that the hash-table does not need to store empty arrays for time-instants that have no expiring events. So, we expect the structure to be fairly small in size. On the other hand, all arrays combined contain as many pointers as the total number of stored events.

Removing expiring events is straightforward. For every time-instant we locate the corresponding expiration array and remove all events contained therein from the spatial index and the event hash table. The deletions can happen in bulk and in a bottom-up fashion for efficiency [18]. This operation is very efficient since the main cost of this process is the update cost of the spatiotemporal index.

*CSP Solver.* The CSP Solver takes as input the constraint graph  $\mathcal{G}$  (line 11 of Algorithm 1) and finds a solution or deduces that the graph is insoluble. The general class of CSPs is NP-complete. However, a number of algorithms have been proposed in the literature that try to efficiently evaluate CSPs [4, 8, 14, 3]. The size of all possible value that can be solutions is the Cartesian product of the variable domains. The most popular search method uses backtracking; variables are instantiated sequentially and as soon as all variables relevant to a constraint have assumed a value, the satisfiability of the constraint is tested. If a partial instantiation violates any constraint, backtracking is performed to the most recently instantiated variable that still has alternatives available. The algorithm can prune a whole subtree of the Cartesian product every time a constraint is violated.

In this work we use a variant of the backtracking algorithm, called Forward Checking (FC) [14]. This algorithm has been shown to be very efficient for a wide variety of CSP settings. In addition, it is very simple to implement and, most importantly, the state that needs to be kept during evaluation has small size. The basic idea behind FC is that every time a variable is instantiated, the new value is checked for consistency with all available values of the domains of outstanding variables. Inconsistent values are immediately removed. The characteristic data structure used by FC is one array per variable domain, with length equal to the size of the domain. Each element of the array is the id of the variable that made the corresponding domain value inconsistent. For few variables and

small variable domains this collection of arrays will be very small in size (each element can be one byte or less). Since we make sure that before the CSP Solver is called the variable domains have been restricted as much as possible, FC is expected to be very robust for spatial and spatiotemporal CSPs, as demonstrated in [24]. An alternative way is to evaluate the CSP as a multiway spatiotemporal join of the variable domains using their binary constraints as join predicates. Nevertheless, secondary memory techniques for multiway joins [21] are not expected to perform better than CSP algorithms for main-memory problems of small domains.

## 4 Alternative Query Evaluation Techniques

In this Section, we propose some variants of the basic algorithm, trading memory requirements for computational performance.

### 4.1 Computing and Updating Tight Expiration Times

Algorithm 1 may compute very loose expiration times for newly arriving events, which affect negatively the memory requirements of the system. Consider again the example query of Figure 1b and the stream of Figure 2. When  $a_1$  arrives, Algorithm 1 sets its expiration time to 11, i.e.,  $a_1.t = 1$  plus the maximum  $t_{ub}$  of an outgoing edge ( $C_{1,3}.t$  in this example). Nevertheless observe that unless an event with property B arrives before time 6,  $a_1$  should be deleted, because  $C_{1,2}.t$  may never be satisfied. Thus a *tight* expiration time for  $a_1$  is 6. When an event of type B arrives at or before time-instant 6, which satisfies  $C_{1,2}$  with  $a_1$ , then the expiration time of  $a_1$  is *renewed*. Indeed, event  $b_2$  satisfies  $C_{1,2}$  with  $a_1$ , thus  $a_1$  remains in the system until time-instant 8 (the expiration time of  $b_2$ ). This simple example shows that we could minimize the memory requirements of CCQ evaluation at the expense of computing and maintaining the expiration times of active events.

Let  $e$  be a *new-coming* event and  $V_i$  a variable in a query  $\mathcal{G}$ , such that  $e \propto C_i$ . Assume that  $e$  is useful with respect to  $V_i$ , i.e., the graph  $\mathcal{G}'$  containing  $\mathcal{V}_i^-$  is solvable. For setting a tight expiration time for  $e$ , with respect to  $V_i$ , we separate the following two cases:

1.  $\mathcal{V}_i^- \cup V_i = \mathcal{G}.V$ . In this case,  $e$  triggers an alert, however, newly arriving events for variables  $V_j \in \mathcal{V}_i^+$  may keep triggering alerts for as long as the temporal constraint  $C_{i,j}.t$  is active (given that the events satisfy the spatial constraints as well). This is true, since for all other  $j$ , the constraints are already satisfied. Intuitively, this can keep happening for as long as the longest lived temporal constraint  $C_{i,j}.t$ . Thus  $e.X$  should be updated to  $\max\{e.X, e.t + \max_{V_j \in \mathcal{V}_i^+} (C_{i,j}.t_{ub})\}$ , exactly like in the original algorithm.
2.  $\mathcal{V}_i^- \cup V_i \neq \mathcal{G}.V$ . In this case, there are future variables to  $V_i$  not in  $\mathcal{V}_i^-$ . If there are many such variables, we should set the expiration time for  $e$  as the *minimum*  $C_{i,j}.t_{ub}$  of all future variables  $V_j$ .

Since  $e$  is independently important for all  $(V_i, \mathcal{G})$  pairs, its expiration time is eventually set as the maximum of all expiration times due to each  $V_i$ . Algorithm 2 summarizes the changes to Algorithm 1 for computing tight expiration times.



**Algorithm 2.** HandleEventTight( $e$ : event,  $Q$ : queries)

---

..... lines 1–11 of Algorithm 1 .....
**if**  $\mathcal{G}$  is satisfiable **then**    **alert** solution;     $e.X := \max\{e.X, e.t + \max_{V_j \in \mathcal{V}_i^+} (C_{i,j}.tub)\};$     **else**  $e.X := \max\{e.X, e.t + \min_{V_j \in \mathcal{V}_i^+} (C_{i,j}.tub)\};$ 

▷ case 2

..... the rest of Algorithm 1 .....

---

Defining tight expiration times requires their correct maintenance as new events arrive. In our example, recall that the expiration time 6 for  $a_1$  was updated to 8, after the arrival of  $b_2$ . Let  $e$  be a new event, handled by Algorithm 2. When solving graph  $\mathcal{G}'$ , for all variables  $V_j \in \mathcal{V}_i^-$  and for each consistent assignment  $V_j = e'$ , the expiration time of  $e'$  is updated to  $\max\{e'.X, e.X\}$ . In other words,  $e'$  should remain in the database at least as long as  $e$  is useful, if there are events future to  $e$  that may generate alerts with  $e$  and  $e'$ . Embedding expiration time updates in Algorithm 2 is straightforward.

## 4.2 Explicit Maintenance of Variable Domains

Algorithm 1 and its extension (Algorithm 2) employs the spatiotemporal index for each incoming event to define variable domains on the fly and then solve the CSP on the domains restricted by binary and unary constraints. Observe that in this way the unary constraint  $C_j$  of a given variable  $V_j$  may be validated on the same event  $e$  multiple times (i.e., the first time  $e$  arrives and every time it satisfies the binary spatiotemporal constraints with a newly arriving event). An alternative method is to apply  $C_j$  only once per event  $e$  and then store  $e$  explicitly in the domain of  $V_j$ , until  $e$  expires. In other words, the domains of past variables  $V_j$  are not computed by probing the spatiotemporal index, but are stored explicitly, making the index obsolete. The space requirements of this approach increase since events may be stored in multiple domains (and hence duplication may occur). Nevertheless, the benefit is that we do not need to maintain a spatiotemporal index, which has a high processing cost. Which approach is better depends on the relative performance of the spatiotemporal structure used and the average cost for CSP evaluation, which in turn depends on the types of registered queries and the data distribution in the stream.

## 5 System Prototype Evaluation

This section presents a system prototype evaluation that will illustrate the applicability of the proposed techniques using real datasets. The performance of a CCQ evaluation engine depends mainly on two major operations, populating the variable domains and solving the CSPs. Therefore, we compare multiple variants of the CCQ prototype to quantify the effects of different evaluation strategies.

### 5.1 Testbed and Methodology

We use off-the-shelf tools to implement our system. More specifically, an R-tree index from [11] and a CSP solver based on the FC algorithm [14]. The system can be downloaded from [1]. All experiments are run on an Intel(R) Xeon(TM) CPU 3.2GHz.

We use real datasets from the Tropical Atmosphere Ocean Project [26], where a large number of buoys have been deployed around the Pacific Ocean to collect oceanic and atmospheric data several times a day. An archive of the measurements of the past 25 years is available from [26]. For our purposes we used a total of 900,000 measurements, interpreted as a stream of data arriving at a central server for processing. These measurements include the location of the buoy at the time of the measurement, sea surface temperature, pressure, dynamic height, salinity, relative humidity, wind speed, and wind direction.

We generate synthetic queries by varying the number and type of variables, the temporal constraints, and the spatial predicates. We generate a large number of query workloads based on query verbosity. Verbosity is defined as the total number of query results (alerts) produced by the query over the total number of streaming events, and can be adjusted by appropriately tuning unary and binary constraints, making event selections tighter or looser. The dataset and the queries can be downloaded from [1].

To test various aspects of the system prototype we used two performance measures: (1) the maximum sustainable processing rate that can be achieved; and (2) the maximum memory utilization. The first measure is computed as the number of streaming events that can be processed per second, and the second as the peak number of events that need to be stored in main memory to achieve the corresponding processing rate. We measure the system's performance according to the following measures: (1) query verbosity; (2) scalability; (3) temporal constraint length, number of variables and number of constraints per query. We compare three CCQ evaluation methods: (1) Algorithm 1 (Loose); (2) Algorithm 2 (Tight); (3) The alternative proposed in Section 4.2 (NoIndex).

*Query Verbosity.* We measured the performance of our system as a function of query verbosity (i.e., number of alerts produced over the total number of events processed). We run all variants using five registered queries of known verbosity.

Results are shown in Figure 5. The trend of the graph shows that for all variants, performance deteriorates as verbosity increases since a growing number of events qualify for the variable domains of the query as more alerts are being produced, meaning that CSP evaluation becomes more expensive. Notice that the Loose variant (the only one that does not utilize tight expiration times) has somewhat worse performance than the rest of the techniques, attributed to the larger variable domain sizes. The NoIndex approach offers the highest processing rates since it does not have to maintain and query an index.

Figure 5b plots the memory utilization for each variant. The numbers on the graph correspond to the peak number of events that need to be stored as a percentage of the total number of events. This measure illustrates the pruning ability of our system with respect to a brute force approach that would retain all measurements. Clearly, all techniques save substantial amount of main memory, especially for higher query verbosity. The NoIndex approach has larger memory requirements due to event replication (see Section 4.2).

*Scalability.* Next, we evaluate system performance as a function of the number of registered queries. We keep the verbosity fixed to 1% and vary from 5 up to 100 registered queries. The NoIndex approach can sustain up to 15,000 events per second even for 100

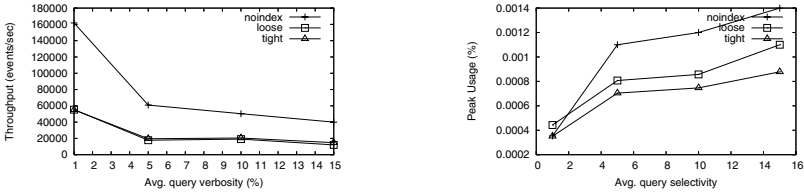


Fig. 5. Query Verbosity

registered queries. The other two approaches suffer as the number of queries increases, but have viable processing rates for most application scenarios even for up to 25 queries (we should stress the fact here that the algorithms produce *exact* query results). In terms of memory utilization, the NoIndex approach introduces substantial event duplication, making it a less favorable approach for tight memory constraints. Nevertheless, the memory requirements of all algorithms are still extremely small.

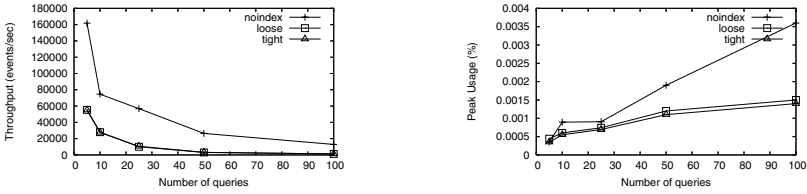


Fig. 6. Number of Queries

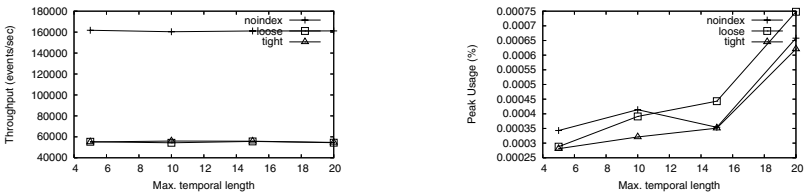


Fig. 7. Temporal Constraint Length

*Temporal Constraint Length.* Finally, we tested the system using queries with varying temporal constraint lengths. We use five registered queries and fix the verbosity to 1%. The larger the temporal extents the bigger the event expiration times, so it is expected that the memory requirements will increase as the temporal extents increase. On the other hand, throughput should remain unaffected since the verbosity is fixed. The results in Figure 7 attest to that observation. We conducted similar experiments for varying number of variables per query and number of constraints. The results were the same. Throughput is only affected by query verbosity and the total number of queries registered in the system.

To conclude, the NoIndex approach exhibits very good scalability in terms of registered queries and query verbosity, but higher memory requirements than the other approaches. The Loose and Tight algorithms can sustain a smaller number of queries at streaming rates due to the index lookups. All approaches prune a very large percentage of events, compared to the naive alternative.

## 6 Related Work

There is a lot of research on data streams in general but little work in the realm of spatiotemporal streams. Moreover, most work has concentrated on the special case of moving object applications. Finally, current work addresses only traditional spatial queries like range searches and nearest-neighbors. In contrast, here we deal with any type of spatiotemporal stream as well as complex queries with numerous problem variables and general spatiotemporal predicates between them (i.e., not only selections but also joins between events).

SINA [23] is a recently proposed framework for incremental evaluation of continuous range queries on data streams. It uses the shared execution paradigm to incrementally evaluate a large number of concurrent queries. SINA indexes the queries along with the data in order to be able to compute answers incrementally. Previous work with the same characteristics include [27][5][9]. In [27] the authors use incremental query evaluation, reversing the role of queries and data, and exploiting the relative locations of objects and queries to provide answers efficiently. They also propose an index method that exploits the maximum permissible velocity of the objects to delay expensive updating operations of the index. Finally, the authors of [5] and [9] assume that clients can process and store information, so that they can share query processing with the server in a distributed fashion. All these works are suited only for moving object applications and continuous range queries with absolute spatial coordinates that do not involve constraints in-between the objects. Similar work, concerning nearest neighbor queries, includes [17][29][31].

Related to our work is research concerning pattern mining with constraints in streaming databases. In [32] the authors address the issue of extracting frequent temporal patterns from the stream. They use a regression based algorithm to scan online transaction flows and generate candidate frequent patterns in real time. Similarly, a mining perspective is also adopted in [25], where the authors introduce techniques for answering queries with a wide range of constraints related to the length of the patterns, the items they contain, their duration, etc. However, these works do not consider transactions with spatial characteristics and concentrate on mining patterns that exceed a user specified threshold instead of identifying tuples that satisfy spatiotemporal or other constraints in real-time.

In [6] a system is proposed that can answer continuous queries over streaming data. The system registers a number of queries and a number of streams and applies new queries to old data, and old queries to new data on the streams. Nevertheless, this system does not consider spatial or temporal constraints between streaming data; it only considers predicates that resemble what we term unary variable constraints. Finally, it does not introduce the concept of expiration times to evict older data from main memory,

but rather indexes all incoming data according to user specified window sizes. Hammad et al. [12] studied continuous multiway join queries over data streams, where the join predicates are temporal. This problem can be viewed as a special case of the problem we study here. In addition, their work does not handle real-time alert triggering, since buffering techniques are employed before processing the registered queries. Thus, alerts may be triggered only with some delay.

## 7 Conclusion

We have presented a system prototype for evaluating *Continuous Constraint Queries* on spatiotemporal streams. The proposed system represents queries as Constraint Graphs that are incrementally evaluated as Constraint Satisfaction Problems every time a new event arrives on the stream. We introduce special algorithms for computing event expiration times in order to limit the number of events that need be maintained for providing exact answers. Finally, we present a concise experimental evaluation of a system prototype implementation. As future work we plan to extend the system for dynamic event properties (that change over time) and also investigate robust approximation policies for limiting main memory consumption even further.

## References

1. CCQ system prototype, <http://www.cs.ucr.edu/~marion/ccq>
2. AOML. Global Drifter Center, <http://www.aoml.noaa.gov/phod/dac/gdc.html>
3. Bessière, C.: J.C. Régin. Refining the basic constraint propagation algorithm. In: Proc. of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 309–315 (2001)
4. Bitner, J.R., Reingold, E.: Backtracking programming techniques. Communications of the ACM (CACM) 18(11), 651–656 (1975)
5. Cai, Y., Hua, K.A., Cao, G.: Processing range-monitoring queries on heterogeneous mobile objects. In: Proc. of the International Conference on Mobile Data Management (MDM), pp. 27–38 (2004)
6. Chandrasekaran, S., Franklin, M.J.: Streaming queries over streaming data. In: Proc. of Very Large Data Bases (VLDB) (2002)
7. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Journal of Artificial Intelligence 49(1-3), 61–95 (1991)
8. Gaschnig, J.: Experimental case studies of backtrack vs. waltz-type vs. new algorithms for satisficing assignment problems. In: Proc. of the Canadian Artificial Intelligence Conference, pp. 268–277 (1978)
9. Gedik, B., Liu, L.: MobiEyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In: Proc. of Extending Database Technology (EDBT), pp. 67–87 (2004)
10. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: Proc. of ACM Management of Data (SIGMOD), pp. 47–57. ACM Press, New York (1984)
11. Hadjieleftheriou, M., Hoel, E., Tsotras, V.J.: Sail: A library for efficient application integration of spatial indices. In: Proc. of Scientific and Statistical Database Management (SSDBM) (2004)

12. Hammad, M.A., Aref, W.G., Elmagarmid, A.K.: Stream window join: Tracking moving objects in sensor-network databases. In: Proc. of Scientific and Statistical Database Management (SSDBM), pp. 75–84 (2003)
13. Hanson, E., Carnes, C., Huang, L., Konyala, M., Noronha, L., Parthasarathy, S., Park, J., Vernon, A.: Scalable trigger processing. In: Proc. of International Conference on Data Engineering (ICDE), pp. 266–275 (1999)
14. Haralick, M., Elliot, J.: Increasing tree-search efficiency for constraint satisfaction problems. *Journal of Artificial Intelligence* 14(3), 263–313 (1980)
15. Keidl, M., Kreutz, A., Kemper, A., Kossmann, D.: A publish & subscribe architecture for distributed metadata management. In: Proc. of International Conference on Data Engineering (ICDE), pp. 309–320 (2002)
16. Kumar, V.: Algorithms for constraints satisfaction problems: A survey. *The AI Magazine* 13(1), 32–44 (1992)
17. Lazaridis, I., Porkaew, K., Mehrotra, S.: Dynamic queries over mobile objects. In: Proc. of Extending Database Technology (EDBT) (2002)
18. Lee, M.-L., Hsu, W., Jensen, C.S., Teo, K.L.: Supporting frequent updates in R-Trees: A bottom-up approach. In: Proc. of Very Large Data Bases (VLDB) (2003)
19. Papadimitriou, C., Grigni, M., Papadias, D.: Topological inference. In: Proc. of the International Joint Conference of Artificial Intelligence (IJCAI) (1995)
20. Madden, S., Shah, M., Hellerstein, J., Raman, V.: Continuously adaptive continuous queries over streams. In: Proc. of ACM Management of Data (SIGMOD), ACM Press, New York (2002)
21. Mamoulis, N., Papadias, D.: Multiway spatial joins. *ACM Transactions on Database Systems (TODS)* 26(4), 424–475 (2001)
22. Mamoulis, N., Yiu, M.L.: Non-contiguous sequence pattern queries. In: Proc. of Extending Database Technology (EDBT), pp. 783–800 (2004)
23. Mokbel, M.F., Xiong, X., Aref, W.G.: SINA: Scalable incremental processing of continuous queries in spatiotemporal databases. In: Proc. of ACM Management of Data (SIGMOD), ACM Press, New York (2004)
24. Papadias, D., Mamoulis, N., Delis, V.: Algorithms for querying by spatial structure. In: Proc. of Very Large Data Bases (VLDB), pp. 546–557 (1998)
25. Pei, J., Han, J., Wang, W.: Mining sequential patterns with constraints in large databases. In: Proc. of Conference on Information and Knowledge Management (CIKM) (2002)
26. PMEL. Tropical Atmosphere Ocean Project, <http://www.pmel.noaa.gov/tao>
27. Prabhakar, S., Xia, Y., Kalashnikov, D., Aref, W.G., Hambrusch, S.E.: Query indexing and velocity constraint indexing: Scalable techniques for continuous queries on moving objects. *IEEE Transactions on Computers* 51(10), 1–17 (2002)
28. Six, H., Widmayer, P.: Spatial searching in geometric databases. In: Proc. of International Conference on Data Engineering (ICDE), pp. 496–503 (1988)
29. Song, Z., Roussopoulos, N.: K-nearest neighbor search for moving query point. In: Proc. of Symposium on Advances in Spatial and Temporal Databases (SSTD), pp. 79–96 (2001)
30. Stonebraker, M., Sellis, T.K., Hanson, E.N.: An analysis of rule indexing implementations in data base systems. In: Expert Database Conference, pp. 465–476 (1986)
31. Tao, Y., Papadias, D., Shen, Q.: Continuous nearest neighbor search. In: Proc. of Very Large Data Bases (VLDB), pp. 287–298 (2002)
32. Teng, W.-G., Chen, M.-S., Yu, P.S.: A regression-based temporal pattern mining scheme for data streams. In: Proc. of Very Large Data Bases (VLDB) (2003)
33. Tsang, E.P.K.: *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego (1993)
34. Yan, T.W., Garcia-Molina, H.: The sift information dissemination system. In: *ACM Transactions on Database Systems (TODS)*, pp. 529–565. ACM Press, New York (1999)

# Collaborative Spatial Data Sharing Among Mobile Lightweight Devices

Zhiyong Huang<sup>1,3</sup>, Christian S. Jensen<sup>2</sup>, Hua Lu<sup>1,2</sup>, and Beng Chin Ooi<sup>1</sup>

<sup>1</sup> School of Computing, National University of Singapore, Singapore

<sup>2</sup> Department of Computer Science, Aalborg University, Denmark

<sup>3</sup> Institute for Infocomm Research, Singapore

**Abstract.** Mobile devices are increasingly being equipped with wireless peer-to-peer (P2P) networking interfaces, rendering the sharing of data among mobile devices feasible and beneficial. In comparison to the traditional client/server wireless channel, the P2P channels have considerably higher bandwidth. Motivated by these observations, we propose a collaborative spatial data sharing scheme that exploits the P2P capabilities of mobile devices. Using carefully maintained routing tables, this scheme enables mobile devices not only to use their local storage for query processing, but also to collaborate with nearby mobile peers to exploit their data. This scheme is capable of reducing the cost of the communication between mobile clients and the server as well as the query response time. The paper details the design of the data sharing scheme, including its routing table maintenance, query processing and update handling. An analytical cost model sensitive to user mobility is proposed to guide the storage content replacement and routing table maintenance. The results of extensive simulation studies based on an implementation of the scheme demonstrate that the scheme is efficient in processing location dependent queries and is robust to data updates.

## 1 Introduction

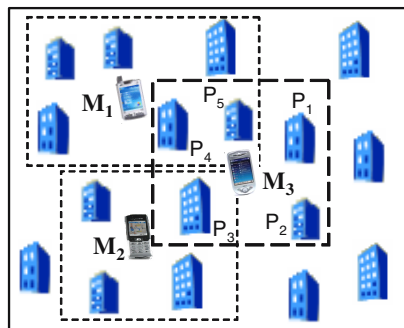
In step with the continued advances in computing electronics and wireless networking technologies, the mobile computing is gaining in prominence. In mobile computing, users equipped with portable devices such as mobile phones and PDAs, termed mobile clients, may issue local queries to learn about their geographic surroundings. For example, mobile services may enable tourists to learn about near-by attractions and may inform shoppers about near-by sales. Traditionally, the data provided by such services are stored in a central database. By means of a point-to-point wireless communication channel, the mobile clients may communicate with an application server that accesses the data and is responsible for the processing of queries.

To reduce the client/server (C/S) communication cost, techniques have been proposed that use client storage to cache results of previous queries and then use these data for answering new queries either fully or, more often, partially [3,11,19]. These techniques almost completely rely on the C/S architecture, with little or no direct communication and collaboration among the mobile devices.

This sole reliance on the C/S architecture fails to take advantage of the new wireless peer-to-peer (P2P) communication capabilities of modern mobile devices. The wireless

P2P channels have considerably more bandwidth than do traditional wireless C/S channels [16]. Moreover, as adjacent mobile devices are likely to issue queries whose results overlap, it is becoming possible, and potentially attractive, for mobile devices to share data in P2P fashion.

In Figure 1, for example, mobile devices  $M_1$  and  $M_2$  have issued queries and have already stored locally data that correspond to the rectangles they belong to. Then a third



**Fig. 1.** Use Peer Storage to Answer Query

device  $M_3$  issues a query for data corresponding to its rectangle, i.e., data corresponding to  $P_1$  to  $P_5$ . Only the data pertaining to  $P_1$  and  $P_2$  are in  $M_3$ 's local storage.

Using traditional techniques,  $M_3$  must access the remote wireless server to obtain data for  $P_3$ ,  $P_4$ , and  $P_5$ . In contrast, with P2P wireless communication  $M_3$  can obtain data for  $P_4$  and  $P_5$  from  $M_1$ , and data for  $P_3$  from  $M_2$ . This reduces the query response time significantly because the P2P bandwidth is much higher than the C/S bandwidth.

With the objective of exploiting the much faster wireless P2P channels, we propose a new collaborative data sharing scheme. An underlying grid-based structure is used for managing the data stored on the server and those portions of the data distributed among the mobile devices. The entire data space is partitioned by the grid, with each cell being a basic unit of data storage. The grid information is organized as a string, each bit of which indicates whether the corresponding grid cell contains any data or not. By broadcasting this space-saving string, the server is able to give clients global knowledge of its data, and to efficiently notify them of updates.

To facilitate the retrieval of peer data, each mobile device maintains a routing table. A routing entry captures which neighbor peer to contact for data pertaining to a specific grid cell, and how many hops to reach that peer. With routing tables, search among peers becomes directed, which contrasts to the blind flooding. This not only speeds up data search but also reduces communication messages. The routing table on each device is dynamically updated when its neighboring peers' storage contents change. Such changes are broadcast locally.

To answer a location-based spatial query with collaborative data sharing, a mobile device first checks its own storage, identifying those grid cells overlapping the query for which data is not available locally. Then it checks each cell of this kind in its routing



table. If a relevant routing entry is found, a data request is sent to the corresponding peer via the fast P2P channel. Only the data not obtained this way is subsequently requested from the server via the C/S channel. Upon receiving data as needed, a mobile device conducts a refinement to reach the exact query result, and places the data in its storage.

To best utilize the limited device storage, a probability-based predictive cost model is proposed for storage replacement and routing table maintenance. Taking into consideration the predicted movement of each device, this model gives priority to the data in grid cells that have the highest probabilities of being reused in the future. By retaining relevant data in device storage, this model reduces the communication between devices and the remote wireless server.

To efficiently handle data updates, the server notifies clients of updates using a compact data format. Upon receiving a notification, a client can easily identify and evict invalidated data.

This paper makes the following contributions: First, it recognizes the discrepancy between previous techniques and current mobile environments, and accordingly proposes a collaborative data sharing framework for location-based spatial queries in such environments. Second, it proposes a probability-based predictive cost model that guides both storage replacement and routing table maintenance. Third, it proposes query processing strategies for mobile devices within the framework. Fourth, it discusses how to accommodate data updates within the framework. Fifth, it conducts extensive experiments to confirm that the paper's proposals are efficient and robust.

One might advocate a wireless C/S architecture based on technologies such as an IEEE 802.11 network, as an alternative to our assumed setting. However, such alternative technologies are not widely deployed, but are limited to very short ranges. In contrast, cellular networks are widely available and have very large numbers of users; these thus provide a huge space for our assumed setting [16].

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 presents the system framework. Section 4 details the collaborative data sharing scheme and the cost model. Section 5 describes the relevant query and update processing techniques. Section 6 reports the experimental study results. Finally, Section 7 concludes.

## 2 Related Work

To reduce the communication between client and server via low-band wireless channels and shorten the query response time, caching techniques have been proposed for mobile environments [3]. Most such techniques are based on semantic caching [9], where semantic descriptions of previous queries as well as their corresponding results are cached. Subsequent queries are fully or partially answered using the cached results by matching the semantic descriptions. Semantic-based clustering [19] and Voronoi diagrams [24] are employed to organize semantic contents cached on mobile devices. However, these semantic caching schemes only support homogeneous queries. To remedy this, Hu et al. [11] propose a proactive technique that stores on the mobile clients both query results and the R-tree index nodes accessed during query processing. This technique may result in substantial device-side space use and processing costs. Later,

Lee et al. [14] propose to cache both concrete data objects of interest and complementary regions at a coarse granularity. Inter-device collaboration is not addressed in any of the above works. Deviating from the previously dominant C/S architecture, Liu et al. [15] propose to cluster mobile devices via wide-band wireless links. In each cluster, one mobile device acts as a gateway that is connected to the Internet via a narrow-band link and forwards queries/results from/to its cluster. This work does not address device-side cache organization, and data are not passed directly among mobile devices, but via the gateways.

Assuming a wireless broadcast environment, Hara [10] proposes several caching strategies that enable clients to cooperate on selecting broadcast items to cache. By utilizing multi-hop routing in an ad hoc network, Yin and Cao [23] propose that a device caches either the requested data or the path needed to get the data when it forwards a query result. Our work differs from these works. First, we consider a hybrid system architecture (see Section 3.1 for details), instead of assuming a traditional wireless C/S environment or a pure ad hoc network. Second, we are specifically interested in location dependent queries that require spatial data. Third, we take advantage of a client's movements for the cache management on devices by using a specific model.

Ku et al. [13] propose a technique for supporting location-based  $k$ NN queries in mobile environments that utilizes objects cached by peers. The core of the technique is specific to  $k$ NN queries and is not applicable to other query types including range query. Recently, Chow et al. [8] consider cooperative caching in a mobile setting that is similar to ours. Their focus is on the grouping of mobile devices with similar mobility patterns in order to improve cache performance.

Next, data management in mobile ad hoc networking (MANETs) [4] or mobile P2P environments has attracted significant research interest. Kortuem et al. [12] discuss scenarios where encountering mobile devices may exchange information, together with challenges faced by mobile ad hoc information systems. At a conceptual level, Xu et al. [22] discuss extensive data management topics in mobile P2P networks, including data modeling and data dissemination specific to mobile peers. Within a hierarchical mobile P2P environment with wireless cells at the bottom and fixed networks at the top, Budiarto et al. [7] discuss mobile data replication strategies in the fixed network. To avoid message flooding and to improve search hit rates, Lindemann et al. [17] propose a distributed document search service that helps access results cached in peer devices by locally broadcasting queries and response messages.

## 3 System Framework

### 3.1 System Architecture

In a traditional, mobile C/S environment, one or more wireless application servers store data and process queries from mobile clients within their corresponding coverage. This setting is being shifted as mobile devices are increasingly being equipped with wireless ad-hoc networking capabilities such as infrared, Bluetooth, or even Wi-Fi. The ad-hoc networking channel usually has much higher bandwidth (1-11Mbps for IEEE 802.11b, and up to 54Mbps for IEEE 802.11a and 802.11g) than the traditional C/S channel (38.6 Kbps to 2.4 Mbps) [16]. This makes it possible and potentially attractive for the

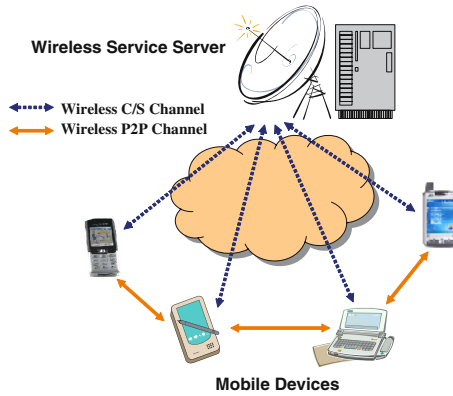


Fig. 2. Hybrid Mobile System Architecture

mobile devices to share their local data with their peers. The architecture with both C/S and P2P communication, as shown in Figure 2, we call a *hybrid* architecture. In this paper, we limit our problem to the extent of a single wireless application server's coverage.

For this as well as the traditional C/S architecture, a key objective is to reduce the communication between client and server, as the bandwidth of the wireless C/S channel is low. In the traditional C/S architecture, local storage is the only resource that a mobile device can attempt to exploit. In the hybrid architecture, the high bandwidth channels among mobile devices are an important resource that can be utilized. By allowing mobile devices to share their data via the P2P wireless channels, the communication between a mobile device and the server is expected to be reduced. We proceed to discuss how to achieve this reduction by using an appropriate data sharing scheme.

### 3.2 Collaborative Data Indexing Requirements

The spatial data on the server are usually indexed by some spatial indexes in order to facilitate efficient spatial-query processing. For the server alone, which has ample storage space and computing power, different spatial indexes are applicable, and their performance differences are not expected to be significant in comparison to as the considerable wireless communication cost.

The situation becomes complex when we intend to store and reuse data from the server on resource-constrained mobile devices. If a spatial query cannot be fully answered by reusing data stored locally, its unanswered portion will be sent to other mobile peers or to the server. This query portion can be processed in a direct way, without any transformation or adjustment, if the organizations of data on the mobile devices and on the server share some basic characteristics. Therefore, it will be beneficial if the index used on server also can be used on the mobile devices.

Since mobile devices usually have limited storage space and computing power, we must be careful to choose an appropriate spatial index that can be shared among the devices and the server. The limited storage makes a spatial index with little storage

overhead attractive. The limited computing power renders it important that the operations to be performed on the spatial index are simple yet efficient.

### 3.3 Collaborative Indexing

Because the R-tree and its variants involve comparatively complex operations and consume extra space for internal nodes, we do not use them on the resource-constrained mobile devices. Rather, we use the simple yet effective grid file [18] as the spatial index in our system. A grid file allows economically storing on each mobile device a summary of the data indexed on the sever, enabling the devices to maintain knowledge of the server data. The entire data space is partitioned by a  $H$  (rows) by  $W$  (columns) uniform grid, yielding a total of  $H \cdot W$  grid cells. The server holds a grid directory, which contains the extent (*left*, *right*, *top*, *bottom*) of the data space and a linear array representing all grid cells in row major order. Each cell has a pointer to the disk page storing those data points.

The summary of the server data indexed by a grid is organized in a compact format and broadcast to all mobile devices within the server's coverage. For each grid cell, one bit is used to indicate if it contains any data points: 0 for an empty cell and 1 otherwise. This way, a total of  $\lceil (H \cdot W)/8 \rceil$  bytes are needed to represent all cells in row major order starting from the top left. In addition, the values of  $H$ ,  $W$ , and the extent coordinates are necessary for a mobile device to perform basic grid indexing operations.

For  $H$  and  $W$ , we use a byte to represent either of them and thus can accommodate up to 64K grid cells, which usually is enough. For the extent coordinates we use floats, which need 4 bytes each. Therefore, every mobile device needs  $\lceil (H \cdot W)/8 \rceil + 18$  bytes to hold the global grid summary. That information is organized into a byte string called a *grid string*, which sequentially contains *left*, *right*, *top*, *bottom*,  $H$ ,  $W$  and bytes for all grid cells. The ending bits in the last byte are not used if the number of grid cells is fewer than 8.

An example is shown in Figure 3. In the upper part of this example, the region of interest is partitioned into 2 rows and 4 column, i.e., 8 cells. In the lower part, the grid string is shown with 19 bytes totally. The bits in the last byte are used to indicate the validity of each grid cell, starting from the top-left one sequentially in row major order.

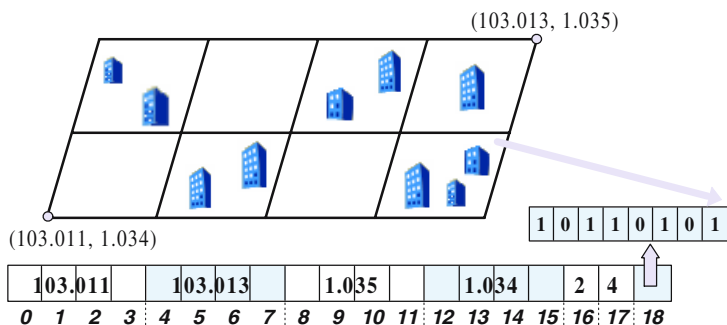


Fig. 3. Example Grid String

## 4 Collaborative Data Sharing Scheme

### 4.1 Storage Scheme for Mobile Devices

Every mobile device receives the grid string from the data server when it enters the coverage area of the server. This string is kept locally and used when processing spatial queries. For a given spatial query, the values of  $H$ ,  $W$ , and the extent coordinates are used to determine which grid cells the query concerns. Then, the relevant bits in the grid string are used to identify the cells that actually contain data points. Only those cells are considered subsequently.

Without loss of generality, we suppose that every point has  $n$  attributes, in addition to the spatial coordinates of float type, and that these attributes occupy  $A_n$  bytes in total. Out of its total storage space, a mobile device  $M_i$  is allowed to use  $S_i$  bytes for data points. This portion is divided into  $N_i$  equal-sized contiguous slots, each of which can contain  $s_i$  points. In other words,  $S_i = N_i \cdot s_i \cdot (8 + A_n)$ .

Two data structures are used for this storage portion. The first, *StoredCells*, is a list of records of the format  $\langle index, count, slot\_IDs \rangle$ , where *index* is the index of a cell stored, *count* is the count of points in that cell, and *slot\_IDs* holds the IDs of the slots that contain the cell's points. To facilitate search for grid cells, *StoredCells* is maintained in ascending order of *index*. The second, *FreeSlots*, is a list of the IDs of unused storage slots. Initially *FreeSlots* contains all storage slots and *StoredCells* is empty.

As the grid parameters  $H$  and  $W$  each occupy a byte, a 2-byte short integer is enough for *index*. Assume a grid cell contains at most  $G_m$  points. Then  $\lceil G_m/256 \rceil$  bytes are needed for *count*. A slot ID needs  $\lceil N_i/256 \rceil$  bytes, while a grid cell needs at most  $\lceil G_m/s_i \rceil$  storage slots. As a result, a stored cell needs at most  $2 + \lceil G_m/256 \rceil + \lceil (G_m \cdot N_i)/(256 \cdot s_i) \rceil$  bytes.

When a new grid cell is to be stored locally, the first step is to determine how many slots are needed. Then we need to search the *FreeSlots* to see if enough free slots are available. If so, free slots are selected to store the cell. *FreeSlots* is modified, and new cell information is created in *StoredCells* accordingly. If not enough free slots exist, some stored grid cells must be evicted to make room for the new one. We discuss this issue in Section 4.3. Next, we address how to share storage contents among mobile devices.

### 4.2 Sharing Data Via Routing Table

A naive approach for a mobile device to retrieve data from peers is to send a message to all its neighbors, each of which then either returns the requested data if it has, or forwards the request to its own neighbors. This approach leads to blind flooding rather and incurs a large number of messages, which may yield long response time.

To facilitate the retrieval of data from peers, we maintain a routing table on each mobile device. A routing table entry on a device  $M_i$  is in the format of  $\langle cell\_id, nb, hops \rangle$ . It tells that a neighbor peer  $nb$  of  $M_i$  has data for cell  $cell\_id$  in its local storage or routing table. And  $hops$  is the count of hops from  $M_i$  to the peer, via intermediate peers, that actually stores the data for the cell.

When  $M_i$  stores a new cell or evicts a cell, it notifies its neighbors by sending out corresponding messages. Upon receiving a message from  $M_i$  invoked by storing  $cell_j$ ,

a mobile device  $M_k$  takes one of the following three actions: (1) ignores it because  $cell_j$  is in local storage; (2) creates a new routing entry for it because  $cell_j$  is in neither its storage nor routing table; (3) compares the length of the new route with the that for  $cell_j$  currently in the routing table, changing the current one if the new route involves fewer hops. Device  $M_k$  can also forward the new route to its own neighbors, if (2) happens. Similar actions are to be taken by subsequent neighbors. Upon receiving a message from  $M_i$  invoked by evicting  $cell_j$ , a mobile device  $M_k$  removes the relevant entry from its routing table if it exists. If a removal occurs,  $M_k$  sends similar messages to its own neighbors who will take similar actions.

A maximum hops value is used as a system parameter to further limit the forwarding of maintenance messages in the last two situations described above. A message for either a new route or for an eviction will not be forwarded any further when it has traveled the maximum number of hops.

When a mobile device  $M_i$  detects a new neighbor  $M_k$  through wireless signaling,  $M_i$  organizes the contents of both its storage and routing table into routing entries and sends them to  $M_k$ . Device  $M_k$  then updates its own routing table by checking the incoming entries against its storage and routing table.

### 4.3 Management of Limited Device Storage

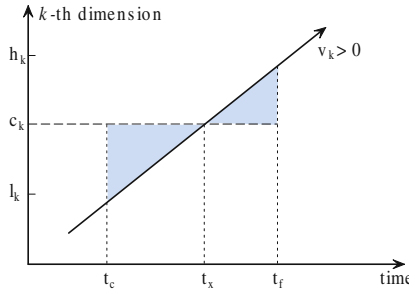
For storage constrained devices, we need to replace both storage contents and routing table entries when necessary. We propose a cost model to guide the replacement.

**Cost Model.** In our storage scheme, two factors need to be considered when choosing one or more victims for replacement. One is that after selecting victim(s), there should be enough free space for the new grid cell. For a stored grid cell  $C_j$ , the count of data points it covers can be found in the *StoredList*, i.e., *count*. The other is that we should attempt to avoid eliminating grid cells that will be accessed in the (near) future. This can be achieved by taking into account a mobile device's predicted movement.

We use  $prob(M_i, C_j, \Delta t)$  to denote the probability that a device  $M_i$  will access cell  $C_j$  in the time period  $[t_c, t_c + \Delta t]$ , where  $t_c$  is the current time. We thus look  $\Delta t$  time units into the future and predict how likely it is that  $M_i$  will need  $C_j$  in that time period. In Section 5.4, we discuss how to choose an appropriate value for  $\Delta t$ .

Probability *prob* also depends on the movement of  $M_i$  and the queries it will issue. Although it is difficult to predict the query pattern of a mobile device, we expect a significant spatial locality for the queries issued by a mobile device. Therefore, the distance between a device  $M_i$  and a grid cell  $C_j$  is of importance. At any single time point, we consider the Manhattan distance between  $M_i$ 's current position and the center of  $C_j$ . Following Tao et al. [21] and Brilingaité and Jensen [5], we assume that each device is aware of its own motion pattern. In contrast to Dar et al. [9], we use an integral to represent the distance between  $M_i$  and  $C_j$  for the period  $[t_c, t_c + \Delta t]$ , as shown in Formula 1. An integral along time has been proven to be effective in dealing with changes covering a future time period [20].

$$iDist_M(M_i, C_j, \Delta t) = \int_{t_c}^{t_c + \Delta t} dist_M(M_i.pos(t), C_j) dt \quad (1)$$



**Fig. 4.** Integral of Distance over Time

Suppose in an  $n$ -dimensional space,  $c_k$  is the middle point of a grid cell on the  $k$ -th dimension, and  $p_k(t)$  is the time-parameterized position of a mobile device on that dimension. Formula 1 can be developed as follows.

$$\int_{t_c}^{t_c+\Delta t} \sum_{k=1}^n |p_k(t) - c_k| dt = \sum_{k=1}^n \int_{t_c}^{t_c+\Delta t} |p_k(t) - c_k| dt \tag{2}$$

This indicates that we can compute the integral on each individual dimension and then sum up all those integrals to obtain the desired distance. For example, in Figure 4 the mobile device has a positive linear velocity (it moves upwards) on the  $k$ th dimension, where the cell  $C_j$ 's range is  $[l_k, h_k]$ . And  $t_f$  is assumed to be a future time point far enough from  $t_c$ , while  $t_x$  is the moment  $p_k(t)$  passes  $c_k$ . Then the integral on that dimension can be expressed as a sum of two parts:

$$\int_{t_c}^{t_x} (c_k - p_k(t)) dt + \int_{t_x}^{t_f} (p_k(t) - c_k) dt = \frac{1}{2} \cdot v_k \cdot (t_x - t_c)^2 + \frac{1}{2} \cdot v_k \cdot (t_f - t_x)^2 \tag{3}$$

Taking into account negative speeds, the appropriate integral value is:

$$\frac{1}{2} \cdot |v_k| \cdot ((t_x - t_c)^2 + (t_f - t_x)^2) \tag{4}$$

Depending on the concrete values of  $t_c$  and  $\Delta t$ , integrating on an individual dimension can involve one or two integral parts with corresponding ranges. Non-linear movements may involve additional integral parts because the mobile device may pass the middle point for more than once.

Intuitively, the closer a mobile device  $M_i$  gets to a grid cell  $C_j$ , the higher the probability that  $M_i$  will be interested in  $C_j$  is. This means  $prob$  is inversely proportional to  $iDist_M$  for the period of consideration. Therefore, we give priority to those cells with large  $iDist_M$  values when choosing victims. These are the least likely to be reused in the future because they are relatively far away from the mobile device  $M_i$ . When a victim is chosen, the storage slots it occupies is released and recorded in the *FreeSlots* list. The selection of a victim is repeated until *FreeSlots* has enough free slots for the new cell.

**Routing Table Size Control.** Based on its own resource availability, a mobile device can determine how much local storage to use for its routing table. To avoid a possible overflow caused by routing table size growth, a mobile device can choose to ignore new route entries coming from peers, or remove existing ones from its own routing table. The decision can be made based on the proposed cost model, by computing the access possibilities of those grid cells in a routing table.

## 5 Query and Update Processing

Our scheme supports heterogeneous queries. We consider the two arguably most popular query types: range queries and  $k$ NN queries. Overall, query processing proceeds as follows. After a location-based spatial query  $Q$  is issued on a mobile device  $M_{org}$ , termed *originator* of  $Q$ , the local storage contents are used to answer the query. If the query cannot be answered fully this way, the originator asks its neighbors for possible data with the help of its routing table. For those data portions that are unavailable in the routing table, requests are sent to the server. For a range query, a local refinement step is required to handle those grid cells that the query only covers partially. For a  $k$ NN query, the search bound is adjusted dynamically during the search to reduce the number of grid cells involved.

### 5.1 Range Queries

A range query  $Q_r$  issued on mobile device  $M_{org}$  is represented by  $\langle pos, d \rangle$ , where  $pos$  is  $M_{org}$ 's current position and the range is the circle centered at  $pos$  with radius of  $d$ .

When a range query  $Q_r$  is issued, the grid string is first used to identify the non-empty grid cells that intersect the range specified in the query. We use  $C(Q_r)$  to represent the set of indexes of all these non-empty grid cells. Then, the local storage is checked to see if any of those involved cells are available locally. We use  $C_l(Q_r)$  to represent the set of indexes of cells stored locally. If all cells are in local storage, i.e.,  $C_l(Q_r) = C(Q_r)$ , the query is answered locally in full. Otherwise, the unavailable cells are retrieved from elsewhere. We let  $C_p(Q_r)$  represent the set of indexes of grid cells that appear in  $M_{org}$ 's routing table, and  $C_u(Q_r)$  represents the remaining grid cell indexes.

For the cells in  $C_p(Q_r)$ , requests are sent to peers according to the routing entries. Each such request is forwarded along a routing path until the peer holding the data is reached. This peer then sends the data to  $M_{org}$ . Due to the dynamic nature of wireless mobile ad hoc networking, it is possible that a mobile device along the routing path receives a request, but fails to find the relevant routing entry or grid cell data locally. Or, a mobile device may get a failure message from the lower protocol level when contacting a peer. When a device faces such situations, it returns a routing failure message to  $M_{org}$  along the reversed path. Each mobile device on the path back removes the relevant routing entry from its own routing table, till  $M_{org}$  places that cell in  $C_u(Q_r)$ . Finally, a request for the cells in  $C_u(Q_r)$  is sent to the server, which in turn sends back the data to  $M_{org}$ .



**Algorithm** kNNSearch( $pos$ )**Input:**  $pos$  is the query originator's current position**Output:**  $k$  nearest neighbors

1.  $d_{bnd} = max$ ;
2. decide the grid cell  $C_{org}$  within which  $pos$  lies;
3. **if** ( $C_{org}$  is not an empty cell)
4.     **if** ( $C_{org}$  in storage)
5.         search  $C_{org}$  and adjust  $d_{bnd}$ ;
6.     **else**
7.         **if** ( $C_{org}$  in routing table)
8.             send request for  $C_{org}$  to peer;
9.         **else**
10.             send request for  $C_{org}$  to server;
11.             search  $C_{org}$  and adjust  $d_{bnd}$  upon receiving;
12. **while** (TRUE)
13.     decide the next  $cells_{srd}$  w.r.t  $pos$  and  $d_{bnd}$ ;
14.     **if** ( $cells_{srd} == \emptyset$ ) **break**;
15.     **for each** cell  $cell_i$  in  $cells_{srd}$
16.         **if** ( $cell_i$  in storage)
17.             search  $cell_i$  and adjust  $d_{bnd}$ ;
18.             remove  $cell_i$  from  $cells_{srd}$ ;
19.     **if** ( $cells_{srd} \neq \emptyset$ )
20.          $cells_{rt} =$  cells in  $cells_{srd}$  and routing table;
21.         send requests for cells in  $cells_{rt}$  to peers;
22.         send request for cells in  $cells_{srd} \setminus cells_{rt}$  to server;
23.         search  $cells_{srd}$  and adjust  $d_{bnd}$  upon receipt;

**Fig. 5.**  $k$ NN Search Framework

## 5.2 $k$ NN Queries

A  $k$ NN query  $Q_k$  issued on mobile device  $M_{org}$  is represented by  $\langle pos, k \rangle$ , where  $pos$  is  $M_i$ 's current location and  $k$  is the number of nearest neighbors required.

Processing a  $k$ NN query  $Q_k$  is relatively complicated compared to a range query, as we cannot directly determine  $C(Q_k)$ , the set of grid cells intersected by  $Q_k$ . We conduct the  $k$ NN search by starting from the cell  $C_{org}$  where  $M_{org}$  is, then spiral through all surrounding cells from inner to outer. A search bound  $d_{bnd}$  is maintained during the procedure, which is the distance between the  $pos$  and the  $k$ -th nearest neighbor or  $max$  if less than  $k$  neighbors have been found thus far. At each step, we first decide the set of cells  $cells_{srd}$  on a surrounding circle that need to be searched. Two kinds of cells are excluded from  $cells_{srd}$ : empty cells indicated by the grid string and those cells outside the search bound. For  $cells_{srd}$ , we first search the cells in local storage; then, for those ones not stored, we send requests to peers if they appear in the routing table, or otherwise to the server. The cells are searched as they are received. The loop terminates when we obtain nothing for the next  $cells_{srd}$ . The framework of  $k$ NN search on the query originator side is shown in Figure 5.

### 5.3 Updates

Updates to the data on the server can affect the grid cells in three different ways. First, an empty cell may become non-empty due to one or more data points being inserted. Second, a non-empty grid cell may become empty because all of its points are deleted. Third, the number of data points in a non-empty cell increases or decreases but remains non-zero, or point attributes change. This is the most likely scenario of the three in real life.

We use a two-tuple  $\langle idx, flag \rangle$  to represent an update, where  $idx$  refers to the grid cell of the object being updated, and  $flag$  indicates which of the above three types of updates it is (numbered I, II, and III, respectively). The server (or its administrator) is responsible for modify the server-side data and index when an update happens. After that, the server notifies the clients of the update by simply broadcasting the two-tuples to them.

Each client  $M_i$  processes an incoming update as detailed in Figure 6. If  $M_i$  has an ongoing query  $q$  whose result so far is invalidated by the update, the query  $q$  is discontinued. If the update is of type I or II,  $M_i$  needs to invert the corresponding bit in the grid string. If the grid cell  $C_{idx}$  involved in the update resides in storage,  $M_i$  evicts it from the storage. Otherwise, if the cell  $C_{idx}$  has a routing entry in the routing table, it is removed from the table.

---

#### Algorithm $update(idx, flag)$

**Input:**  $idx$  is the index of the grid cell updated  
 $flag$  is the update type

1. **if** (query  $q$  is ongoing **and**  $q$ 's result so far covers  $C_{idx}$ )
  2.     abandon query  $q$ ;
  3. **if** ( $flag$  is I or II)
  4.     invert  $C_{idx}$ 's bit in the local grid string;
  5. **if** ( $C_{idx}$  in storage)
  6.     evict  $C_{idx}$  from storage;
  7. **else if** ( $C_{idx}$  in routing table)
  8.     remove  $C_{idx}$ 's entry from routing table;
- 

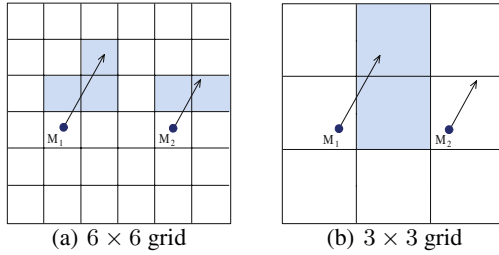
**Fig. 6.** Update Processing on A Device

Our proposal is able to efficiently handle updates because the compact yet informative collaborative indexing scheme works successfully between the server and the clients.

In the experimental evaluation (see Section 6), we assume that updates happen at random across both space and time. We vary the *update ratio*, the ratio of the number of point updates during the experiment period to the total number of points used in the experiment, to see its impact on the performance of our proposal.

### 5.4 Effects of Grid Configuration and $\Delta t$

As a grid cell is the basic unit in our storage and sharing scheme, its size has important impact on system performance. If a cell is too large, which indicates it probably contains



**Fig. 7.** Effects of Grid Cell Size and  $\Delta t$

more data points, it will require too much storage space while actually most of data points within may not be used by the mobile device storing it. In contrast, if a cell is too small, new requests may become frequent, and query performance will deteriorate. The cell size also affects the probability estimation in Section 4.3 because the Manhattan distance in Formula 1 is relevant to the size of cell  $C_j$ .

See the example in Figure 7. The whole region of interest is partitioned using two different grids, a  $6 \times 6$  grid and a  $3 \times 3$  grid. Assume the existence of two devices  $M_1$  and  $M_2$ , whose current positions are represented as dots. The vector attached to each device indicates its movement, and its length indicates how far the device moves during  $\Delta t$ . Thus,  $M_1$  moves faster than  $M_2$  here. In the  $6 \times 6$  grid,  $M_1$  will need 3 cells within  $\Delta t$ , all of which are shaded in the figure. Though in the  $3 \times 3$  grid,  $M_1$  will only need two cells, the number of data points to be stored is considerably increased unless many empty cells are involved. This contrasts the situation of  $M_2$  who does not need to store new cells in the  $3 \times 3$  grid, at the cost of storing a large cell already.

Next, parameter  $\Delta t$  determines how far we will look into the future when estimating the probabilities for a cell to be reused. It also affects how many grid cells will be involved. Refer to Figure 7(a) and let  $M_1$  and  $M_2$  have the same velocity, but  $M_1$  have a larger  $\Delta t$  than  $M_2$  (note now a vector length indicates the  $\Delta t$  value). Consequently,  $M_1$  needs to consider three cells while  $M_2$  can do with two. Because different mobile devices can have different resources, computing capacities, and even movements, each mobile device should hold its own  $\Delta t$  when it computes the probabilities during storage replacement. Furthermore, a mobile device can use different  $\Delta t$ 's to estimate probabilities at different times.

## 6 Experimental Evaluation

### 6.1 Experimental Settings

We implement our proposals using JiST-SWANS [1], a Java-based MANET simulator. We use a dataset named NE [2] of 123,593 points in float that represent metropolitan area postal addresses. We transform the dataset into the data space of  $[1000 \times 1000]$ . For each point we generate at random four attribute values in integer. We consider four main performance aspects: (1) the overall response time; (2) the local/peer storage hit rate; (3) the local/peer storage use rate; (4) the number of messages used to forward

routes/queries between mobile devices. We investigate how these aspects are affected by different storage sizes, grid configurations, mobile network scales, update ratios, and  $\Delta t$  settings. The simulation experiments are conducted on an IBM x255 server running Linux with four Intel Xeon MP 3.0GHz/400MHz processors and 18G DDR main memory.

Table 1 lists the parameters used in the simulation. The settings in bold are the defaults, used when their corresponding parameters are not varied. Initially, all data are stored in the simulated server, and no device stores any data in local storage. For a two-hour simulation period, every mobile device issues 10 to 100 queries whose type, range or  $k$ NN, are determined randomly. For a range query, the ratio of its radius to a grid cell's side length is randomly picked among 0.1, 0.2, ..., 1. For a  $k$ NN query,  $k$  is chosen at random from 1 to 5. We vary the number of mobile devices from 50 to 100, which yields a moderate-scale MANET [4]. All devices move within the spatial domain according to the random waypoint mobility model [6]. We set the maximum hops to forward a routing message to 3. This value, which was chosen experimentally, yields good cache effects, but does not incur high additional costs.

**Table 1.** Parameters Used in Simulation

Parameter	Setting
Grid configuration	<b>50×50</b> , 60×60, ..., 100×100
Number of mobile devices	<b>50</b> , 60, ..., 100
Storage slot size	32 (data points)
Storage slot count	<b>50</b> , 60, ..., 100
$\Delta t$	50s, <b>100s</b> , ..., 300s
Data update ratio (%)	<b>0</b> , 10, ..., 60
Speed range	0.1unit/s–1unit/s
Max hops to forward routes	3
Holding time	60s
Wireless routing protocol	AODV

## 6.2 Response Time

The response time is defined as the elapsed simulation time from the moment that a query is issued at a mobile device  $M_{org}$  to the moment that  $M_{org}$  gets all answers. In the simulation, we set the mobile P2P channel bandwidth to 11Mbps (IEEE 802.11b), and the wireless C/S channel bandwidth to 384Kbps, which is what 3G wireless networks are expected to offer for mobility at pedestrian speed [4]. We compare three different strategies: no local storage, local storage only, and collaborative sharing. The average simulation results are shown in Figure 8. If no device storage is used at all, the response time is the longest. We also see that collaborative data sharing shortens the response significantly. This is because collaborative data sharing uses the faster wireless P2P channels rather than the slow wireless C/S channel.

Figure 8(a) shows that an increase in storage space favors collaborative sharing over the local storage strategy. This is because the extra space retains more data requested

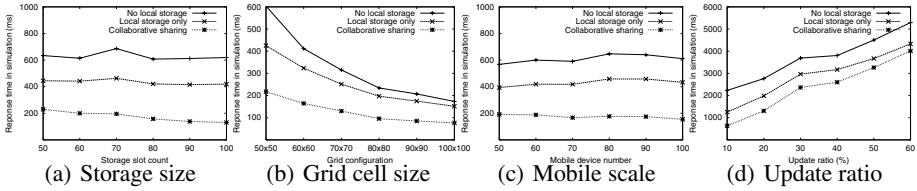


Fig. 8. Response Time in MANET Simulation

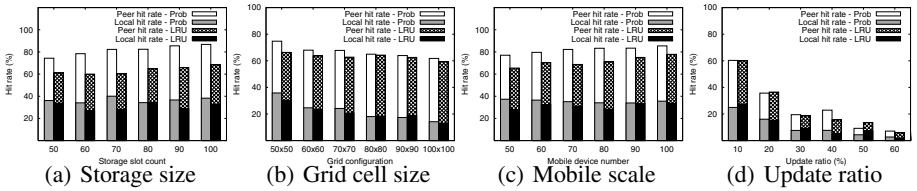


Fig. 9. Storage Hit Rate in MANET Simulation

by peers via collaborative sharing. All strategies benefit from small grid cells, as shown in Figure 8(b), because in our cell-based scheme, a smaller cell contains less data and needs less time for transmission between devices or between devices and the server. As the number of devices increases, the response time of the collaborative sharing strategy decreases slightly, as shown in Figure 8(c). Increased mobility provides a higher collaborative sharing capacity, which, however, is countered by additional costs, including query and result forwarding via multiple hops. Compared to the aforementioned results, all strategies degrade in the presence of updates, as shown in Figure 8(d). This is so because updates may delay, if not invalidate, ongoing queries and evict data in local storage. Nevertheless our collaborative sharing strategy still performs the best, since inter-device sharing remains effective.

### 6.3 Storage Hit Ratios

A storage hit occurs when a desired grid cell is found without it having to be retrieved from the server. We distinguish between two types of storage hits: hits in the local storage of a device and hits in the storage of peers. For each device, we use the *local hit ratio* to represent the percentage of requested grid cells found in its local storage, and we use the *peer hit ratio* for the percentage found in peer storage. In the simulation, we compare our probability-based storage replacement policy with the traditional LRU policy. Our policy outperforms the LRU policy for almost all settings used in the experiments, as shown by the results reported in Figure 9. Our proposal predicts the near-future movement of a device and uses this for computing probabilities when it makes replacement decisions. In contrast, the LRU policy treats all grid cells from a static point of view.

Referring to Figure 9(a), it is as expected that increased storage yields a higher hit ratio, as more data can be stored on the devices. Figure 9(b) shows that a coarser grid

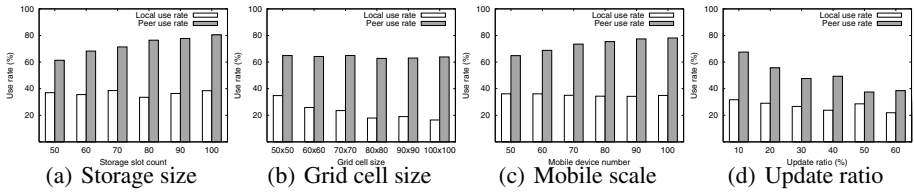


Fig. 10. Storage Use Rate in MANET Simulation

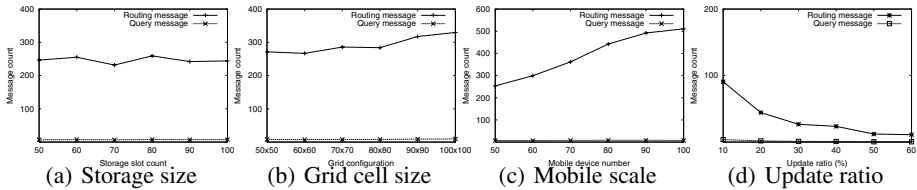


Fig. 11. Message Count in MANET Simulation

incurs a higher local hit ratio, but a lower peer hit ratio. Larger grid cells means that a bigger spatial region is stored on a device; hence, the device's future queries can find more data points locally, as queries on the same device exhibit locality. On the other hand, as adjacent devices are more likely to issue overlapping spatial queries than identical ones, they do not benefit from the larger grid cells and bigger spatial regions stored on the peers.

Nevertheless, larger grid cells imply the transfer of more data and may lead to longer response times, which is shown in Figure 8(b). According to the experiment covered by Figure 9(c), the peer hit ratio roughly grows proportionally with the number of device—with more devices, there is more storage for collaborative sharing, and hence more data can be obtained from peers instead of from the server. Referring to Figure 9(d), when updates are allowed both policies achieve lower hit ratios, but our proposal remains best for almost all cases. Updates tend to invalidate data in local storage, thus reducing the hit ratios. Our collaborative data sharing scheme is able to offset this effect to some degree.

We also consider the effect on the storage hit ratio by parameter  $\Delta t$  used in the probability-based replacement. The experimental results reported in Figure 12(a) indicate that  $\Delta t$  should be neither too short nor too long to ensure a high storage hit ratio.

#### 6.4 Storage Use Ratios

It is also of interest to know how much of a device's data is actually used in query processing. As for the hit ratios, we distinguish between the *local use ratio*, the percentage of the stored grid cells used by local queries, and the *peer use ratio*, the percentage used by peer queries. Simulation results are reported in Figure 10. In addition, we study the effect of parameter  $\Delta t$  used in our probability based replacement policy on the storage use ratio, as reported in Figure 12(b). Most results here are in line with their

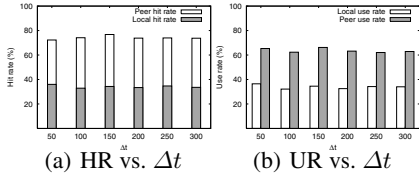


Fig. 12. Effect of  $\Delta t$

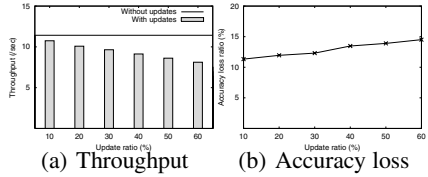


Fig. 13. Query Throughput and Accuracy

counterparts as reported in Section 6.3 on the storage hit ratios. Similar reasons as for the previous batch apply to these findings.

### 6.5 Message Counts

We also explore the wireless P2P message consumption of our method, distinguishing between two kinds of messages: *routing messages* and *query messages*. The former are used to disseminate routing table entries between peers, while the latter are used when forwarding queries and relevant grid cell data in the MANET. Average simulation results are reported in Figure 11. For the four sets of experiments covered, the query message cost is consistently very small compared to the routing message cost. However, the two are related: it is the extra routing messages that bring about the small number of query messages and fast retrieval of data for any device issuing queries. The extra routing messages pay off as they are utilized and amortized across the queries issued by multiple mobile devices. This study indicates that our query processing based on collaborative data sharing is efficient and robust.

From Figure 11(a), we see that the routing message cost of our proposal is insensitive to storage size variations. In contrast, the routing message cost exhibits an increasing trend as the grid becomes finer, as shown in Figure 11(b). Our storage and sharing mechanism uses grid cells as the basic units, the number of which increases as the grid granularity becomes finer. As a result, the increased numbers of grid cells involved in the storage and sharing produce more routing messages between the mobile devices. Figure 11(c) shows that the routing message cost increases almost linearly with the number of mobile devices, which demonstrates the scalability of our method. As shown in Figure 11(d), the message cost decreases as the update ratio increases. As updates tend to cause less data to be shared among peers, and as evictions due to updates do not invoke routing messages (all peers remove the relevant routing entries), the numbers of routing messages decrease. As less data are retrieved from peers, inter-device query messages decrease, too.

### 6.6 Throughput and Accuracy Under Updates

In this batch, we do not limit the number of queries each mobile device can issue and stick to range queries only because their search ranges can be exactly determined for accuracy concerns. We then examine the impact of updates on the system wide query *throughput*, the number of queries successfully answered per second in the simulation, and the *accuracy loss ratio*, the ratio of cells invalidated by co-occurring updates for an

abandoned query. Figure 13(a) shows that the presence of updates reduces the throughput compared to the cases without updates. This is so, as updates not only invalidate queries, but also consume resources that otherwise could be used by queries. As seen in Figure 13(b), the accuracy loss ratio increases as more updates occur, but stays below 15%. These results indicate that our proposal is robust and reliable under updates.

## 7 Conclusion

Assuming a hybrid mobile environment within which mobile devices can communicate wirelessly with not only an application server via a slow channel, but also with peer devices via fast P2P channels, this paper proposes a collaborative and predictive data sharing scheme that exploits the P2P capabilities of mobile devices.

Based on a uniform grid, we maintain a simple yet efficient collaborative indexing structure on the server and each mobile device within the server's coverage. Each device is able to issue spatial queries, and the devices request, store, and share data in units of grid cells. In contrast to the traditional C/S mobile computing, this collaborative sharing scheme exploits the high P2P bandwidth, thus shortening query response time significantly. Special routing tables are used to direct request forwarding among peers. A predictive cost model is proposed for storage replacement and routing table maintenance on resource-limited devices. This model takes into account the predicted movement of each device when assigning to its grid cells probabilities that they are to be reused by future queries. Extensive experiments conducted on a MANET simulator, elicit design properties of our proposals, indicating that they are efficient in answering queries and robust to data updates.

## Acknowledgments

The work of Zhiyong Huang, Hua Lu and Beng Chin Ooi was in part funded by A\*STAR under grant no. 032 101 0026. Christian S. Jensen is also an adjunct professor at Agder University College, Norway. His work was funded in part by the Danish Research Agency's Programme Commission on Nanoscience, Biotechnology, and IT.

## References

1. JiST/SWANS, <http://jist.ece.cornell.edu>
2. The R-tree Portal, <http://www.rtreeportal.org>
3. Barbará, D., Imielinski, T.: Sleepers and workaholics: Caching strategies in mobile environments. In: Proc. SIGMOD, pp. 1–12 (1994)
4. Basagni, S., Conti, M., Giordano, S., Stojmenovic, I (eds.): *Mobile Ad Hoc Networking*. Wiley-IEEE Press, New Jersey (2004)
5. Brilingaitė, A., Jensen, C.S.: Enabling routes of road network constrained movements as mobile service context. *GeoInformatica* 11(1), 55–102 (2007)
6. Broch, J., Maltz, D.A., Johnson, D.B., Hu, Y.-C., Jetcheva, J.: A performance comparison of multi-hop wireless ad hoc network routing protocols. In: Proc. MOBICOM, pp. 85–97 (1998)



7. Budiarto, S.N., Tsukamoto, M.: Data management issues in mobile and peer-to-peer environments. *Data Knowl. Eng.* 41(2-3), 183–204 (2002)
8. Chow, C.-Y., Leong, H.V., Chan, A.T.S.: GroCoca: Group-based peer-to-peer cooperative caching in mobile environment. *IEEE Journal on Selected Areas in Communications* 25(1), 179–191 (2007)
9. Dar, S., Franklin, M.J., Jónsson, B.T., Srivastava, D., Tan, M.: Semantic data caching and replacement. In: *Proc. VLDB*, pp. 330–341 (1996)
10. Hara, T.: Cooperative caching by mobile clients in push-based information systems. In: *Proc. CIKM*, pp. 186–193 (2002)
11. Hu, H., Wong, W.S., Lee, D.L., Zheng, B., Xu, J.: Proactive caching for spatial queries in mobile environments. In: *Proc. ICDE*, pp. 403–414 (2005)
12. Kortuem, G., Schneider, J., Preuit, D., Thompson, T.G.C., Fickas, S., Segall, Z.: When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. In: *Proc. P2P Computing*, pp. 75–91 (2001)
13. Ku, W.-S., Zimmermann, R., Wan, C.-N.: Location-based spatial queries with data sharing in mobile environments. In: *Report, USC-CS-TR05-843, Univ. of Southern California* (2005)
14. Lee, K., Lee, W.-C., Zheng, B., Xu, J.: Caching Complementary Space for Location-Based Services. In: *Proc. EDBT*, pp. 1020–1038 (2006)
15. Liu, B., Lee, W.-C., Lee, D.L.: Distributed caching of multi-dimensional data in mobile environments. In: *Proc. MDM*, pp. 229–233 (2005)
16. Luo, H., Ramjee, R., Sinha, P., Li, L.E., Lu, S.: UCAN: A unified cellular and ad-hoc network architecture. In: *Proc. MOBICOM*, pp. 353–367 (2003)
17. Lindemann, C., Waldhorst, O.P.: A distributed search service for peer-to-peer file sharing in mobile applications. In: *Proc. P2P Computing*, pp. 73–80 (2002)
18. Nievergelt, J., Hinterberger, H.: The grid file: an adaptable, symmetric multikey file structure. *ACM TODS* 9(1), 38–71 (1984)
19. Ren, Q., Dunham, M.H.: Using clustering for effective management of a semantic cache in mobile computing. In: *Proc. MobiDE*, pp. 94–101 (1999)
20. Saltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the positions of continuously moving objects. In: *Proc. SIGMOD*, pp. 331–342 (2000)
21. Tao, Y., Faloutsos, C., Papadias, D., Liu, B.: Prediction and indexing of moving objects with unknown motion patterns. In: *Proc. SIGMOD*, pp. 611–622 (2004)
22. Xu, B., Wolfson, O.: Data management in mobile peer-to-peer networks. In: *Proc. DBISP2P*, pp. 1–15 (2004)
23. Yin, L., Cao, G.: Supporting cooperative caching in ad hoc networks. In: *Proc. INFOCOM* (2004)
24. Zheng, B., Lee, D.L.: Semantic caching in location-dependent query processing. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) *SSTD 2001. LNCS, vol. 2121*, pp. 97–116. Springer, Heidelberg (2001)

# A Study for the Parameters of a Distributed Framework That Handles Spatial Areas\*

Verena Kantere and Timos Sellis

School of Electr. and Comp. Engineering, National Technical University of Athens  
{vkante, timos}@dbnet.ece.ntua.gr

**Abstract.** In this work we study the construction of a framework for autonomous sites that are bound to spatial information and that form an overlay network; we investigate the parameters of such a distributed system in order to perform search guided by locality and directionality in space. We present the main parameters of the framework and propose appropriate values for them. A theoretical study discusses the overall search efficiency limits for two approaches concerning the main framework parameter, i.e. the distance metric. Furthermore, the behavior of the rest of the framework parameters is examined based on an experimental study.

## 1 Introduction

In this work we are interested in creating a framework that is inherently distributed and operates in space or handles spatial data. Our basic assumption is that there are autonomous sites that each is 'bound' to a specific spatial area. This can be interpreted in two ways:

- each site handles the information for a specific spatial area
- each site resides on a specific spatial area

We assume that the autonomous sites form an *overlay* network by having links among each other. Thus, each site, hereafter *node*, of the overlay can route search processes targeting other nodes that either reside on a specific area, or handle information about a specific area. In that sense we consider a distributed system related to spatial information that operates based on the features of spatial areas. Accordingly, these spatial areas 'exist' in the system either by being handled or by being the reference point of a network node. In the following we will address the described distributed system without specializing to any of the two interpretations. Therefore, we will talk about spatial areas that *exist* in the system, without determining if they exist as reference points or handled information of network nodes.

---

\* This work has been funded by the project PENED 2003. The project is cofinanced 75% of public expenditure through EC - European Social Fund, 25% of public expenditure through Ministry of Development - General Secretariat of Research and Technology and through private sector, under measure 8.3 of OPERATIONAL PROGRAMME "COMPETITIVENESS" in the 3rd Community Support Programme.

The considered space on which such a system can operate can be either continuous or discrete. For the second case, we assume that space is represented using a 2-dimensional grid. For simplicity, in this work we assume that each spatial area is identified by the  $x, y$  coordinates of the center of it. Thus, we do not consider any influence of the size or the shape of the area - this is future work. In the following, again for simplicity we consider rectangular areas of the same size, that in case of a grid-partitioned space, coincide with the grid cells.

In such a distributed system our major interest is the performance of search operations for spatial areas. We assume that the only information available is the area ids, (i.e. coordinates). Thus, search for an area has to be performed guided by *locality* and *directionality*. A search process for a specific area has to be routed towards this area following a network path. In order to reach the target area the search process has to be propagated to areas that are close (locality) to the target area, and of course, towards the same direction (directionality) to the target area. We define two major requirements:

- a search operation for any spatial area originating from any spatial area can be performed: this means that a search routing path for target area  $T$  will eventually reach  $T$ , if this exists in the system, no matter which is the source area  $S$  on which the search process originates.
- any search operation must be guaranteed to be performed *efficiently*: this means that any search routing path has to be very short (in terms of number of routing hops) compared to the total number of existing areas in the system. Later on we will make this requirement more specific.

Given the above assumptions and requirements, our goal is to investigate the parameters that constitute a framework that can realize the described system. It is obvious that, the search guarantees that we require, rely inherently in the way that spatial areas in the system are connected, i.e. in the links that each spatial area maintains to other areas.

Such a framework may have various real applications. One such application can be a distributed system of nodes following the peer-to-peer paradigm, that handles spatial information. Another representative application may be a distributed network of sensors that reside in space, or ad hoc networks that aim to peer-to-peer cooperation. Finally, the considered framework is applicable to stable wireless or even adaptable to mobile networks.

In Section 2 the necessary links between spatial areas are defined based on a discussion about locality and *proximity* of areas. In Section 3 we discuss the performance of search using the necessary area links. In Section 4 we discuss the addition of more area links that can expedite the search process and we perform an experimental study. Finally, Section 5 discusses related work and Section 6 concludes the paper.

## 2 Linking Spatial Areas

In order to be able to perform searches for any area from any other area, each area should be linked with one or more other areas. As discussed in Section 1, we assume that search is routed according to locality and directionality. This means that search is propagated to the area that is closer to and towards the same direction with the sought

area, choosing from the available areas that are linked to the one on which the search is currently operating. Thus, it is very important for the framework to provide area links to each area, that guarantee the correct routing of any search operation.

Since we consider the 2-dimensional space, there are at most 4 *vertical* to each other directions towards which a search can be routed, and, thus, towards which an area should have links to other areas. Let us assume that space is addressed with the orthonormal set of axes  $\mathcal{O} = \{x'y, y'y\}$ . Then, with reference to a specific area  $O$ , there are four semi-axes that determine four vertical directions  $\mathcal{D}_O = \{Ox', Ox, Oy', Oy\}$ . Of course we can rotate the orthonormal axes and produce new equivalent sets of vertical directions. It is straightforward that, for the continuous space, rotating the orthonormal axes, does not make any difference. Yet, for a grid-partitioned space the rotation of the axes may influence the search process. It is out of the scope of this work to examine the behavior of search for rotated sets of axes with respect to a grid-based partitioning of space.

The  $\mathcal{O}$  set of axes with reference to area  $O$ , produces four *quadrants* that originate at  $O$ . As proved in [7], each area  $O$  should have exactly four links (if these are available), one inside each one of the set of four quadrants that are defined with respect to  $\mathcal{O}$ . These four links are called *successors* of  $O$  and are necessary in order for  $O$  to perform search for any other area in the system, according to locality and directionality<sup>1</sup>.

### 2.1 Defining Locality of Areas

Up until this point we have talked about locality of areas without specifying how the *closeness* of areas is defined. Yet, this definition is of vital importance since the essence of the framework is concentrated on the notion of locality. Locality has to be defined based on a metric that determines the distance between two areas. It is straightforward to consider the *Euclidean* distance metric, hereafter  $E$ , as our first option, since it measures the real spatial distance between two points. The  $E$  distance between two areas  $A_1(x_1, y_1), A_2(x_2, y_2)$  is defined as:

$$E(A_1, A_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{1}$$

Furthermore we consider the  $D$  distance between two areas  $A_1(x_1, y_1), A_2(x_2, y_2)$  that is defined as:

$$D(A_1, A_2) = (d_1(A_1, A_2), d_2(A_1, A_2)) \tag{2}$$

where:

$$d_1 = \max\{|x_1 - x_2|, |y_1 - y_2|\} \tag{3}$$

$$d_2 = \min\{|x_1 - x_2|, |y_1 - y_2|\} \tag{4}$$

In the above definition we add that  $d_1$  has priority over  $d_2$ ; thus, the  $D$  distance is a metric (see [7]). The reason we consider  $D$  is because it defines square equidistant

<sup>1</sup> Note, that even though two links per area are necessary in order to keep the overlay connected, these are not enough in order to perform search by locality and directionality. Intuitively, since there are four vertical directions with respect to  $O$ , there is a need for one link in each direction, in order to propagate a search process towards any other area in space.

frontiers instead of circular. The effect of  $D$  is to consider that areas are equally distant from an area of reference if we have to access the same minimum number of areas in order to reach them. This is more apparent if we consider a grid-partitioned space: Actually, the  $D(A_1, A_2)$  represents the notion of how many areas apart are the two areas  $A_1, A_2$  if we travel on a straight line, combined with a weight related to their real spatial distance. This is very useful in the discussed framework since it considers traveling (i.e. searching) ‘hopping’ from area to area based on locality and directionality.

The  $D$  distance encapsulates both the Euclidean and the Manhattan distance. If  $E$  and  $Mn$  denote the Euclidean and the Manhattan distance of  $A_1$  and  $A_2$  respectively, we have  $E = \sqrt{d_1^2(A_1, A_2) + d_2^2(A_1, A_2)}$  and  $Mn = d_1(A_1, A_2) + d_2(A_1, A_2)$ .

A distance metric should have properties that enable or amend application-specific operations. Concerning a distributed environment related to spatial information in the two ways described in Section 1, a good property of a distance metric used to define locality is to be representative of the search path lengths. Specifically, the following proposition holds:

**Proposition 1** (*Desired Distance Property*). *It is desirable that for a distance metric  $M$  the distance value  $M(A_1, A_2)$  for two areas  $A_1$  and  $A_2$  that exist in the overlay is representative of the search path length between  $A_1$  and  $A_2$ .*

The above property is useful in many situations that can occur in the investigated framework. First of all, in case of networks that consist of autonomous nodes this property allows them to take correct decisions about the area they prefer to reside or to handle (depending on the application, see Section 1). For example, if a joining node is interested in an area  $A$  that covers several areas  $\mathcal{A} = \{A' \subset A\}$  on which it can reside/handle, then this node can pick an area  $A' \in \mathcal{A}$  and be sure that it is close enough, (using the employed distance metric), to the rest of the areas  $\mathcal{A} - \{A'\}$ . Moreover, a network node that is interested in areas that are far from the area that it handles/resides in, can keep links to remote nodes that are close to these areas (for more details on remote links see Section 4). If the distance metric does not indicate the search path length, then the node may keep links to nodes that do not serve its needs well. Finally, links to specific remote nodes may expedite searches for specific areas.

## 2.2 Defining Proximity of Areas

Locality is the basic notion of our framework, since it is necessary in order to do the following: (a) define the successors of the areas, (b) search for an area. However locality is not enough: since space is symmetric<sup>2</sup>, the equidistant frontiers determined by the distance metrics  $E$  and  $D$  define more than one equidistant areas with respect to an area of reference. Thus, the distance metric defines a partial ordering of all the areas with respect to the reference area. Yet, it is necessary to refine this partial ordering into a total one, in order to be able to determine successors and search routing hops in all cases. Therefore we define the notion of *proximity* of areas, that is an ordered list of criteria that produce a total ordering of areas with respect to a reference area.

<sup>2</sup> Obviously, continuous space is symmetric along infinite axes and a grid-partitioned space is symmetric along the axes of  $\mathcal{O}$  and the diagonals of the quadrants that the latter define.

**Definition 1.** Assume a reference area  $A$  and another area  $A'$ . There are four quadrants that are determined by the set of axes  $\mathcal{O}$ . The proximity of  $A'$  to  $A$  is determined according to the following criteria, in the specified order:

- $P1$  the distance of  $A, A'$  measured with some distance metric, for example,  $E$  or  $D$ .
- $P2$  priority of quadrants (see below)
- $P3$  priority of coordinate values:
  - $P3a$  priority of  $x$  values
  - $P3b$  priority of  $y$  values

The above definition ensures that whatever distance metric is used, the partial orderings introduced by the distance metric can be sorted out with the rest of the criteria. The priority of quadrants is determined according to a pre-specified global priority order of quadrants, (for example  $1^{st} < 2^{nd} < 3^{rd} < 4^{th}$  where the quadrants are numbered in a clockwise order). In each quadrant there may be many areas that are equally distant from the reference area  $A$ . Thus, it is necessary to distinguish among them using the  $P3$  criterion, which orders equidistant areas according to the  $x$  and  $y$  values<sup>3</sup> (for example areas for areas  $A_1, A_2$  with the same  $x$  coordinate  $A_1$  has priority over  $A_2$  if  $A_1.y < A_2.y$ ). Note that not all the criteria are or have to be applicable for any distance metric that can be defined/used in the framework. Yet, the above is a complete set of criteria that can solve and partial ordering ambiguities in any case.

### 3 Search by Proximity

The definition of proximity on top of locality in the previous section enables linking and search of areas in the framework. Focusing on the search process, if we perform search by proximity in the addressed space, the search will eventually converge to the target area. However, it is very important to note, that this is guaranteed only if the proximity criteria are fulfilled in the pre-defined order. This means that in order for search to decide about the next routing hop based on a specific proximity criterion, there has to be a tie for all the criteria ahead in the defined hierarchy. For example, suppose that search about a target area  $T$  is routed to area  $A'$  instead of  $A$  according to priority of  $y$  values (criterion  $P3b$ ). Then, this means that there is a tie for criteria  $P1, P2, P3a$  for areas  $A$  and  $A'$  with respect to the area  $T$ . We say that a proximity criterion is *fulfilled* if there is a tie for it.

**Proposition 2.** If the proximity criteria are fulfilled in the specified global order of definition<sup>4</sup> then a search process for any target area originating on any source area will eventually converge to the target area.

Therefore it is important to examine if the priority criteria are always fulfilled in the specified order for the considered distance metrics,  $E$  and  $D$ . Furthermore, in order for

<sup>3</sup> Of course, it is possible to switch the order of criteria  $P3a$  and  $P3b$ . Moreover, even though the proposed definition of proximity is artificial, it can be shown that the proposed criteria and their respective hierarchy leads to an even linkage of areas in the overlay.

search by proximity to guarantee the convergence to the target area, it is necessary to be propagated always to an area that is closer or equally close to the target in terms of locality than the previous one. If this is not possible then the search path may diverge, in terms of locality, from the target area. Let us examine the case on this matter for the considered distance metrics,  $E$  and  $D$ . For this purpose, we consider an example. Assume that the search about area  $A_0$  is iterated on area  $A_n$ ,  $n > 2$ . Then  $A_n$  has to propagate the search process to the successor of it that is equally close or closest than itself to area  $A_0$ . It is straightforward that the successors which can be considered for the propagation are the two successors that are on the same semi-plane as  $A_0$  with reference to  $A_n$ . However, is this always possible? If not, can this search path converge to  $A_0$ ?

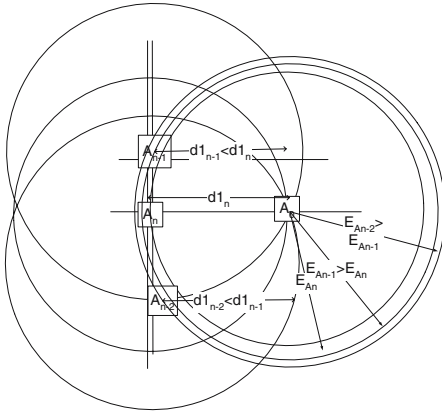
### 3.1 Search by Proximity Using the $E$ Distance Metric

Figure 1 shows the situation of the example where distance  $E$  is used for the definition of locality. It is obvious that both the successors of  $A_n$  that are on the same side as  $A_0$  can be farther away from  $A_0$  than  $A_n$  itself. Since there is no equality with respect to the  $E$  values of the two successors, we cannot proceed based on the rest of the proximity criteria. Yet, since search is guided by locality first of all, it is rational to choose to propagate the search process to the successor that is closest to  $A_0$  from the two of them. Assume that the closer successor is  $A_{n-1}$ . Now  $A_{n-1}$  has to repeat the procedure and propagate the search process to its own successor that is closest to  $A_0$ . Assume that this is  $A_{n-2}$ . It is obvious that  $A_{n-2}$  can be farther away than  $A_{n-1}$  from  $A_0$ . Therefore, this situation may occur on several consecutive nodes of the search path when locality is defined using  $E$ . Thus, the proximity search may be driven away instead of closer to the target area.

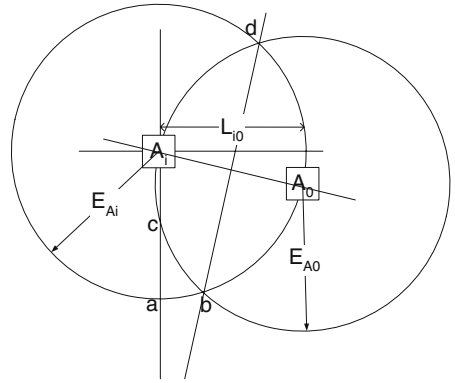
Yet, will the search process converge eventually to the target area, or is it possible to diverge forever? First, note that each node  $A_i$  on which the search is propagated has to be closer vertically or horizontally (depending on which part of space, with respect to  $A_i$ ,  $A_0$  resides). More formally, the distance of areas  $A_i$ ,  $A_0$  on the  $x'x$  axis is decreasing (in the specific example) as the search is executed recursively. Additionally, the line that passes through the points of intersection of the circles around  $A_i$ ,  $A_0$  with radiuses  $E_{A_i} = E_0 = |A_i A_0|$  constantly diverges from the  $y'y$  axis, as it is shown in Figure 2. This happens because  $A_i$  has to have a bigger value on the coordinate that is vertical to  $d_1$  as  $i$  increases, since  $A_i$  has to reside out of the  $A_{i-1}c$  arc. Thus, the space  $abc$  of circle  $(A_i, E_{A_i})$  in which  $A_i$  should have a successor (in order for search to diverge from  $A_0$ ) decreases as  $i$  increases (i.e. at each hop of the search process). Moreover, the space  $abc$  decreases even faster as  $i$  increases because the circles  $(A_i, E_{A_i})$ ,  $(A_0, E_{A_0})$  enlarge, meaning that eventually the points  $a, b$  and  $c$  'fall' onto each other. Hence, we can conclude that the search process can diverge to nodes far from the target area but up to a point with respect to their distance from the latter.

It is easy to see that the bigger the difference of distances  $E(A_i, A_0)$  and  $E(A_{i-1}, A_0)$ , the fewer the nodes  $A_i$  of divergence are, meaning the smaller  $i$  is. It is obvious that, if the considered areas are defined on continuous space,  $\lim(E(A_{i-1}, A_0) - E(A_i, A_0)) \rightarrow 0 \Rightarrow i \rightarrow \infty$ .

Beyond the convergence of search to the target area, we would like to investigate if the good metric property 1 holds for  $E$ . It can be proved that:



**Fig. 1.** A search process targeting area  $A_0$  and iterated on area  $A_i$  can diverge to areas that are farther away than  $A_i$ , if there is space where it can have successors that are farther away than itself from  $A_0$



**Fig. 2.** Using the  $E$  metric may result in propagation of the search process for area  $A_0$  to area  $A_{n-1}$  for which  $E(A_0, A_{n-1}) > E(A_0, A_n)$  and so on

**Theorem 1.** *The distance value  $E(A_n, A_0)$  does not give any bound for the search path length between areas  $A_n$  and  $A_0$ .*

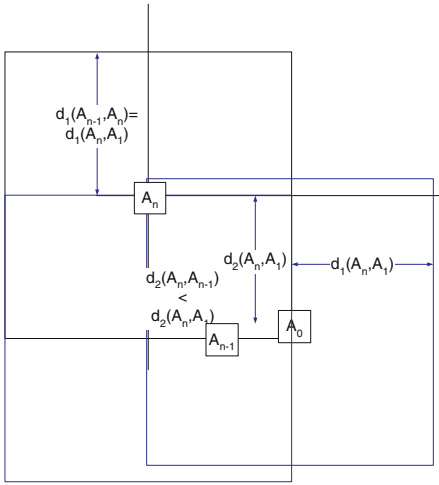
The proof of the above theorem is based on the fact that it is possible for search to have to diverge in terms of locality from the target area (see [7]). The combination of the above discussion and theorem make apparent that the  $E$  metric does not have the good property described in Section 2.1. The distance value  $E(A_n, A_0)$  is not representative of the search path length between areas  $A_n$  and  $A_0$ . Thus, the following proposition summarizes these conclusions:

**Proposition 3.** *Using the  $E$  distance metric may lead a search process to diverge from the target area. However, this divergence can be extended in a limited space. Thus, if the framework is defined on continuous space it is possible for a search process to diverge infinitely and not reach the target. However, if space is grid-partitioned, then even though search may diverge, it will eventually converge to the target area. Finally,  $E$  values do not represent the search path length for any pair of areas in the overlay.*

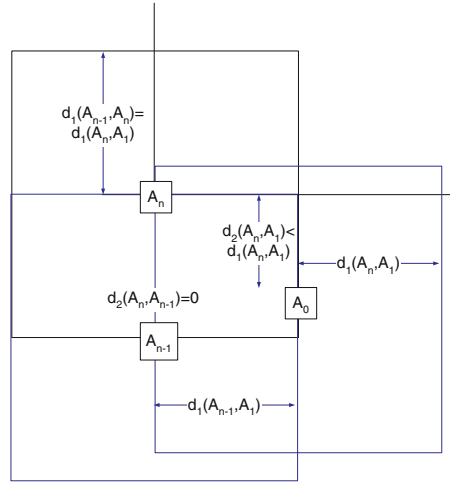
### 3.2 Search by Proximity Using the $D$ Distance Metric

Figure 3 shows a situation of search of target area  $A_0$  iterating on area  $A_n$ . It is obvious that if the  $D$  distance is used instead of  $E$ , there is no such area as  $abc$  in Figure 1 in which area  $A_n$  could have a successor  $A_{n-1}$  that is farther away from area  $A_0$  than itself. The reason is that the equidistant frontiers defined by  $D$  are parallel/vertical to the set of axes  $\mathcal{O}$  that define the quadrants where the successors reside; therefore, the frontier of  $D(A_n, A_0)$  centered on  $A_n$  and the same frontier centered on  $A_0$  intersect each other in a way that there is no space of the first or the second left uncovered by the second or the first, respectively. This is obvious in Figure 3: the frontiers  $D(A_n, A_0)$





**Fig. 3.** The use of distance  $D$  instead of  $E$  leaves no space in which area  $A_n$  could have a successor  $A_{n-1}$  that is farther away from area  $A_0$  than itself



**Fig. 4.** Areas  $A_n$  and  $A_{n-1}$  can reside in different (yet contiguous) quadrants of  $A_0$ ; or  $A_{n-1}$ , and  $A_0$  can either be on the same horizontal or vertical axis (not shown here)

centered on  $A_n$  and on  $A_0$  cover each other totally in the quadrant of  $A_n$  where  $A_0$  resides (and the opposite, of course). The  $D$  distance is composed of two metrics, a basic one,  $d_1$  that defines the (coarse) equidistant  $d_1$  frontier, and a second one,  $d_2$  that refines the distance values on the same frontier, in order to avoid too many equally distant areas on the same  $d_1$  frontier. Therefore, the successor of  $A_n$  in the quadrant that  $A_0$  resides will be either closer to  $A_n$  (in terms of  $d_1$ ) or equally distant from it. Let us suppose that the successor of  $A_n$  in the quadrant that  $A_0$  resides is  $A_{n-1}$ . Then, if  $d_1(A_n, A_{n-1}) < d_1(A_n, A_0)$ ,  $A_{n-1}$  resides inside the intersection of the frontiers  $D(A_n, A_0)$  centered on  $A_n$  and  $A_0$ . Hence, unlike  $E$  the distance  $D$  is totally suitable, both for continuous and non-continuous space, for search based on locality (and therefore proximity) because it guarantees a tie in the primary metric  $d_1$  if the choice of locality is decided based on  $d_2$ . Further on, if the  $d_2$  distance value between  $A_n$  and  $A_0$  is the minimum:  $d_2(A_n, A_0) = 0$ , which means that  $A_0$  resides on a semi-axis originating on  $A_n$ , then it is not possible to have a  $A_{n-1}$  successor in the same quadrant that  $A_0$  resides, such that  $d_1(A_n, A_{n-1}) = d_1(A_n, A_0)$ .

Let us suppose that  $d_1(A_n, A_{n-1}) = d_1(A_n, A_0)$ , as in Figure 3. We would like to know what is the maximum number of search hops that can iterate on areas  $A_i$  with the same value  $d_1(A_n, A_i)$ . It is straightforward that in such case, the successor of  $A_n$  on which the search process is propagated,  $A_{n-1}$ , and  $A_0$  can either be on the same horizontal or vertical axis, or  $A_n$  and  $A_{n-1}$  can reside in different (yet contiguous) quadrants of  $A_0$ , as shown in Figure 4. This means that  $A_{n-1}$  can have as a successor the target area  $A_0$ , if there are no areas  $A_k$  with  $d_1(A_k, A_0) < d_1(A_n, A_0)$ . Therefore, search is actually guided almost solely by locality. The following theorem holds:

**Theorem 2.** *The distance value  $D(A_n, A_0)$  gives an upper bound for the search path length between areas  $A_n$  and  $A_0$ .*

The proof of the above theorem is based on the fact that the  $D$  distance guarantees continuous convergence of any search towards the target area (see [7]). The following proposition summarizes these conclusions:

**Proposition 4.** *The  $D$  metric enables search that is always guided by proximity. Search not only converges to areas that are closer to the target, but it is guided only by the metric  $d_1$ , since there can be up to two hops with the same  $d_1$  value. Finally, the  $D$  distance values represent the search path length for any pair of areas in the overlay.*

## 4 Additional Links That Expedite Search

Until this point we have assumed that each area in the system has links only to the closest areas and we have named these links the successors of the area. However, in the considered framework areas may be provided with knowledge about areas that are farther away than their successors. These links, that we call *indexed* areas, can make search paths much shorter than if employing only the successors. As it is very common in Distributed Hash Table techniques (for example Chord [14], CAN [11], etc.), the efficiency of the search process in such a distributed environment is derived by providing each overlay node with knowledge of other nodes that are far away, based on a distance metric, and the density with respect to the (data and node)id space of the known nodes fades out with their distance from the reference node. The most widely-known DHT techniques, augment the distance of the known nodes from the reference node with logarithmic steps; thus, they guarantee search paths that are  $O(\log n)$  long, for an overlay with  $n$  nodes. Inspired by these techniques, and since we intend to construct a framework that is applicable in similar environments that handle spatial data, we would like to provide the framework with a similar indexing technique, based on the locality of areas in the system. Therefore, each area should have links to other areas, beyond its successors, that are pre-specified and that will guarantee a search path with limited length.

Towards this end we have to study the directions with reference to an area along which indexing should be applied, as well as the influence of the distance metric to the indexing operation.

As we have discussed earlier, searching for a target area  $T$  in our framework is performed based on the proximity criteria, following the appropriate successor link on each area on which the search process is iterated. It is straightforward that, in order to expedite the search process by allowing the propagation of it to areas that are farther away than successors, there is a necessity for index links towards each direction of each of the four successors of an area. This means that each area should index other areas in all the four quadrants that are defined by the set of axes used to address the space. We remind that we have addressed space with the the pair of orthonormal axes that, in case of a grid, are parallel to the columns and rows. We have named this set of axes  $\mathcal{O}$ .

Indexing areas in the quadrants with reference to a specific area, can be achieved by indexing the closest areas towards specific directions. Actually, one-dimensional DHTs (e.g. Chord [14]) index nodes in the exact same way, except that in their case there is a

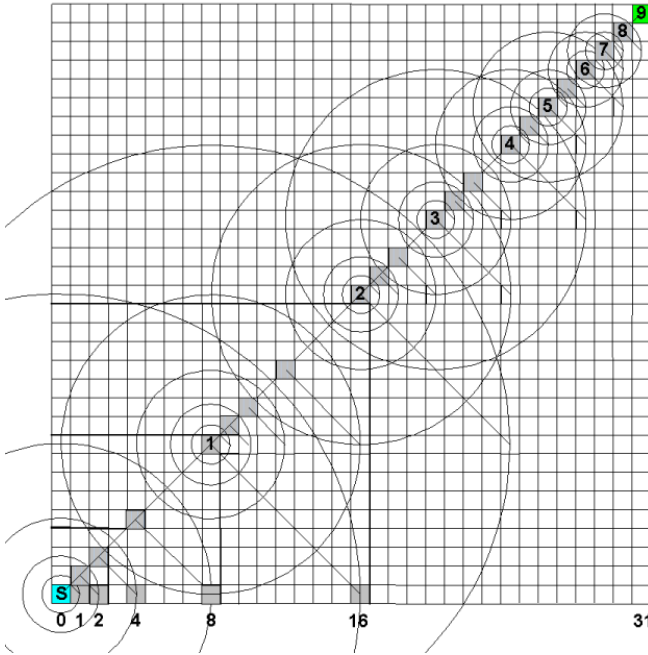
sole dimension to index. The most straightforward way to apply this requirement to a 2-dimensional space is to index the orthonormal axes that address the space. Similarly to the DHT methods we can index with logarithmic steps the four semi-axes of  $\mathcal{O}$ . This means that each area  $A$  will index the areas that are the closest to points  $b^i \cdot \lambda$  on each one of the four orthonormal semi-axes that originate on  $A$ . The parameter  $b$  is the logarithmic base; following the steps of DHT techniques, we can choose  $b = 2$ . Also,  $i$  is the parameter that augments the index step and is an integer, i.e.  $i = 0, 1, 2, \dots$ . Finally, if we consider continuous space, we have to define a measurement unit,  $\lambda$ . However, if we consider a grid-partitioned space it is natural to use the grid cell as the measurement unit:  $\lambda = 1$ . For convenience we will continue this discussion assuming that space is grid-partitioned; all results can be applied in a straightforward manner to continuous space. Note that we use the words ‘cell’ and ‘area’ interchangeably.

Let us assume a  $n \times n$  basic grid and that indexing is guided by logarithmic steps defined as above. For simplicity we focus on the area  $S$  at the lower left corner of the grid and the index on the horizontal right semi-axis (see Figure 5). We want to investigate the influence of the indexing to the complexity of the search path length towards any direction in space. Therefore we have to study which, how many and the density of the indexed areas of area  $S$ .

It is straightforward that if the areas on the horizontal axis exist, then the ones that should be indexed, will be indexed themselves. Specifically,  $S$  will index areas  $A_i = (2^i, 0)$ , i.e.  $i \leq \log_2 n$ . Figure 5 shows this situation for  $n = 32$ . Node  $S$  is the lowest leftmost area and the (should-be-indexed but also) indexed areas are the light gray areas on the same row. Thus,  $S$  indexes  $i \leq \log 32 = 5$  areas. Obviously, each  $i - 1$  indexed area divides the total length (in terms of grid cells, but also the real length) from  $S$  to  $A_i$  into half. It is very important to observe that due to this fact  $S$  can find any area along the horizontal axis in  $\log n$  hops, which is the actual goal of the indexing process.

Yet, what happens if the areas on the horizontal axis do not exist?  $S$  should index other cells based on their proximity to absent areas that should be indexed. Thus, the way that locality is defined influences in a pretty direct manner the final indexing of areas and the result of the indexing algorithm. Let us assume that only the cells residing on the diagonal that starts on  $S$  exist in the system. Figure 5 shows that if  $S$  uses the distance  $E$  to define locality, it ends up indexing  $\log 32 = 5$  but very close (to each other and to  $S$ ) areas. The reason is that the equidistant frontiers defined by  $E$  indicate that the closest existing areas are far closer to  $S$  in terms of grid cells than the areas that should be indexed. Specifically, each area  $A_i = (2^i, 0)$  that should be indexed is closest to the area  $A'_i = (2^{i/2}, 2^{i/2})$ , meaning that  $S$  ends up indexing areas that have double density in space than they were supposed to. Most importantly, the knowledge of  $S$  about the overlay reaches a quarter of the total distance, since its most remote indexed area is  $A'_{\log n} = (2^{\lceil \log n / 2 \rceil}, 2^{\lceil \log n / 2 \rceil})$  instead of  $A_{\log n} = (2^{\log n}, 2^{\log n})$ . In Figure 5 for  $n = 32$ , the knowledge of  $S$  about the overlay reaches up to  $2^{\lceil 5/2 \rceil} = 8$  areas, whereas there are areas up to 31 grid cells away. If  $S$  uses the  $D$  metric instead of  $E$ , the result would be the same. Both distance metrics indicate the same area residing on the diagonal for each area of the horizontal axis that should be indexed.

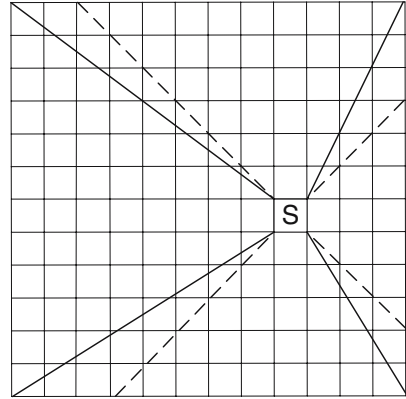
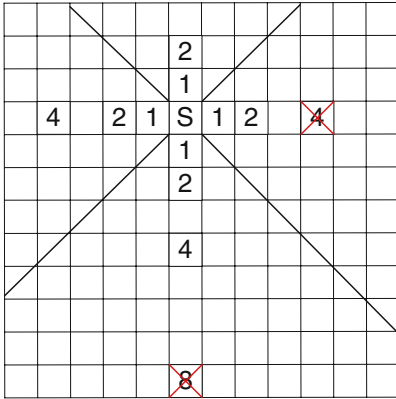
The above situation leads to bad indexing of space: even though the intention of the indexing algorithm is to guarantee to each node knowledge of other areas depending



**Fig. 5.** The lower left area  $S$  of a  $32 \times 32$  grid initiates a search for the upper right area and the overlay stores only the areas residing on the diagonal. If the  $E$  distance is used, the search process needs two hops instead of one to be propagated to an area that is half-way to the searched area. Thus, the total number of hops is 9 instead of 5.

on the  $\log n$  value, the distribution of areas stored in the system influences the indexed areas and cause deviation of the initial goal of the indexing algorithm. The result is that in the worst case the search path length may be up to double the length that is intended by the indexing algorithm. Figure 5 shows such a situation: in a  $32 \times 32$  grid the lowest leftmost area initiates a search for the upper rightmost area and the overlay stores only the areas residing on the diagonal. The search process needs two hops instead of one to be propagated to an area that is half-way to the searched area. Thus, the total number of hops is 9 instead of 5. In general, in the worst case, the total number of search hops is  $2 * (\log n - 1) + 1$ , instead of  $\log n$ .

As we have seen, both the distance metrics  $E$  and  $D$  have a similar behavior in this matter for the diagonal of the grid. Yet the two distances differentiate their behavior for other area distributions. The angle of the diagonal with respect to the horizontal axis is  $45^\circ$ ; as the angle of the area distribution diminishes towards the horizontal axis, the  $E$  distance metric behaves better than  $D$ , since it indicates as closest (to the areas that should be indexed) areas that are closer to the  $2^i, i = 0, \dots, n - 1$ , columns. Oppositely, for area distributions that follow a line with angle greater than  $45^\circ$ , the  $E$  metric is worse than  $D$  in that it indexes areas farther away from the columns of the respective areas that should be indexed.



**Fig. 6.** The indexing of the quadrant diagonals may result in missing areas that should be indexed. In the example shown there are two areas that should be indexed but do not 'correspond' to any area on the respective quadrant diagonals.

**Fig. 7.** The diagonals and the diameters of an area get closer as the position of the area gets closer to the center of the grid

**4.1 Indexing Various Semi-axes**

The length of each semi-axis of the area of reference is not always indicative of the diameter of the overlay. The diameter of the overlay in each quadrant, let us call it the *quadrant diameter*, is actually the line that originates on the area of reference and ends at the outer corner of the quadrant. The diameter of the quadrant is different than the diagonal of the quadrant, that is actually the line that originates on the area of reference and has angle 45°. As it is shown in Figure 7 the diagonals and the diameters of an area get closer as the position of the area gets closer to the center of the grid. The quadrant diameters are longer than the quadrant diagonal or the respective orthonormal semi-axes. the quadrant diagonal or the respective orthonormal semi-axes, the quadrant diameter intersects more cells than the other two semi-axes. If we choose to index the diameters instead of the semi-axes of the quadrants, then we may achieve a better indexing of space, since we aim at knowledge of areas that are towards the longest straight line in each quadrant. Note that it is not wise to index the diagonals of the quadrants since they can actually be much shorter (in number of areas that they intersect) than the respective semi-axis. Figure 6 shows that if we index the quadrant diagonals we may miss areas that could be indexed.

**4.2 Indexing More Than Four Semi-axes**

In the previous discussion we have acknowledged the importance of the logarithmic indexing of space evenly towards any direction on the grid and we have investigated ways to achieve this goal by best exploiting the potential of the distance metrics and by changing the setting of indexing directions. In this section we will discuss the 'brute'

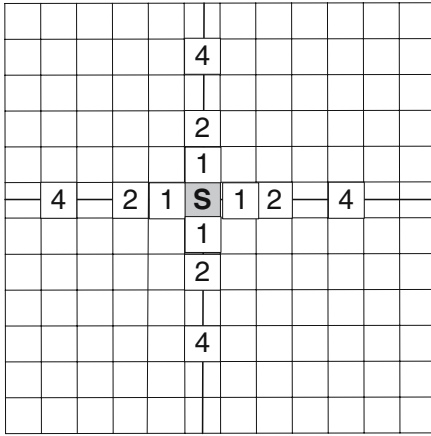


Fig. 8. Indexing 4 semi-axes

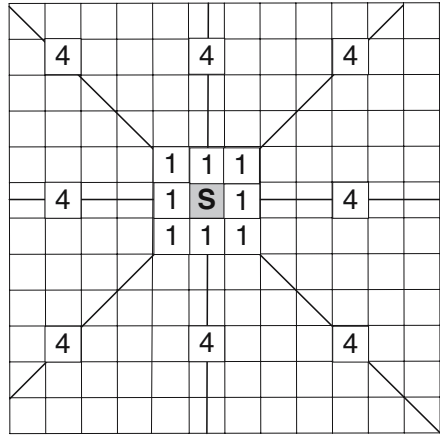


Fig. 9. Indexing 8 semi-axes

solution of this problem simply by adding more indexing semi-axes other than the horizontal and vertical ones of the basic framework. Of course it is straightforward that the more indexing semi-axes are added for each area, the bigger the index of this area will be. Thus, the interesting part of this solution is to examine the trade-off between the index size and the evenness of the distribution of the indexed areas taking into consideration the logarithmic scale. This means that, ideally, we would like to keep the index size constant as we increase the indexing directions. Therefore, since the indexing semi-axes are increased, we have to reduce the indexing areas on each one of them. Specifically, the index size can remain the same if for every semi-axis that we introduce in a part of the space we reduce by half the indexing areas on the semi-axis that is responsible for the indexing of this part of space. Figure 8, 9 show an example of indexing using 4 semi-axes and an example of indexing using 8, respectively. Suppose that we use the  $d_1$  frontiers to define the position of the indexed areas; then fewer and sparser areas will be indexed on each semi-axes in Figure 9 than in Figure 8.

However, the direct consequence of this solution is that fewer areas close to the area of reference are indexed as new indexing semi-axes are added. This means that even though we even the indexing towards remote areas, we deprive close areas from good indexing. In more detail, areas close to the area of reference may have to be reached with hops from successor to successor because the search process cannot hop long distances using indexed areas. Taking into account that the basic framework indexes the four semi-axes (one corresponding to each quadrant) using the logarithmic base  $2 = 2^1$ , we can infer that, assuming that there are  $k$  indexing semi-axes in a quadrant, they should all use the logarithmic base  $2^k$ .

Considering that the indexing of close areas is really important in order for our framework to guarantee that close areas can be found within very few hops, we propose the employment of 8 indexing semi-axes in total (i.e. 2 in each quadrant): the four horizontal/vertical semi-axes that define the quadrants as well as the diagonals of the quadrants. The 8 indexing semi-axes have to be indexed using the logarithmic base

$2^2 = 4$ . Figures 8, 9 show the indexing of the 4 and 8 semi-axes according to the proposed solution. We reckon that using more indexing semi-axes can severely hurt the efficiency of search of close areas, since each semi-axis is very sparsely indexed.

### 4.3 Experimental Study About the Indexing Directions

In the previous discussion of this section we have proposed that the spatial areas should index other remote areas in order to expedite the search process. We have concluded that the search efficiency can be influenced by:

- the type of indexing semi-axes
- the number of indexing semi-axes

However, it is not possible to prove theoretically which indexing combination is more efficient with respect to search and under which overlay circumstances this can be true. Thus, we conducted experiments in order to study the behavior of the framework for combinations of the following:

- indexing (a) the orthonormal semi-axes or (b) the quadrant diameters
- indexing (a) four or (b) eight semi-axes evenly distributed in space

Without loss of generality we have chosen to experiment on an application of the first of two types discussed in Section 1. Specifically, our experimental setup is a peer-to-peer overlay in which nodes handle spatial information. We call this framework SPATIALP2P (see 8) and due to lack of space we do not elaborate on framework details. For our experimental setup we have chosen SPATIALP2P to handle a grid-partitioned space. In the experiments we range the size of the  $n \times n$  grid from  $n = 32$  to  $n = 128$  and we examine *sparse* to *very dense* overlays, meaning for a total number of nodes  $N$  ranging from  $N = 1000$  to  $N = 10000$ . We used random data distribution. All the experiments measure the efficiency of the search process in number of search path hops. Additionally, we present the average size of the index lists of nodes. Overall, the search path lengths for the specific experiments were competitive for the  $D$  and the  $E$  distance metrics. Thus, we present the average of the results for the two metrics.

Table 1 summarizes the experimental results for random queries from any source to any target area. The experiments are performed for a total number of nodes  $N = 1000, 4000, 10000$  that form overlays of various densities depending on the size  $n$  of the square grid. A general outcome based on the results of Table 1 is that indexing the 4 orthonormal semi-axes is much more efficient than if indexing the 4 quadrant diameters. In Section 4.1 we proposed the indexing of the quadrant diameters since these are the longer directions in a quadrant and offer the possibility for indexing numerous and more distant areas. Table 2 shows that actually this goal is achieved: the average index size is greater in all cases if indexing is performed on the quadrant diameters than on the orthonormal semi-axes. Surprisingly, the bigger indices do not diminish the average search path length. Moreover, the average search path length is shorter if indexing the orthonormal semi-axes than the quadrant diameters. The reason is not so obvious and we can track it in the following cases of search: assume that a search for a target area  $T$  is on an area  $S$ , and  $S$  and  $T$  are on almost the same column/row; since areas index

**Table 1.** Results for the search path length in number of hops for random distance values of source and target search areas

Metric & Setting	1000 32	1000 64	1000 128	4000 64	4000 128	10000 128
<b>4axes</b>	2,867	3,668	3,823	3,548	4,424	4,496
<b>4diameters</b>	2,917	3,744	3,840	4,016	4,770	5,445
<b>8axes</b>	2,863	3,775	3,521	3,850	4,099	4,258

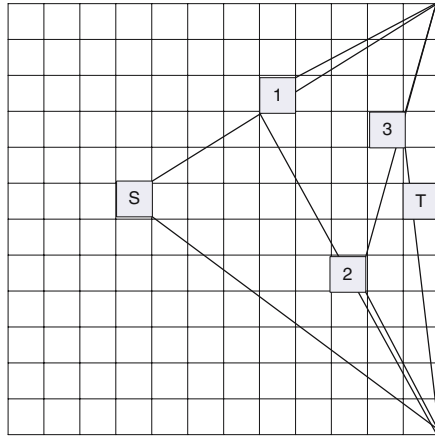
**Table 2.** Results for the index size

Metric & Setting	1000 32	1000 64	1000 128	4000 64	4000 128	10000 128
<b>4axes</b>	17,580	18,585	18,765	22,320	23,235	24,830
<b>4diameters</b>	18,259	20,253	20,738	22,755	23,560	25,420
<b>8axes</b>	17,330	16,662	18,810	20,367	22,736	24,985

the diameters, there is high possibility that  $S$  will send the search process to an area  $A_i$  that is farther away from  $T$  than itself in terms of columns/rows. Area  $A_i$  is closer to  $T$  in terms of rows/columns than  $S$ , meaning that the diameters that it indexes are farther away from  $T$  than the diameters indexed by  $S$ . This results in search hops on areas that are on rows/columns that are higher and lower in the grid interchangeably; this means that, even though  $S$  and  $T$  are actually close in terms of rows/columns the search reciprocates along these areas and results in long paths. Figure 10 shows an example of this deficiency of indexing the quadrant diameter. As it is apparent from the results in Table 1, these situations occur more often for large grids, since search may target very distant areas. Nevertheless, the indexing of the quadrant diameters performs better for sparse overlays, where it achieves even indexing along any direction in a quadrant. In this case, it can even outperform indexing of the orthonormal semi-axes, as the results show for the employment of the  $D$  metric on  $64 \times 64$  and  $128 \times 128$  grids and  $N = 1000$  areas.

In the study we have also considered indexing 8 semi-axes, namely, the orthonormal semi-axes and the quadrant diagonals. The results are better for the setting with 8 indexing semi-axes than in the normal setting almost in all cases. Specifically, the indexing of 8 semi-axes achieves a better performance for big and dense overlays (for example  $N = 4000$  for a  $64 \times 64$  grid as well as  $N = 10000$  for a  $128 \times 128$  grid. In these cases indexing 8 semi-axes instead of 4 results in more even indexing towards any direction in space. Furthermore, the results show that the setting with 8 indexing semi-axes has a better performance for sparse or medium dense than very dense overlays. Obviously, as the overlay becomes denser, indexing 4 semi-axes becomes more even and the possibility for a search process to be propagated on the indexing axis of the target area of search increases; thus, even though the setting with 8 indexing semi-axes provides larger indices they are not so useful.





**Fig. 10.** The search from  $S$  to  $T$  can hop on areas that are on rows/columns that are higher and lower in the grid interchangeably; this means that, even though  $S$  and  $T$  are actually close in terms of rows/columns the search reciprocates along these areas and results in long paths

Furthermore, we compared the setting of indexing 4 or 8 semi-axes for searches between areas that are close in terms of rows/columns. Actually, we have ranged the distance metric  $d_1$  between source and target search areas in values 1 – 17 for a very dense ( $N = 1000$ )  $32 \times 32$  grid. Thus, the maximum vertical distance of the source and target areas is ranged from the distance of adjacent cells to half the size of the grid. Table 3 shows the results of these experiments. It is apparent that the setting of indexing 8 semi-axes achieves shorter search paths than that of indexing 4 semi-axes for very close areas. The reason is that very close areas have more links among each other in the first than in the second case (for example for  $d_1 = 1$ , all areas are successors in the first case). However, the setting with 4 indexing semi-axes performs better for close enough but not to close areas. The reason is that in these situations search is benefited by dense indexing on fewer axes than sparse indexing on more axes. The indexing of 8 semi-axes increases its performance for quite distant source and target search areas, thus for  $d_1 > 1$ .

**Table 3.** Results for the index size and the search path length in number of hops for limited distance values of source and target search areas

Metric & Setting	N = 1024 for a $32 \times 32$ grid					avg index size
	$d_1 = 1$	$d_1 = 2$	$d_1 = 3$	$d_1 = 5$	$d_1 = 7$	
4axes	0,344	0,791	1,256	1,495	1,884	
8axes	0	0,567	0,983	1,318	1,715	
	$d_1 = 9$	$d_1 = 11$	$d_1 = 13$	$d_1 = 15$	$d_1 = 17$	
4axes	1,887	2,151	2,349	2,453	2,439	17,6
8axes	2,024	2,227	2,389	2,442	2,345	18,45

## 5 Related Work

As discussed in Section 1 one application of the proposed framework is the handling of spatial data in a peer-to-peer environments. There are some attempts in this field that propose algorithms for the distribution of multidimensional content data in general. The works such as SCRAP [4], MAAN [2], CISS [9], SQUID [13] and SkipIndex [15], propose partitioning the data using a space-filling curve and indexing them using a one-dimensional DHT method. Based on SCRAP MURK [4] and furthermore ProBE [12] are proposed. The latter is a system that supports range queries on multidimensional data, focusing on load balancing: virtual peers that are used to transfer content from one neighbor to another. The whole setting is fundamentally static, so that not much can be done for dynamic content reallocation.

Another approach to the problem is to construct distributed indices for multidimensional data. Existing works towards this direction try to distribute traditional multidimensional indices in P2P networks. The first noticeable such work is [5] that proposes the distribution of a centralized Quad-tree index to the peers; however the root bottleneck problem is statically solved. Recently, in [6] space is partitioned and indexed using a tree-like centralized index, such as R-tree, M-tree. Then, tree nodes (both inner nodes and leaves) are distributed in overlay peers. Peers are linked such that ancestors are neighbors with their descendants. Beyond vertical indexing, indexing is also performed horizontally in the tree. Yet, locality of data and locality-driven search are not considered in this work.

Beyond, peer-to-peer applications our framework can be applied to spatial overlays where search and linking can be performed based solely on nodes location. Related works to this aspect of the framework are those that consider positioned-based routing [10,3]. However, in this field the interest of the solutions is how to perform routing in arbitrary formed networks, often focusing on mobility or power-save for sensor networks. Finally, the work in [1] discusses geographical routing but assumes an overlay network linked based on triangulation.

## 6 Conclusions and Future Work

In this work we study the construction of a framework that can handle in a totally distributed manner spatial information. Specifically, we consider autonomous sites that are bound to spatial areas and that form an overlay network; we investigate the parameters of such a distributed system in order to perform search guided by locality and directionality. We have presented the main parameters of the framework and we have proposed appropriate values for them. One of the main parameters is the distance metric that is used in order to define locality in space. The theoretical discussion proves that a metric that defines square equidistant frontiers is more appropriate than one that defines circular frontiers. Also we have considered the indexing of remote areas and we have proposed solutions for this issue that were tested experimentally. Our next goal is to perform a thorough experimental study in order to confirm the theoretical results about the distance metrics. Furthermore, we will to extend the considered framework for areas of different sizes.

## References

1. Araujo, F., Rodrigues, L.: Geopeer: A location-aware peer-to-peer system (2004)
2. Cai, M., Frank, M., Chen, J., Szekely, P.: Maan: A multiattribute addressable network for grid information services. In: Proc. of Grid (2003)
3. Contla, P.A., Stojmenovic, M.: Estimating hop counts in position based routing schemes for ad hoc networks. *Telecommunication Systems* 22(1-4), 109–110 (2003)
4. Ganesan, P., Yang, B., Garcia-Molina, H.: One torus to rule them all: Multi-dimensional queries in p2p systems. In: Proc. of WebDB (2004)
5. Harwood, A., Tanin, E.: Hashing spatial content over peer-to-peer networks. In: Proc. of ATNAC (2003)
6. Jagadish, H.V., Ooi, B.C., Vu, Q.H., Zhang, R., Zhou, A.: VBI-tree: A peer-to-peer framework for supporting multi-dimensional indexing schemes. In: Proc. of ICDE (2005)
7. Kantere, V.: Query management in P2P overlays. PhD thesis, National Technical University of Athens, School of Electrical and Computer Engineering, Forthcoming work
8. Kantere, V., Sellis, T., Skiadopoulos, S.: Storing and indexing spatial data in p2p systems (Submitted for publication)  
<http://www.dblab.ece.ntua.gr/~verena/spatialp2p.pdf>
9. Lee, J., Lee, H., Kang, S., Choe, S., Song, J.: CISS: An efficient object clustering framework for DHT-based peer-to-peer applications. In: Proc. of DBISP2P (2004)
10. Mauve, M., Widmer, J., Hartenstein, H.: A survey on position-based routing in mobile ad-hoc networks. *IEEE Network Magazine*, pp. 30–39 (November 2001)
11. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. In: Proc. of SIGCOMM (2001)
12. Sahin, O.D., Antony, S., Agrawal, D., Abbadi, A.E.: PRoBe: Multi-dimensional range queries in p2p networks. In: Proc. of WiSe (2005)
13. Schmidt, C., Parashar, M.: Flexible information discovery in decentralized distributed systems. In: Proc. of HPDC (2003)
14. Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. of SIGCOMM (2001)
15. Zhang, C., Krishnamurthy, A., Wang, R.Y.: Skipindex: Towards a scalable peer-to-peer index service for high dimensional data. Technical Report (TR-703-04) (2004)

# Distributed, Concurrent Range Monitoring of Spatial-Network Constrained Mobile Objects

Hua Lu<sup>1,2</sup>, Zhiyong Huang<sup>1,3</sup>, Christian S. Jensen<sup>2</sup>, and Linhao Xu<sup>1</sup>

<sup>1</sup> School of Computing, National University of Singapore, Singapore

<sup>2</sup> Department of Computer Science, Aalborg University, Denmark

<sup>3</sup> Institute for Infocomm Research, Singapore

**Abstract.** The ability to continuously monitor the positions of mobile objects is important in many applications. While most past work has been set in Euclidean spaces, the mobile objects relevant in many applications are constrained to spatial networks. This paper addresses the problem of range monitoring of mobile objects in this setting, in which network distance is concerned. An architecture is proposed where the mobile clients and a central server share computation, the objective being to obtain scalability by utilizing the capabilities of the clients. The clients issue location reports to the server, which is in charge of data storing and query processing. The server associates each range monitoring query with the network-edge portions it covers. This enables incremental maintenance of each query, and it also enables shared maintenance of concurrent queries by identifying the overlaps among such queries. The mobile clients contribute to the query processing by encapsulating their host edge portion identifiers in their reports to the server. Extensive empirical studies indicate that the paper's proposal is efficient and scalable, in terms of both query load and moving-object load.

## 1 Introduction

In the management of moving objects, continuous monitoring queries have recently gained attention, as they provide the fundamental support to different mobile services, including services that monitor traffic at intersections, services that monitor fleets of vehicles such as buses or police cars, and a variety of services that monitor sensitive regions. Existing work on continuous monitoring queries has predominantly assumed that the moving objects are embedded into two-dimensional Euclidean space, and has relied on Euclidean distance as the relevant notion of distance. However, in many application scenarios, including the ones mentioned above, the moving objects are constrained to a spatial network, typically a road network. In this setting, Euclidean distance is not the relevant notion of distance—rather, network distance is.

In a spatial network setting, the ability to efficiently monitor the part of a network within a certain network distance of a query point for moving objects constitutes fundamental functionality for many mobile services. This paper proposes efficient techniques for such continuous range monitoring queries in spatial networks, which we term  $CRMQ_N$  queries. Figure 1 exemplifies a  $CRMQ_N$  query. In the partial road network displayed in the figure, a  $CRMQ_N$  query  $q$  is issued with the position represented by a small square and with a 5 km distance of interest. The dashed circle identifies the

range determined by the Euclidean distance, while the arrows with short bars identify the sub-network within network distance 5 km of the query point. Therefore, for the time point captured in the figure, moving objects *A*, *B*, *C*, and *D* belong to the result; object *E* does not, as the network path from query point *q* to *E* exceeds the query range. Note that a moving object’s membership in the query result generally changes as time elapses due to the object’s movements. The range monitoring query continuously maintains the correct query result as time elapses.

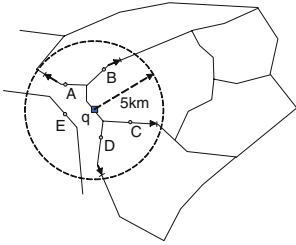


Fig. 1. CRMQ<sub>N</sub> query example

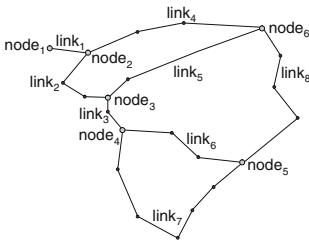


Fig. 2. Road network example

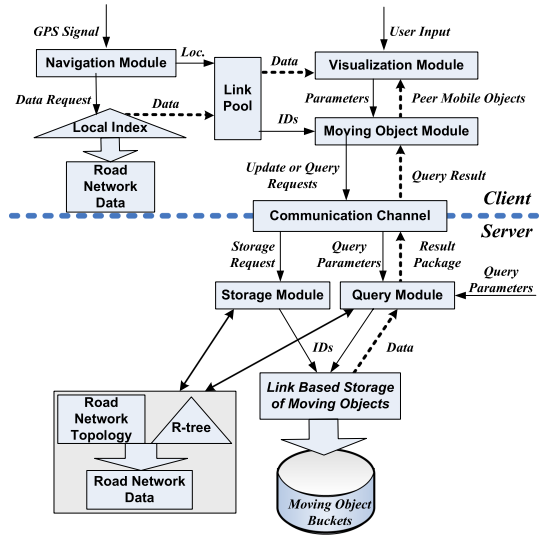


Fig. 3. System framework

We assume a client/server system architecture, in which the moving objects (clients) report their location information to the central server that is responsible for storing the locations of the moving objects and for processing the range monitoring queries. As the moving objects continually send their location reports to the server, the server must update the results of the active queries to maintain the query results. We also assume that query processing occurs in main memory, as do most online monitoring systems.

For each query, we initially use a network-expansion-based algorithm for identifying those network links that are either fully or partially covered by the query. Then in the subsequent query processing, all other ones are ignored as they have no impact on the query result. A *network link* (formally defined in Section 2.1) is a maximum part of a spatial network for which no exchange of traffic is possible .

For each link fully covered by the query, we report every object moving on it in the query result; for each link partially covered by the query, additional refinement based on the network distance is performed to discard non-qualifying objects. This identification

of different link types enables us to reduce query processing cost by expending different amounts of effort on different types.

The identification of the links covered by each range monitoring query also discloses the overlaps among concurrent queries. In other words, different queries may cover the same link(s), although they have different query points and ranges. By exploiting such overlaps, we develop a shared maintenance mechanism for concurrent queries. This mechanism comprises particular data structures and relevant algorithms to efficiently update the results of concurrent queries.

In addition, we employ a novel client design that enables the mobile terminals to contribute their computing capabilities to the continuous query maintenance, thus reducing the server side computation. In particular, the host links of a moving object (the road link on which it is moving) is easily determined on client side, and its identifier is then encapsulated in the location report sent to the server, which uses this to facilitate the shared maintenance of concurrent queries.

The paper's key contributions are fourfold. First, we identify all links relevant to a range monitoring query, differentiating between those links that are fully covered versus partially covered by the query. This minimizes the subsequent query maintenance costs, and it also reveals the overlaps among concurrent queries, thus enabling a shared maintenance of these queries. Second, as realistic scenarios entail large numbers of concurrent queries, it is essential to be able to maintain multiple query results correctly and efficiently. We achieve this by means of shared maintenance based on the identification of relevant links. We propose hash-based structures with efficient processing algorithms for this purpose. Third, in link based query processing, the host links on which the mobile objects reside must be identified. To offload the server, we delegate the host link identification to the moving objects. This is enabled via a novel client design that adapts the terminal's navigation software module to determine the host link. Fourth, we report on extensive empirical studies that characterize the efficiency and scalability of our proposal.

In Section 2, we proceed to present some preliminaries. Section 3 describes the specific data management on the mobile terminals. Section 4 details the server-side design for concurrent continuous range monitoring queries of network constrained mobile objects. Section 5 experimentally evaluates the paper's proposals. Section 6 briefly reviews related work, and Section 7 concludes this paper.

## 2 Preliminaries

### 2.1 Data Model

Similarly to other proposals (e.g., [3]), we model a road network as a particular kind of spatial network that captures both the graph aspect of a road network and also captures the embedding of a road network into geographical space. Thus, a spatial network  $\mathcal{SN} = (\mathcal{N}, \mathcal{L})$  consists of a set of nodes  $\mathcal{N} = \{n_1, n_2, \dots, n_N\}$  and a set of links  $\mathcal{L} = \{l_1, l_2, \dots, l_L\}$ . A link consists of a pair of elements from  $\mathcal{N}$ , and a total function  $weight : \mathcal{L} \rightarrow \mathbb{R}^+$  assigns a weight to each link.

The embedding into geographical space is accomplished with two total functions. First,  $pos_{\mathcal{N}} : \mathcal{N} \rightarrow \mathbb{R}^2$  maps each node to a position in two-dimensional Euclidean

space. Second,  $pos_{\mathcal{L}} : \mathcal{L} \rightarrow \{(pos_n(n_s), p_1, \dots, p_k, pos_n(n_e)) \mid n_s, n_e \in \mathcal{N} \wedge p_1, \dots, p_k \in \mathbb{R}^2 \wedge k \geq 0\}$  maps each link to a polyline. A link  $l = (n_s, n_e)$  is then mapped to a polyline with the positions of  $n_s$  and  $n_e$  as its delimiting positions. The  $k$  positions in-between these two are termed intermediate points. Consequently, a link is modeled by a polyline consisting of  $k + 1$  line segments.

We will assume that the weight of a link is the length of the associated polyline. Next we assume a function  $d_N$  that takes any two positions on polylines of a spatial network as arguments and returns the (shortest) network distance between these. Such a function can be defined in straightforward fashion. We will also assume that links model bidirectional roads.

An example of road network is shown in Figure 2. Nodes are given by big dots and intermediate points by small dots. The road network encompasses 8 links, one of which does not contain any intermediate points (i.e.,  $k = 0$ ). Link 7 has the largest number of intermediate points ( $k = 5$ ).

We assume a set  $\mathcal{M}$  of moving objects that are constrained to the spatial network. Thus, a moving object at any point in time resides on a link in the network, and its position intersects with the polyline that represents the link. Function  $pos_{\mathcal{M}}$  maps an object to its current position as known by the server.

## 2.2 Problem Statement

We assume a spatial network  $\mathcal{SN}$ , a set of moving objects  $\mathcal{M}$  constrained to this network, and a network distance function  $d_N$ . Moving objects continually issue updates to the server, thus updating function  $pos_{\mathcal{M}}$ .

A *continuous network-distance-based range monitoring query*  $\mathcal{R}$  takes  $\mathcal{M}$  as argument and accepts two parameters:  $(p, r)$ , where  $p$  is the (stationary) query point and  $r$  is the network query range. Such a query, termed *CRMQ<sub>N</sub>*, is activated at some time  $t_s$ , the start time of the query, and it is terminated at some later time  $t_e$ , the end time of the query. The query result is maintained from time  $t_s$  to time  $t_e$ .

$$\forall t \in [t_s; t_e] \quad (o \in \mathcal{R}[p, r](\mathcal{M}) \Leftrightarrow o \in \mathcal{M} \wedge d_N(pos_{\mathcal{M}}(o), p) \leq r)$$

The problem addressed in this paper is that of providing a complete set of techniques that enable the correct and efficient maintenance of multiple such range monitoring queries.

## 2.3 Distributed System Architecture

Our proposal is based on a client/server architecture, in which the clients are moving objects equipped with mobile terminals and a central data server is in charge of the query processing. The clients communicate with the server via some form of wireless network, e.g., a 2.5 or 3G cellular network. The system framework is shown in Figure 3.

At the highest level of abstraction, a mobile terminal consists of three modules. A navigation module is responsible for retrieving spatial data that represents the object's current location; this is obtained from positioning hardware such as a GPS receiver. A visualization module is responsible for displaying the object's location and results of queries on a background map on the terminal's screen; and it is responsible for passing

user input as query parameters to the moving object module. A moving object module issues update and query requests to the server, and passes query results back to the visualization module. The spatial network is organized in files and indexed by a composite structure that includes a compact R-tree and sequential indexes. The spatial data in use is maintained in a pool that is shared between the different modules. The data management on the clients is detailed in Section 3.

The important modules on the server are the storage and query modules. The former receives location reports from the moving objects and is responsible for storing this moving object information. The query module is responsible for processing queries issued by either the server itself or a moving object. Both modules access the moving objects via a network-link-based index, which also refers to road network data. The server-side data management is covered in Section 4.

### 3 Client-Side Data Management

Without loss of generality, we assume a client setting that resembles a GPS-enabled smartphone, e.g., a pocket PC. These have relatively limited flash storage and no disk, and they are currently being sold with navigation software. As they may obtain power from the vehicles in which they are installed, we do not consider power issues.

This section first describes the compact client-side data structures used for spatial networks, and then presents the client-side functionality that utilizes these structures.

#### 3.1 Client-Side Index Structure

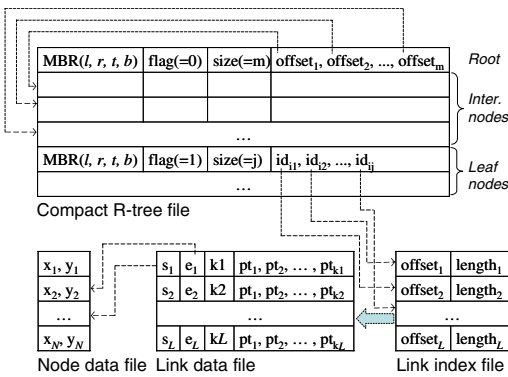


Fig. 4. Index structure on device

We store the nodes and links of a spatial network in two separate sequential data files. Due to the relatively limited storage available on a mobile terminal, it is not feasible to maintain all records from these files in main memory. For navigation purposes, we need to efficiently retrieve the data from the two files that relate to a region of interest. To enable this, we need a proper index structure. In particular, to support the range queries required for navigation and data display, we adapt the R-tree [10] into a compact structure suitable for a mobile environment.

The client-side index structure, comprising the compact R-tree and the two sequential files, and an additional index file are illustrated in Figure 4 and explained next.

The sequential file of nodes stores the positions of the nodes, which are ordered according to their identifiers. Similarly, the sequential file of links stores for each link the two delimiting points of its polyline, as pointers to the node file; it stores the number of intermediate points; and it stores the coordinates of the intermediate points.



An R-tree is created on the set of polylines representing all the links in the spatial network. Each polyline is a unit to index. All non-leaf nodes are organized as in a standard R-tree. Each leaf node holds the identifiers of all the network links it covers, rather than pointers to the real link records in the link file. As link records are not of equal length, a dense sequential index file on the link file is used. To access the record of the link with identifier  $l_i$ , the  $i$ th record in the sequential index is retrieved. This record contains the offset and content length of the link record in the link file. By using a separate sequential index file, we retain the road link identifiers. This facilitates potential identifier-based random data access on the client side.

The R-tree is mapped to a sequential file in a specific way. All nodes are stored in a breadth-first order. Each non-leaf node is stored according to the format  $\langle MBR, flag, size, offsets \rangle$ , where  $MBR$  is the minimum bounding rectangle of a network link,  $flag$  indicates whether the node is a leaf node,  $size$  contains the number of child nodes, and  $offsets$  are offsets into the file that point to the child nodes. The leaf-node format is similar, except that the field  $offsets$  is replaced by a field  $ids$  that stores the integer identifiers of the network links covered by the leaf node; these identifiers correspond to row numbers in the sequential index on the link file. To conserve storage space, we allocate bits to the node fields in a compact way according to the ranges of parameters such as R-tree fanout, R-tree height, and the number of links in the road network.

### 3.2 Client-Side Navigation and Visualization

The navigation module receives so-called NMEA sentences with location data from the GPS receiver, retrieves pertinent link records via the data structures detailed in the previous section, and places the link records in a link pool that is used by the visualization module for display of part of the map containing the object's current location.

Initially, given the current location  $(x_c, y_c)$  of the object, the data within rectangle  $(x_c - w, y_c + h, x_c + w, y_c - h)$  (left, top, right, bottom coordinates, as for bounding rectangles) is to be displayed on the terminal's screen, where  $w$  and  $h$  are configurable parameters. This rectangle is called the *active window*, and a window query against the data structure is used for retrieving the relevant data. Every link in the pool is of format  $\langle id, MBR, points \rangle$ , where  $id$  is the link's identifier,  $MBR$  is its minimum bounding rectangle, and  $points$  is the sequence of coordinates defining the link polyline.

When receiving a new position from the GPS receiver, the navigation module first checks whether the current location remains in the active window. If so, no new data retrieval is needed. Otherwise, a new active window  $win_{new}$  is computed, those currently pooled links that are outside  $win_{new}$  are discarded, and those links that intersect  $win_{new}$ , but are not in the pool, are retrieved via the data structure. This retrieval is achieved by a modified window query that uses two windows  $win_{new}$  and  $win_{old}$ . It returns the links that intersect with  $win_{new}$ , but do not intersect with  $win_{old}$ . Standard depth-first or best-first R-tree traversals can be adapted to process such modified window queries. Upon the retrieval, the active window is set to  $win_{new}$ . During the data retrieval, the network link and line segment on which the current location  $(x_c, y_c)$  resides are determined on the fly. We call this link (segment) the handset client's *host link* (*host segment*) with respect to its current location. If no new data is to be retrieved, the host link and segment are determined by the moving object module, covered next.

### 3.3 The Moving Object Module

The moving object module is responsible for ensuring that the server has up-to-date location information for the object. It continually receives location data from the navigation module, and maintains a record of the form  $\langle loc_c, vel_c, t_c \rangle$  that captures the current location, velocity and the time of those for the object. It also maintains a record of the form  $\langle lnk_h, seg_h \rangle$  that captures the host link and segment that correspond to the current location.

If the navigation module has not performed data retrieval for the most recent location, an update without a known host link and segment is sent to the moving object module. The relevant algorithm is shown in Figure 5. First, it is checked whether the new location is still on the currently recorded host segment. If so and if the velocity has changed markedly, an update message UPT is issued to the server and the records are updated (lines 2–4). A pre-specified threshold  $\Delta v$  of velocity change is used to judge whether an update to the server is necessary (line 2). For such an update involving the same host link, the UPT message includes the location, host link identifier and time. The server will use the host link identifier to efficiently locate the record corresponding to the moving object. If the object is not on the recorded host segment, the adjacent segments on the recorded host link are checked. The check is carried out sequentially in two directions one after the other: first from the segment after  $seg_h$  to the last one; then from the one before  $seg_h$  to the first one (line 6). When a new host segment is found for the location, an update message is sent to the server, the records are updated, and the algorithm terminates (lines 8–10). If the object is no longer on the recorded link, a new host link and segment are found by searching the link pool (lines 11–13). An update message is sent, the records are updated, and the algorithm stops (lines 14–16).

---

**Algorithm** `updateWithoutHosts`( $loc_n, vel_n, t_n$ )

**Input:**  $loc_n$  is the new location;  $vel_n$  is the new velocity;  $t_n$  is the report time

// Determine the host link and segment for  $loc_n$

1. **if** ( $loc_n$  is still on  $seg_h$ )
  2.     **if** ( $|vel_n - vel_c| > \Delta v$ ) // Marked velocity change
  3.         Send UPT( $oid, loc_n, lnk_h, t_n$ ) to the server;
  4.          $loc_c = loc_n$ ;  $vel_c = vel_n$ ;  $t_c = t_n$ ;
  5.     **else**
  6.         **for each** segment  $seg_i$  after/before  $seg_h$  of link  $lnk_h$
  7.             **if** ( $loc_n$  is on  $seg_i$ )
  8.                 Send UPT( $oid, loc_n, lnk_h, t_n$ ) to the server;
  9.                  $loc_c = loc_n$ ;  $vel_c = vel_n$ ;  $seg_h = seg_i$ ;  $t_c = t_n$ ;
  10.             **return**;
  11.     **for each** link  $lnk_i \neq lnk_h$  in pool
  12.         **for each** segment  $seg_i$  of link  $lnk_i$
  13.             **if** ( $loc_n$  is on  $seg_i$ ) // Moved to another link
  14.                 Send UPT( $oid, loc_n, lnk_h, lnk_i, t_n$ ) to the server;
  15.                  $loc_c = loc_n$ ;  $vel_c = vel_n$ ;  $lnk_h = lnk_i$ ;  $seg_h = seg_i$ ;  $t_c = t_n$ ;
  16.             **return**;
- 

**Fig. 5.** Update without known hosts

If the navigation module has just invoked a data retrieval for the new location, an update attached with the new host link and segment identifiers is sent to the moving object module. If the update is the first positioning report received, a registration message REG is sent to the server and the report is recorded. Otherwise, the process is similar to that of the previous algorithm, by distinguishing among three cases.

## 4 Server-Side Data Management

We first present how mobile objects are organized on the server side in accordance with the client design presented in Section 3. Then, we describe how concurrent continuous range monitoring queries are processed on the server side.

### 4.1 Server-Side Mobile Object Management

**Link-Based Moving Object Indexing.** On the server side, a hashing mechanism is used for indexing the moving objects according to their current network locations. The composite index structure is shown in Figure 6 and is explained next.

Recall that the  $L$  links in the spatial network are assigned integer identifiers from 1 to  $L$ . The top-level index is simply a sequential file with one entry for each link. With the same link index file as on the client side, a link record can be easily fetched given its identifier. Each link entry contains a pointer to a moving-object bucket, which keeps all objects currently moving on that link. The identifier of a link is also used to locate the polyline associated with the link and the link's topology information. All links in polylines are organized in a sequential link file, as on the client side. Via a link index file, the link record is easy to fetch. Node topology information is organized in a separate file. Each node topology entry in this file contains the count of its incoming links, the count of its outgoing links, and the identifiers of these two types of links sequentially. Link records and node topology information are associated via a node index file, which for each node stores its offset into the topology file. To speed up spatial operations needed in query processing, all network links are indexed by an R-tree, and so are all network nodes. These R-trees are not illustrated in Figure 6. Unlike previous indexes for disk-resident network constrained moving objects, our indexing solution does not employ R-trees in a hierarchical fashion [8,16] or index the trajectories of moving objects [2].

Each element of an object bucket has the format  $\langle oid, loc, time \rangle$ , where  $oid$  is an object identifier,  $loc$  is the most recently reported location, and  $time$  is the time of the most recent report. To simplify the processing, we allocate  $oid$ 's starting at 1. The hashing of a moving object works as follows. Given a moving object  $oid$ , its host link  $lnk$  is determined by its location  $loc$ . Identifier  $lnk$  is used to obtain the bucket pointer  $bucket\_ptr$  in the corresponding link index entry. The object  $oid$ 's record is kept in the object bucket pointed by  $bucket\_ptr$ . The hashing for an object is created or updated when the server receives an update report from that object, explained next.

**Insertion.** An insertion occurs when a moving object issues a REG( $oid, loc, lnk, time$ ) message to the server. An insertion is quite straightforward. The field  $lnk$  is first used together with the index structure to locate the bucket into which the moving object should be inserted; then an object entry is created and inserted into that bucket.

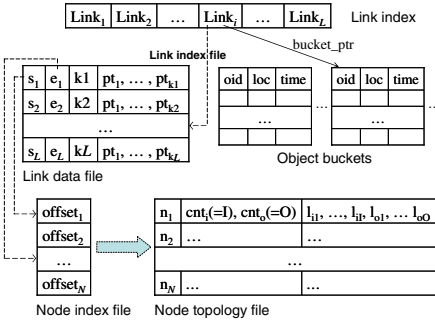


Fig. 6. Server-side index structure

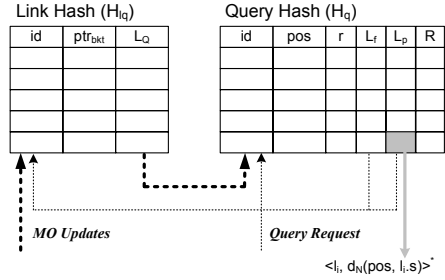


Fig. 7. Data structures and query execution

**Deletion.** An object may delete itself from the system. This can occur when a mobile terminal is switched off, e.g., the user’s vehicle is parked. In this case, the terminal issues a delete message prior to powering down. A deletion is performed on the server when a  $DEL(oid, lmk)$  message is received from an object. Upon receiving this message, the server removes the relevant object from the database. Field  $lmk$  is used to locate the moving object bucket via the index structure, and then the bucket is scanned to remove the element in the bucket with identifier  $oid$ . As an option, by periodically checking object buckets we are able to identify objects that have been inactive for a long time, and delete them from the database.

**Update.** Updates occur when the server received an UPT message from an object. As mentioned in Section 3.3, there are two types of UPT messages. For an  $UPT(oid, loc, lmk, time)$  message, which indicates that the object remains on the recorded host link, the object’s entry is found, and the relevant record fields  $loc$  and  $time$  are updated. In case of an  $UPT(oid, loc, lmk, lmk_n, time)$  message, which indicates that the object has moved out of the recorded host link, a deletion and an insertion are performed. Field  $lmk$  is used to locate the current bucket for the deletion, while  $lmk_n$  is used to find the new bucket for the insertion.

After each index operation is finished, the query maintenance function will be invoked to correctly adjust query results accordingly. This is discussed in Section 4.4.

**Discussion.** The server-side index structure has several important properties. First, its overall performance is balanced. If a network link contains few moving objects, index maintenance is inexpensive. If the number of moving objects on a link is large, the query processing is less expensive. This balance between index maintenance and query processing costs is hard to achieve in moving object indexes [15]. Second, the index structure supports easy and high concurrency as it separates moving objects on different network links. Within each link, concurrency control on a hash table is much easier compared to that on any tree index. Third, the index structure efficiently supports queries based on network distance, which has not been addressed well in most previous work on the indexing of moving objects. We do not organize all moving objects in a global hash table, because a single large hash table is much more difficult

to manage in terms of scalability and conflict rate, and it does not facilitate our query processing.

## 4.2 Solution Overview and Data Structures for Concurrent Queries

In a scenario with monitoring queries, multiple such queries will typically be active at a given point in time. We thus proceed to employ the *shared execution* [13,18] philosophy to process concurrent queries. As the movements of all objects in  $\mathcal{M}$  are constrained to the spatial network, only specific network links are relevant for each  $CRMQ_N$  query. Further, the relevant links of different, concurrent  $CRMQ_N$  queries may intersect. These observations provide the foundation for our shared query processing solution for  $CRMQ_N$  queries.

Our solution identifies network links that are covered by multiple queries, organizes the relevant information together with the query results in two hash tables, and updates the hash table contents, including the query results, accordingly upon receiving moving object updates. The data structures and query execution are illustrated in Figure 7.

Given a query, those links that are relevant for the query, termed its *sensitive links*, are identified and classified into two categories: those completely within the query region, termed fully sensitive, and those that merely intersect with the query region, termed partially sensitive. The former links are kept in a list  $L_f$ , and the latter links are kept in a list in  $L_p$ . Then the sensitive links are searched for moving objects within the query region. For each fully sensitive link, the search simply retrieves all objects in its bucket. For each partially sensitive link, its moving-object bucket is searched to retrieve those objects whose network distance to the query point is within the query range.

We use a link-to-query hash table named  $H_{lq}$  that maps link identifiers to lists of queries for which the link is a sensitive link: for a link  $l_i$  the list  $L_{Qi}$  thus contains the queries for which  $l_i$  is sensitive. The query identifiers start from 1. For each query that has  $l_i$  as a fully sensitive link, its identifier is simply stored in list  $L_{Qi}$ . For each query that has  $l_i$  as a partially sensitive link, its identifier is stored in  $L_{Qi}$  with a negative sign. Additionally, a pointer to the bucket of all objects currently moving on  $l_i$  is stored in each element of  $H_{lq}$ .

All queries are stored in a hash table  $H_q$  that maps a query identifier  $q_i$  to an entry consisting of five relevant elements: the query point  $pos$ , the network query range  $r$ , the fully sensitive links  $L_f$ , the partially sensitive links  $L_p$ , and the current query result  $R$ . For each partially sensitive link  $l_i$ ,  $L_p$  maintains a mapping of  $l_i$  to the network distance from the query point  $pos$  to the link's start or end node. If the link is partially covered by the query from its end node, the distance is attached with a negative sign. Otherwise the original distance value is used. This facilitates the network distance computation during query processing.

For continuous maintenance, the link identifiers enclosed in the update messages from the moving objects are used for accessing hash table  $H_{lq}$ . This way, the identifiers of all relevant queries, whose results may be affected by the update, are found. These identifiers are in turn used to explore hash table  $H_q$ , making it possible to update the results of multiple queries in a shared fashion. To uninstall a query  $q$ , the identifiers of its sensitive links, as stored in  $L_f$  and  $L_p$  in hash table  $H_q$ , are retrieved. These are

---

**Algorithm monitorInit**( $pos, r, id$ )

**Input:**  $pos$  is the query point;  $r$  is the network query range;  $id$  is the query identifier

**Output:** the initial result

```

1.  $R = \emptyset; Q = \emptyset;$ 
   // Determine starting links
2. if ( $isNode(pos)$ )
3.    $Q.enqueue(\langle node(pos), 0 \rangle)$ 
4. else
5.    $l_q = find\_host(pos);$ 
6.   if ( $d_N(pos, l_q.s) < r$ )  $Q.enqueue(\langle l_q.s, d_N(pos, l_q.s) \rangle);$ 
7.   if ( $d_N(pos, l_q.e) < r$ )  $Q.enqueue(\langle l_q.e, d_N(pos, l_q.e) \rangle);$ 
8.   if ( $(d_N(pos, l_q.s) \leq r)$  and ( $d_N(pos, l_q.e) \leq r$ ))
9.      $R = R \cup l_q$ 's bucket; Add  $id$  to  $H_{l_q}(l_q).L_Q$ ; Add  $l_q$  to  $H_q(id).L_f$ ;
10.  else
11.     $R = R \cup search(l_q)$ ; Add  $-id$  to  $H_{l_q}(l_q).L_Q$ ; Add  $\langle l_q, +/ - d_N \rangle$  to  $H_q(id).L_p$ ;
   // Expansion, search for sensitive links
12. while ( $Q$  is not empty)
13.    $(n, d_N) = Q.pop()$ ;
14.   foreach unvisited link  $l_i$  connected to  $n$ 
15.     if ( $d_N + l_i.len > r$ ) // A partially sensitive link
16.        $R = R \cup search(l_i)$ ; Add  $-id$  to  $H_{l_i}(l_i).L_Q$ ;
       Add  $\langle l_i, +/ - (d_N + l_i.len) \rangle$  to  $H_q(id).L_p$ ;
17.     else // A fully sensitive link
18.        $R = R \cup l_i$ 's bucket; Add  $id$  to  $H_{l_i}(l_i).L_Q$ ; Add  $l_i$  to  $H_q(id).L_f$ ;
19.       if ( $d_N + l_i.len < r$ )  $Q.enqueue(\langle l_i.n \neq n, l_i.len \rangle);$ 
20. return  $R$ ;
```

---

**Fig. 8.** Initialization of a  $CRMQ_N$  query

then used for accessing hash table  $H_{l_q}$  from where the query identifier stored in relevant link's query list  $L_Q$  is removed.

### 4.3 Query Initialization

The initialization of a single range monitoring query identifies all its sensitive links via incremental network expansion [17], and it searches these (mainly the partially sensitive ones) to determine the initial result. The algorithm, shown in Figure 8, does a network expansion from the query point  $pos$  and places all nodes within the network query range  $r$  in a priority queue  $Q$ . This  $Q$  gives priority to those nodes with shorter network distances to  $pos$ , and supports network expansion to identify all sensitive links. If the query point  $pos$  coincides with the position of a node, as determined by the function  $isNode(pos)$  that searches the R-tree of all nodes, the node with a distance of 0 is pushed into queue  $Q$  (lines 2–3). If  $pos$  is not a node, its host link  $l_q$  is determined (line 5) by calling  $find\_host(pos)$ , which is implemented by searching the network link R-tree. Then  $l_q$ 's two nodes are checked to determine whether they are within distance  $r$  of  $pos$  (lines 6–7). If both of them are within  $r$ ,  $l_q$  is searched as a fully sensitive link and added to  $H_{l_q}$  and  $H_q(id).L_f$  (lines 8–9). Otherwise,  $l_q$  is searched as a partially sensitive

---

**Algorithm sharedMaintain**( $oid, loc, loc_n, lmk, lmk_n$ )

**Input:**  $oid$  is the client object's identifier

 $loc$  is the client object's old position;  $loc_n$  is the client object's new position

 $lmk$  is the client object's old host link;  $lmk_n$  is the client object's new host link

1. **if** ( $oid < 0$ ) // Object deletion
  2.     **foreach** query identifier  $i \in H_{lq}(lmk)$
  3.         **if** ( $(i > 0)$  **or** ( $oid \in H_q(-i)$ 's result)
  4.             Remove  $oid$  from  $H_q(|i|)$ 's result;
  5.     **else if** ( $loc_n == null$ ) // First report
  6.         **foreach** query identifier  $i \in H_{lq}(lmk)$
  7.             **if** ( $(i > 0)$  **or** ( $d_N(H_q(-i).pos, loc) \leq H_q(-i).r$ ))
  8.                 Add  $oid$  to  $H_q(|i|)$ 's result;
  9.     **else if** ( $lmk_n == null$ ) // Still on the old link
  10.         **foreach** query identifier  $i \in H_{lq}(lmk)$
  11.             **if** ( $i > 0$ ) **continue**;
  12.             **if** ( $oid \in H_q(-i)$ 's result)
  13.                 **if** ( $d_N(H_q(-i).pos, loc_n) > H_q(-i).r$ )
  14.                     Remove  $oid$  from  $H_q(-i)$ 's result;
  15.                 **else if** ( $d_N(H_q(-i).pos, loc_n) \leq H_q(-i).r$ )
  16.                     Add  $oid$  to  $H_q(-i)$ 's result;
  17.     **else** // Link change
  18.         **foreach** query identifier  $i \in H_{lq}(lmk)$
  19.             **if** ( $(i > 0)$  **or** ( $oid \in H_q(-i)$ 's result)
  20.                 Remove  $oid$  from  $H_q(|i|)$ 's result;
  21.         **foreach** query identifier  $i \in H_{lq}(lmk_n)$
  22.             **if** ( $i > 0$ ) // A fully sensitive link
  23.                 Add  $oid$  to  $H_q(i)$ 's result;
  24.             **else if** ( $d_N(H_q(-i).pos, loc_n) \leq H_q(-i).r$ )
  25.                 Add  $oid$  to  $H_q(-i)$ 's result;
- 

**Fig. 9.** Shared maintenance of concurrent queries

link and is added to  $H_{lq}$  (line 11). In the link-to-query hash table  $H_{lq}$ , fully sensitive and partially sensitive links are distinguished by the sign attached to their query identifiers. A partially sensitive link  $l_q$  is also added to  $H_q(id).L_p$  with a corresponding distance value (line 11). The distance value's sign is determined based on whether  $l_q$ 's start node or end node is met.

Next, network expansion is repeated to identify all sensitive links until all unvisited nodes are too far away from  $pos$  (lines 12–19). For each fully sensitive link, all objects on it enter the initial result (line 18). Each partially sensitive link needs to be searched for the objects that are really covered by the query range (line 16). Similar to the case for  $l_q$  above (lines 8–11), relevant operations are carried out on hash tables  $H_{lq}$  and  $H_q$ .

#### 4.4 Shared Concurrent-Query Maintenance

Shared maintenance of concurrent range monitoring queries occurs when the server receives a registration, deletion or update message from an object. The pseudo code for the shared maintenance mechanism is shown in Figure 9. The maintenance mechanism

---

**Algorithm queryUpdate**( $id, pos, L'_f, L'_p$ )

**Input:**  $id$  is the query identifier;  $pos$  is the new location of query point  
 $L'_f$  is the updated fully sensitive link list;  $L'_p$  is the updated partially sensitive link list

**Output:** the updated result

1. **foreach** link  $l_i \in H_q(id).L_f$
  2.   **if** ( $l_i \notin L'_f$ )
  3.     Remove  $l_i$  from  $H_q(id).L_f$ ; Remove  $l_i$ 's objects from  $H_q(id).R$ ;
  4.   **else** Remove  $l_i$  from  $L'_f$ ;
  5. **foreach** link  $l_i \in L'_f$
  6.   Add  $l_i$  to  $H_q(id).L_f$ ; Retrieve  $l_i$ 's objects to  $H_q(id).R$ ;
  7. **foreach** link  $l_i \in H_q(id).L_p$
  8.   **if** ( $l_i \notin L'_p$ )
  9.     Remove  $l_i$  from  $H_q(id).L_p$ ; Remove  $l_i$ 's objects from  $H_q(id).R$ ;
  10. **foreach** link  $l_i \in L'_p$
  11.   Add  $l_i$  to  $H_q(id).L_p$ ; Search  $l_i$  and add resultant objects to  $H_q(id).R$ ;
  12. **return**  $R$ ;
- 

**Fig. 10.** Support for moving queries

identifies all the relevant queries from  $H_{l_q}$  by hashing given link identifiers. If the message from an object is a deletion (indicated by a negative  $oid$  in line 1), the object is removed from the results of all relevant queries by means of the link-to-query hash table (line 2). Upon receiving a first-time report, if link  $lnk$  is a fully sensitive link of some query ( $i > 0$  in line 7) or the object is within the query range of  $pos$  on a partially sensitive link, the object is reported (line 8). For an update on the same link, any query having link  $lnk$  as a fully sensitive link is skipped (line 11) as no result change occurs, while any query having  $lnk$  as a partially sensitive link needs further checking (lines 12–16). For an update involving a link change, the object is removed from the results of current relevant queries if necessary (lines 18–20), and it is added to the results of new relevant queries if necessary (lines 21–25).

#### 4.5 Support for Moving Queries

A monitoring query may change its position, e.g., because it is attached to a moving object. We term this a moving query. Naturally, such queries can be supported by terminating the old query and initiating the new query when such an update in the position of a query happens. This approach is already supported by the proposals presented thus far. However, we can do better. With our hash-based data structures available, we do not need to re-evaluate the new, or updated, query from scratch.

We apply a simplified variation of algorithm **monitorInit** (Figure 8). This algorithm uses network expansion to identify the updated fully (partially) sensitive link list  $L'_f$  ( $L'_p$ ) with respect to the query's new query point without searching them. Then the update procedure shown in Figure 10 is invoked. Each expired fully sensitive link, together with those objects on it, is removed (lines 2–3); for any new fully sensitive link, objects on it are included in the result (lines 5–6). Each expired partially sensitive link, together with the objects on it, is removed (line 8–9). Each remaining/new partially sensitive link is searched, and the qualifying objects are included in the result (lines 10–11).



To support moving queries efficiently as detailed above, a minor change to the result ( $R$ ) data structure is needed. It must consist of a hash table that maps link identifiers to the lists of objects on the links. This facilitates the necessary link-based objects removal in the algorithm above.

## 5 Empirical Performance Study

We first study the server-side design and then consider the client side.

### 5.1 Server Side Experiments

**Settings.** To obtain good server-side performance, we have proposed two strategies: that of shifting part of the server load to the clients by letting them report host links; and that of processing concurrent queries in a shared manner. These two strategies are orthogonal. Therefore, by varying each of them, we obtain four different system schemes, as listed in Table 1. In the horizontal dimension, clients report either their locations (XY) or their host links (ID). In the vertical dimension, the server processes concurrent queries either one by one in an isolated way (I) or in a shared manner (S). Consequently, IXY, IID, SXY, and SID represent four possible system schemes. In the \*XY schemes, upon receiving a client report, the server invokes an additional procedure that identifies the host link via the link R-tree and link index before further processing. In the I\* schemes, the link hashing table ( $H_{lq}$ ) does not contain the list  $L_Q$  that contains the queries on a link. In the experiments, we compare these four system schemes to investigate the performance gains of the proposed strategies.

We used Brinkhoff's generator and the road network of Oldenburg [4], to generate network-based moving object workloads of 1K to 10K moving objects. Each workload is active for 1000 time stamps. At each time stamp, new objects come to be active, existing objects report to the server, and/or existing objects stop being active. The maximum object velocity is varied from 10 to 100 map units per time stamp. We generated static query sets of 1K to 10K queries. Each static query is produced as follows. First, a road network link is selected at random from all links. Then a position on that link is chosen at random as the query point (by choosing a line segment of that link at random and interpolating with a random ratio along that segment). Finally, the query range is decided as a multiple (randomly picked from 1, 2 to 5) of the length of the host link just selected. For the mobile query sets, we made the moving objects as query issuers, and determined the query ranges in the same way as for static queries. The parameters used in the experiments are listed in Table 2. The values in bold are the default settings used when their corresponding parameters are fixed. We implemented all system schemes in Java (JDK 1.4), and conducted all experiments on a Pentium IV desktop PC running MS Windows XP with a 3.00GHz CPU and 1GB RAM.

**Experimental Results on Static Query Sets.** We consider two performance metrics: (1) the amortized CPU time spent on query processing per time stamp and (2) the average main memory consumption per time stamp. Each experiment ran for 1000 time stamps, with all queries active from start to finish.

**Table 1.** Different system schemes

<i>Client report</i>		
<i>Query execution</i>	<i>Object Location</i>	<i>Host Link ID</i>
<i>Isolated</i>	IXY	IID
<i>Shared</i>	SXY	SID

**Table 2.** Parameters used in experiments

Parameter	Setting
Total time stamps	1000
Moving object card.	1K, 2K, ..., 10K
Max object vel.	10, 20, ..., 50, ..., 100
Query sets card.	1K, 2K, ..., 10K
Query range multiple	1, 2, ..., 5

To understand the effect of object cardinality on CPU time, we varied the cardinality from 1K to 10K. The results are reported in Figure 11(a). SID always performs significantly better than other schemes, while IXY is always the worst. SID saves considerable CPU time in the processing of network distance based queries by not only executing relevant concurrent queries in a shared fashion, but also exploiting the clients' capabilities to determine host links. For object cardinalities up to 6K, SXY outperforms IID; the opposite is seen when the cardinality exceeds 6K. For small object sets, the shared execution of concurrent queries has the largest effect. However, for large object sets, the host link ID reporting strategy becomes crucial, because the server needs to process much more frequent update reports from the clients; thus, host link IDs in the reports save significant CPU time. As object cardinality increases the CPU time cost of SID grow slowly, which certainly demonstrates that it is scalable.

The results reported in Figure 11(b) show the effect on CPU time of the maximum object velocity, which we varied from 10 to 100 map units per time stamp. Not surprisingly, SID remains the best and most scalable scheme because it uses both efficient strategies. The relative performance for SXY and IID is alike to that for the previous experiment. As the objects move faster, their updates to the server become more frequent, which finally renders the host link ID reporting strategy crucial.

To see the effect of query cardinality on CPU time, we varied it from 1K to 10K. The results are reported in Figure 11(c). Due to the lack of shared execution, IXY and IID degrade badly as more and more concurrent monitoring queries exist in the system. In contrast, SXY and SID both perform steadily because of the shared execution strategy. The slight improvements from 9K to 10K are due to more overlaps of sensitive links among different queries caused by the larger number of concurrent queries.

The results reported in Figure 11(d) show the effect on CPU time of the query range multiple, which we varied from 1 to 5. A larger query range means that more links are covered by queries. Shared execution helps alleviate the burden caused by additional sensitive links, as indicated by the better performance of SXY and SID. Nevertheless, SXY degrades much more than SID because the latter benefits substantially from host link ID reporting when many links are involved.

The experiments above also observed memory consumption, as shown in Figure 12. For memory cost, we only consider the data structures used to process the queries. As SXY (IXY) and SID (IID) use the same data structure on server side, we only compared the costs of the "Shared" and "Isolated" schemes. We see that the shared scheme consumes moderately more memory than the isolated scheme. This additional cost is due

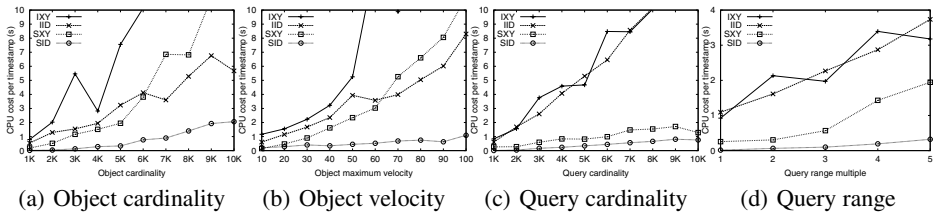


Fig. 11. CPU times on static query sets

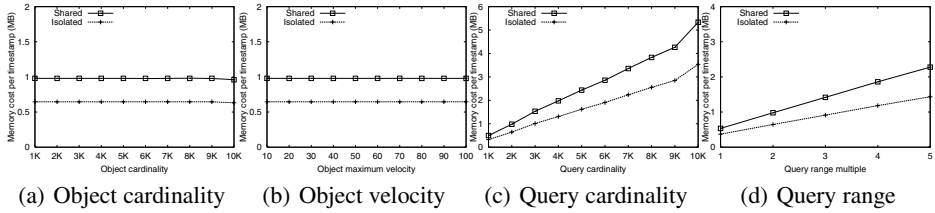


Fig. 12. Memory costs on static query sets

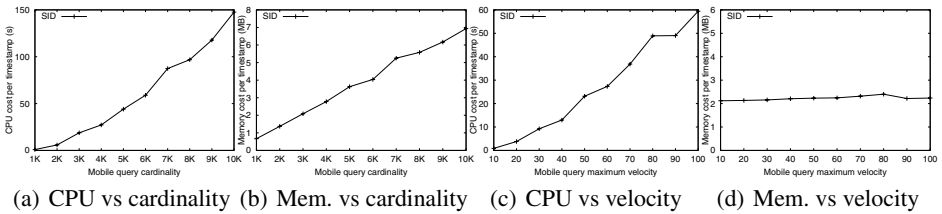


Fig. 13. Results on mobile query sets

to the link-to-query mapping in  $H_{lq}$ . As the query cardinality increases, more entries are maintained in the query hash table ( $H_q$ ). This leads to the memory cost increase in both schemes, as shown in Figure 12(c). As the query range increases, more links are maintained in the sensitive-link lists. This explains the observation in Figure 12(d) that the memory costs grow as query range increases. The (almost) linear growing patterns, together with the fact that neither object cardinality nor velocity impact the memory consumption, indicate that our techniques are scalable.

**Experimental Results on Mobile Query Sets.** For this set of experiments, we let all moving objects send continuous range monitoring queries to the server with the query points being their current positions. Each moving object issues a continuous query when it sends a REG message to the server, and the query is updated when update reports are received by the server. When a DEL message is received, the object’s mobile query is terminated. As SID is shown to be the most efficient scheme in Section 5.1, we only consider SID for the mobile query sets. Each experiment also ran for 1000 time stamps.

We first varied the mobile query cardinality, i.e., the number of moving objects, from 1K to 10K. The CPU costs are shown in Figure 13(a). As expected, the CPU cost is much higher compared to those obtained for static query sets—mobile queries invoke extra updates. Figure 13(b) shows that as the mobile query cardinality increases, the memory cost grows at a rate close to what is seen for static queries (Figure 12(c)). This is because mobile queries use the same query hash as do static queries, and almost need no extra memory space except the special mappings facilitating result removal.

Next we fixed the mobile query cardinality at 2K and varied the query velocity from 10 to 100. Results are reported in Figure 13(c) and 13(d). Faster mobile queries lead to more frequent server-side computations, updating both the objects position information and the query results. This trend is consistent with the results shown in Figure 13(c). As seen in Figure 13(d), the memory cost does not change much as the query velocity changes; the memory cost is affected mainly by the query cardinality. We also varied the mobile query range multiple from 1 to 5 to observe its effect on the SID performance. Both CPU time and memory cost exhibit slight variation as the query range changes, and they are very close to their counterparts on the 2K mobile query set covered in Figure 13(a) and 13(b). Due to the space limitation we omit the graphs here.

## 5.2 Client Side Experiments

We implemented our client design using SuperWaba [11], a Java-based open-source platform for mobile terminal applications development. We conducted a set of experiments on an HP iPAQ h6365 pocket PC, running MS Windows Mobile 2003 with a 200MHz processor and 55MB user accessible SDRAM.

Our main concern is to determine how much extra CPU time is spent on our special client design that involves the reporting of host links, compared to a usual mobile handset client that only reports coordinate locations to the server. We used the moving-object workloads generated in Section 5.1. For each data set, our client program on the pocket PC processes all location updates for each network-constrained moving object. For each object, the extra CPU time is accumulated and finally amortized across all time stamps. We then report the average of all objects' amortized extra CPU time costs.

The results reported in Figure 14(a) are those gained for moving objects of 1K to 10K. It is seen that at each time stamp, the client's extra CPU time cost is close to 6 milliseconds, which is negligible compared to the gain we achieved by shifting the server load to the clients, according to the results reported in Figure 11(a). This demon-

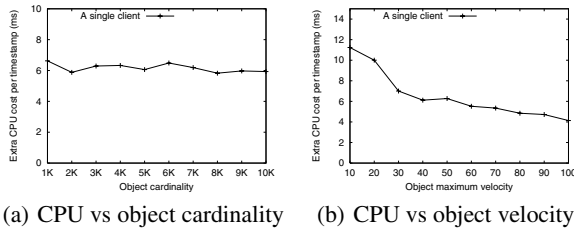


Fig. 14. Additional CPU times on client side

strates the benefit of our shifting strategy again. We also investigated the effect of the maximum object velocity on the client side CPU time. The results are reported in Figure 14(b). During a fixed active period, as an object moves faster, it produces more location updates and tends to invoke more data retrieval because it moves out of the current display window. Consequently, the extra CPU time costs due to host link identification decreases as the navigation module in our design contributes more by triggering a simple update procedure with known hosts, described in Section 3.3. Therefore, the amortized cost over time stamps decreases in spite of more updates occurring. The observations provide evidence that our client design is effective in taking advantage of the navigation module, whose own costs increase as the object moves faster independently of whether our specific additional design is present or not.

## 6 Related Work

Several research results have recently been reported that concern spatial-network constrained moving objects. Shahabi et al. [19] employ an embedding technique to transform a road network to a higher dimensional space, where shortest paths between original network points are computed and used to help pruning in  $k$ NN search based on network distance. Jensen et al. [12] formalize the data model and problem definition for nearest neighbor queries in road networks, and they adapt Dijkstra's algorithm [7] to compute the nearest neighbors for a mobile query point on the fly. Cho and Chung [6] consider continuous  $k$ NN queries aiming at static points of interest in a road network, whereas we query against mobile objects in this paper. Mouratidis et al. [14] address continuous  $k$ NN monitoring for spatial network constrained moving objects. They employ specific tree structures to support the shared execution of multiple  $k$ NN monitoring queries. In contrast, we use simple yet efficient hashing tables for concurrent range monitoring queries. All these works [6,12,14,19] assume that a central data server is solely responsible for the query processing. Our work differs in that we exploit the computing capabilities of the mobile terminals to aid the server in the query processing.

The idea of shifting work from the server to the clients has been addressed in the context of monitoring free-moving objects. Gedik and Liu [9] propose a distributed mobile system architecture, in which relevant monitoring query information is installed on mobile clients, enabling them to delay update reports to the server and process queries locally. Cai et al. [5] identify a resident domain for each mobile client, which in turn reports to the server only when it enters/leaves a resident domain so that queries can be updated correctly. Hu et al. [11] propose to use safe regions for the send mobile clients, within which continuous spatial monitoring results do not change. These techniques, however, are not applicable to our problem concerning network-constrained moving objects and network distance-based queries.

## 7 Conclusion

In this paper, we have proposed a full-fledged, novel solution to the continuous range monitoring of moving objects in a road network setting. The proposal takes advantage of the computing capabilities of mobile terminals to off-load computation from the

server-side to the moving objects, and the server groups relevant concurrent queries together and then processes the queries in each group in a shared fashion. Extensive experiments with the prototype implementation capture the performance characteristics of the proposal, and suggest that the proposal is efficient and applicable in practice.

## Acknowledgments

The work of Hua Lu, Zhiyong Huang and Linhao Xu was in part funded by A\*STAR under grant no. 032 101 0026. Christian S. Jensen is also an adjunct professor at Agder University College, Norway. His work was funded in part by the Danish Research Agency's Programme Commission on Nanoscience, Biotechnology, and IT.

## References

1. Superwaba (2006), <http://www.superwaba.com>
2. de Almeida, V.T., Güting, R.H.: Indexing the trajectories of moving objects in networks. *Geoinformatica* 9(1), 33–60 (2005)
3. de Almeida, V.T., Güting, R.H.: Using Dijkstra's algorithm to incrementally find the k-nearest neighbors in spatial network databases. In: *Proc. ACM SAC*, pp. 58–62. ACM Press, New York (2006)
4. Brinkhoff, T.: A framework for generating network-based moving objects. *Geoinformatica* 6(2), 153–180 (2002)
5. Cai, Y., Hua, K.A., Cao, G.: Processing range-monitoring queries on heterogeneous mobile objects. In: *Proc. MDM*, pp. 27–38 (2004)
6. Cho, H.-J., Chung, C.-W.: An efficient and scalable approach to CNN queries in a road network. In: *Proc. VLDB*, pp. 865–876 (2005)
7. Dijkstra, E.W.: A note on two problems in connection with graphs. *Numerische Mathematik* 1, 269–271 (1959)
8. Frentzos, E.: Indexing objects moving on fixed networks. In: Hadzilacos, T., Manolopoulos, Y., Roddick, J.F., Theodoridis, Y. (eds.) *SSTD 2003*. LNCS, vol. 2750, pp. 289–305. Springer, Heidelberg (2003)
9. Gedik, B., Liu, L.: Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) *EDBT 2004*. LNCS, vol. 3268, pp. 67–87. Springer, Heidelberg (2004)
10. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: *Proc. ACM SIGMOD*, pp. 47–57. ACM Press, New York (1984)
11. Hu, H., Xu, J., Lee, D.L.: A generic framework for monitoring continuous spatial queries over moving objects. In: *Proc. ACM SIGMOD*, pp. 479–490. ACM Press, New York (2005)
12. Jensen, C.S., Kolár, J., Pedersen, T.B., Timko, I.: Nearest neighbor queries in road networks. In: *Proc. ACM GIS*, pp. 1–8. ACM Press, New York (2003)
13. Mokbel, M.F., Xiong, X., Aref, W.G.: SINA: Scalable incremental processing of continuous queries in spatio-temporal databases. In: *Proc. ACM SIGMOD*, pp. 623–634. ACM Press, New York (2004)
14. Mouratidis, K., Yiu, M.L., Papadias, D., Mamoulis, N.: Continuous nearest neighbor monitoring in road networks. In: *Proc. VLDB*, pp. 43–54 (2006)
15. Ooi, B.C., Tan, K.-L., Yu, C.: Frequent update and efficient retrieval: an oxymoron on moving object indexes? In: *Proc. WISE Workshops*, pp. 3–12 (2002)

16. Pfoser, D., Jensen, C.S.: Indexing of network constrained moving objects. In: Proc. ACM GIS, pp. 25–32. ACM Press, New York (2003)
17. Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query processing in spatial network databases. In: Proc. VLDB, pp. 802–813 (2003)
18. Prabhakar, S., Xia, Y., Kalashnikov, D.V., Aref, W.G., Hambrusch, S.E.: Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Trans. Computers* 51(10), 1124–1140 (2002)
19. Shahabi, C., Kolahdouzan, M.R., Sharifzadeh, M.: A road network embedding technique for k-nearest neighbor search in moving object databases. In: Proc. ACM GIS, pp. 94–100. ACM Press, New York (2002)

# Compression of Digital Road Networks<sup>\*</sup>

Jonghyun Suh<sup>1</sup>, Sungwon Jung<sup>1</sup>, Martin Pfeifle<sup>2</sup>,  
Khoa T. Vo<sup>3</sup>, Marcus Oswald<sup>3</sup>, and Gerhard Reinelt<sup>3</sup>

<sup>1</sup> Department of Computer Science, Sogang University, Seoul, Korea  
{joe77, jungsung}@sogang.ac.kr

<sup>2</sup> Siemens VDO Automotive, Regensburg, Germany  
martin.pfeifle.ext@siemensvdo.com

<sup>3</sup> Department of Computer Science, University of Heidelberg, Germany  
{khoa.vo,marcus.oswald,gerhard.reinelt}@informatik.uni.heidelberg.de

**Abstract.** In the consumer market, there has been an increasing interest in portable navigation systems in the last few years. These systems usually work on digital map databases stored on SD cards. As the price for these SD cards heavily depends on their capacity and as digital map databases are rather space-consuming, relatively high hardware costs go along with digital map databases covering large areas like Europe or the USA. In this paper, we propose new techniques for the compact storage of the most important part of these databases, *i.e.*, the road network data. Our solution applies appropriate techniques from combinatorial optimization, *e.g.*, adapted solutions for the minimum bandwidth problem, and from data mining, *e.g.*, clustering based on suitable distance measures. In a detailed experimental evaluation based on real-world data, we demonstrate the characteristics and benefits of our new approaches.

## 1 Introduction

In the last few years, the demand for portable navigation systems has steadily increased. In Germany, for instance, in 2006 more than four times as many systems were bought than in the previous year. Obviously, the higher the data coverage, the higher the benefit for the end user. Unfortunately, high data coverage goes along with high hardware cost. The compressed storage of digital map databases is beneficial not only for SD cards but also for traditionally used DVDs, since it leads to an enormous reduction of time-critical I/O operations. Thus, one of the ultimate goals for the database providers is to store digital map databases in an extremely size-effective way.

In a very complex process, which often takes several days, digital map databases are transformed from a rather space-consuming raw data format, *e.g.*, in some GDF-flavour, into a compressed format useful for navigational applications. The time required for generating such compressed digital map databases is not very critical as it is done “off-line.” On the other hand, decompressing

---

<sup>\*</sup> This work was supported in part by KOSEF grant No. R01-2006-000-10536-0(2007) and in part by the Brain Korea 21 Project in 2007.



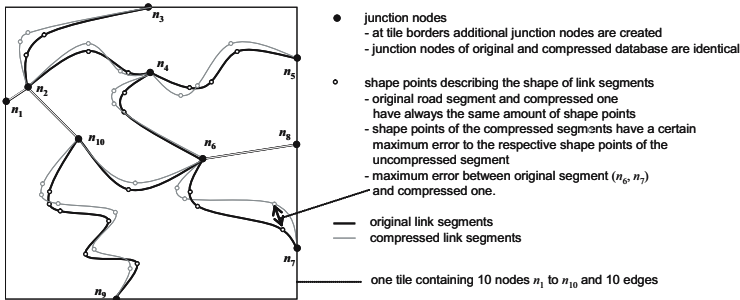


Fig. 1. Road network within one tile

the database needs to be as efficient as possible because it is done repeatedly “on-line” during the navigation process.

A central part of a digital map database is the routing building block which contains information on the topological road network along with some additional geometric information on the positions of nodes and the shapes of links. The topological road network is used for route calculation. The additional geometric information is used, for instance, for positioning, route guidance, and map display. For route guidance, it is important that the exact positions of the nodes, *i.e.*, the junctions, are accurate in order to inform the driver in due time to turn right or left. On the other hand, small deviations on the shapes of the links are acceptable (cf. Figure 1).

In order to have handy loading units, most database providers partition the road network into small cells, sometimes also called *tiles* or *parcels*. The cells can either be created by strict Quadtree tiling of the data space or by some more data oriented techniques like the creation of a KD-tree on the node data. In any of the two partitioning cases there is a subgraph of the complete road network stored in one tile (cf. Figure 1).

In this paper, we introduce techniques which reduce the size of the road network. For the lossless compression of the topological information, an adaptation of the most suitable solution for the minimum bandwidth problem is applied. Basically, our algorithm orders the nodes of the road network in such a way that each edge can be described by a small and limited number of bits. The lossless compression of the geometrical information is done by finding for each node a suitable reference node for encoding the positional differences. In this paper, we propose several different solutions among which the use of minimal spanning trees on the road network is the most promising one. Finally, for compressing the shapes of links, we exploit the fact that there exist a lot of similar shapes in digital map databases. We cluster all shapes based on a non-symmetric distance measure, and then extract suitable representatives from the resulting clusters. Each shape in the digital map database is represented by a reference to a pattern shape, if the thereby introduced error does not exceed a maximum user-defined value.

The remainder of this paper is organized as follows: In Section 2, we survey related work in the area of compressing digital road networks. In Section 3,

we describe the topological compression of the road network, and in Section 4 the geometrical compression. Section 5 presents our experimental evaluation, which is based on real-world data. We close this paper in Section 6 with a short summary and a few remarks on future work.

## 2 Related Work

Traditionally, the road network data is stored in one big file and the data of a certain tile is consecutively stored as one data chunk. Such a data chunk is usually further divided into data describing the topology and data describing the geometry.

### 2.1 Topology

Because road networks can be regarded as rather sparse graphs, the topology of digital map databases is usually stored in some variant of an adjacency list. Typically, the graph is assumed to be undirected. Directional along with other information such as speed limit is stored separately for each link.

Assuming we are given an (arbitrary) node ordering (cf. Figure 1), we store the graph as follows: First, we store the number of nodes  $n$  and the maximum node degree  $d_{max}$ . Next, we store for each node the number of adjacent nodes. For this we need at most  $\lceil \log_2(d_{max} + 1) \rceil$  bits. Next, the list of adjacent nodes according to the node ordering follows. For each entry  $\lceil \log_2(n + 1) \rceil$  bits are required. Note that we do not store edges twice. We only store them at the node with the smaller node id. For the example of Figure 1, we would store the values shown in Table 1.

**Table 1.** Storage of the topological information

Data	Values (no. of bits)	Remarks
$n$	10(16)	2 bytes for the no. of nodes per tile
$d_{max}$	4(8)	1 byte for the maximum node degree
node 1	1(3) 2(4)	one edge: $(n_1, n_2)$
node 2	3(3) 3(4)4(4)10(4)	three edges: $(n_2, n_3), (n_2, n_4), (n_2, n_{10})$
node 3	0(3)	zero edges
node 4	2(3) 5(4)6(4)	two edges: $(n_4, n_5), (n_4, n_6)$
..	..	..

A small improvement of the approach outlined above would be to store only the delta information of the node numbers rather than the absolute node values when encoding edge information. Delta encoding is used in various applications, e.g., HTTP, video codecs, image processing, and so on. Note that big differences between the node numbers are still possible, even for those nodes connected by an edge (cf.  $(n_2, n_{10})$  in Figure 1 where the difference is still 8).

Basically, in the approach introduced in Section 3, we try to minimize the maximum delta between node numbers for nodes connected to each other. For

instance, if in Figure 1, we exchange the node numbers of node  $n_{10}$  and  $n_7$ , the maximum delta between two connected nodes decreases from 8 to 5, which would allow encoding each edge using only 3 bits instead of 4 bits.

### 2.2 Geometry

**Node Encoding.** Based on an already existing node ordering, the longitude and latitude values can be stored using delta encoding. Thereby, the longitude and latitude values of the first node are stored absolutely, whereas for each subsequent node only the positional difference to its predecessor node is stored. A commonly used way to encode (absolute and delta) integer values is to use a UTF-8 like notation, where the first bit indicates whether there is another byte available belonging to this integer value. In 1 this approach is called *variable-byte approach*. Although only 7 bits remain for encoding the (delta) integer values, it is possible to encode a lot of small integer values by only one or 2 bytes instead of 4. Other useful techniques, in particular for encoding small integer values, can also be found in 1, e.g., *Elias Gamma* and *Delta compression*, or *Golomb compression*. Note that any kind of delta encoding fails to produce good results if the reference node is far away, e.g., in Figure 1 the positions of  $n_8$  and  $n_9$  differ greatly.

Since delta encoding has already been extensively discussed in the literatur, we concentrate in this paper on the finding of suitable reference nodes (cf. Section 4.1).

**Shape Point Encoding.** For reducing the size of the shape information the Douglas-Peucker Algorithm 2 is often used (cf. Figure 2). Basically, the algorithm removes shape points as long as the thereby introduced error is smaller than a user-defined threshold  $\epsilon$ . In Section 4.2, we propose a completely different and new approach exploiting similarities between shapes rather than trying to compress shapes individually.

## 3 Topological Compression

In Section 2.1, we showed that traditional approaches require  $\lceil \log_2(n + 1) \rceil$  bits for encoding a single edge within a tile containing  $n$  nodes. The basic idea of this

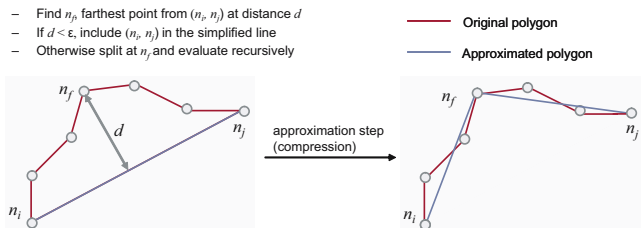


Fig. 2. Douglas-Peucker algorithm

section is to order the nodes in such a way that we need a much smaller number of bits for encoding each single edge. To achieve this goal, we adapt solutions for the bandwidth minimization problem which allows us to store a single edge with only  $\lceil \log_2 (UB_{bdw} + 1) \rceil$  bits where  $UB_{bdw}$  reflects an approximated upper-bound solution of the problem.

### 3.1 Bandwidth Minimization Problem

The *bandwidth minimization problem* is a classical combinatorial optimization problem that has been studied since around 1960. It is formulated as follows. Let  $G = (V, E)$  be an undirected graph with node set  $V$ , where  $|V| = n$ , and edge set  $E$ , where  $|E| = m$ . The task is to find a labeling  $l$  of the nodes with numbers 1 through  $n$  such that the maximum difference  $|l(u) - l(v)|$ , for  $(u, v) \in E$ , is minimized. The name itself originates from an equivalent matrix problem. Given an  $(n, n)$ -matrix  $M$ , the bandwidth problem consists of finding a simultaneous permutation of the rows and columns of  $M$  such that the distance of nonzero elements to the main diagonal is as small as possible, *i.e.*, to obtain a permuted matrix of minimum bandwidth. Figure 3 shows an example with the original matrix on the left and the permuted matrix with reduced bandwidth on the right. The equivalence between the two formulations can be observed when forming the adjacency matrix of the graph. Then the minimum bandwidth of this matrix corresponds to an optimal node ordering. Thereby, optimality means that the maximum distance in the node ordering between two arbitrary nodes, which are connected to each other in the graph, is as small as possible.

The problem is NP-hard [3], even for binary trees [4]. Due to this difficulty, algorithms computing optimum solutions are very time consuming in general and therefore heuristics are used for finding good approximate solutions. To assess the quality of solutions it is important to compute lower bounds for the optimum value. The distance between lower and upper bound is called *solution gap*. Currently, the best exact method is the branch-and-bound algorithm *BB* developed by Caprara and Salazar [5]. They also introduce two formulae for

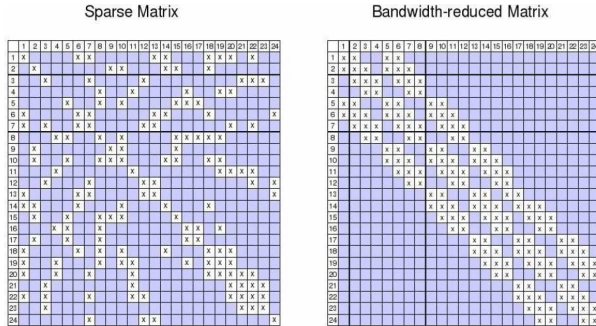


Fig. 3. Matrix bandwidth minimization

computing lower bounds efficiently. The best heuristic approach is the algorithm *SS\_TS* (*Scatter search with Tabu search*) by Campos, Piñana, and Martí [6]. However, the solution gap stays very large. For example, BB takes 36 minutes for a rather sparse graph of 200 nodes, and results in a lower bound of 54 and an upper bound of 84. *SS\_TS* takes 100 seconds for this graph yielding the lower bound 57 and the upper bound 67.

For our application, we do not necessarily need to find the optimal bandwidth of a graph. Since we use binary encoding, a bandwidth of 8 yields the same compression ratio as a bandwidth of 15, and the bandwidth 63 is as good as the bandwidth 32. In other words, if we find a solution gap that is contained in an interval  $[2^{k-1}, 2^k - 1]$  for some  $k$ , then we know that the optimal solution for encoding the edges requires  $k$  bits. Note that the computation of the upper bound value  $UB_{bdw}$  should also yield a concrete node ordering. This node ordering can then be used for storing the topology of the road network as described in Section 2.1 by using only  $\lceil \log_2(UB_{bdw} + 1) \rceil$  bits instead of  $\lceil \log_2(n + 1) \rceil$  bits for encoding an edge.

Since a road network database is huge, containing up to a hundred thousand tiles each containing around 250 nodes, faster heuristics than existing ones are required.

**Definitions and Notations.** Let  $G = (V, E)$  be a connected graph. The *degree* of a node  $v$  is the number of nodes adjacent to  $v$ . Given two nodes  $u, v \in V$ ,  $d(u, v)$  denotes the distance between  $u$  and  $v$ , *i.e.*, the number of edges lying on the shortest path from  $u$  to  $v$ .  $d(u, V)$  denotes the maximum distance between  $u$  to nodes in  $V$ .

For a subset  $S \subseteq V$  we define its *diameter*  $d(S) = \max\{d(u, v) : u, v \in S\}$ , which is the maximum distance between two nodes of  $S$ .

A *level structure* is a partition of  $V$  into sets  $L_1, L_2, \dots, L_k$  called levels. If node  $u \in L_i$  is adjacent to node  $v \in L_j$ , then  $|i - j| \leq 1$ .

The *width of a level* is the number of nodes contained in it. The *width of a level structure* is the maximum width among its levels. A *level structure*  $L_v(G)$  *rooted at*  $v$  satisfies  $L_1 = \{v\}$ , and for every  $i > 1$ ,  $L_i$  is the set of all nodes adjacent to nodes in  $L_{i-1}$  that have not yet been assigned to any level. (In other words, nodes in level  $L_i$  are at distance  $i - 1$  from  $v$ .) Obviously,  $L_v(G)$  can easily be obtained with breadth-first-search started at  $v$ .

### 3.2 Lower Bound Computation

In [5], Caprara and Salazar gave the following two lower bounds on the minimum bandwidth of a graph:

$$\begin{aligned} \gamma(G) &= \min_{v \in V} \max \left\{ \left\lceil \frac{|N_k(v)| - 1}{k} \right\rceil \mid k = 1, \dots, d(v, V) \right\} \\ \alpha(G) &= \max_{v \in V} \max \left\{ \left\lceil \frac{|N_k(v)| - 1}{2k} \right\rceil \mid k = 1, \dots, d(v, V) \right\} \end{aligned}$$

where  $N_k(v)$  is the set of nodes with distance at most  $k$  from  $v$ .

These two lower bounds  $\gamma(G)$  and  $\alpha(G)$  can be computed in  $O(nm)$  time because, for every  $v \in V$ , it takes linear time  $O(m)$  to compute the breadth-first-search tree rooted at  $v$ .

### 3.3 Upper Bound Computation

As the computation of the upper bound mentioned in section 3.1 has to be applied to hundred thousand tiles in a couple of days, each one is limited only to a few seconds. Therefore neither the best exact method [5] nor the best heuristic [6] are appropriate for our application. Instead, we improve the fast heuristic proposed by Gibbs, Poole, and Stockmeyer [7]. It can be described as follows.

(1) *Find endpoints of a pseudo diameter*

An attempt is made to find two nodes with nearly maximal distance. The procedure selects nodes  $v$  of minimal degree and generates level structures rooted at  $v$ . Then it chooses a node  $u$  in the last level of the level structure of smallest width and returns  $u$  and  $v$ .

(2) *Minimize level width*

As level structures of smaller width result in a smaller bandwidth, this step generates two level structures rooted at  $u$  and  $v$ , then combines these two level structures into one, which usually has smaller width.

(3) *Node numbering*

Labels to nodes in the combined level structure are assigned consecutively from 1 to  $n$  level by level, labeling nodes with smaller degree first.

Our improvement uses the fact that the set  $N_k(v)$ , or a level structure rooted at  $v$ , has to be computed for each node  $v$  to compute the lower bound. In addition, because the numbering step is consecutive, the algorithm result depends largely on the way nodes are arranged in each level. The improvement is as follows:

(1) *Find nodes generating level structures of minimal width*

From the lower bound computations we obtain, find level structures with exact minimal width, instead of nearly minimal width as in the original algorithm.

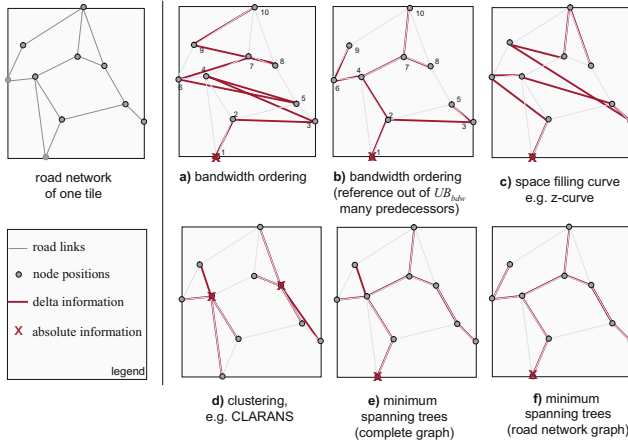
(2) *Generate and test*

Given the list of nodes of the previous step, we apply steps (2) and (3) of the original algorithm, and select the labeling with smallest bandwidth.

The worst case complexity is  $O(nm)$  for finding the list, and  $O(n^2 \log(n))$  for assigning and getting the best bandwidth, so the overall complexity is  $O(nm + n^2 \log(n))$ .

## 4 Geometrical Compression

In this section, we examine the geometrical compression of a road network. The position of node and shape points is usually given by two 4-byte integer values



**Fig. 4.** Compression of node positions

representing longitude and latitude values. As already mentioned, for application areas like guidance, it is critical that we have a lossless compression of the nodes, *i.e.*, junctions. On the other hand, a slight positional deviation for shape points is acceptable (cf. Figure 1).

#### 4.1 Node Compression

In order to keep from spending 8 bytes every time a position is encoded, delta encoding is usually applied (cf. Section 2.2). For delta encoding, it is necessary to encode the position of a point by referring to a nearby point. For encoding deltas, an appropriate function, *e.g.*, one of the approaches explained in 2, can be used. In this subsection, we discuss several approaches to finding appropriate reference points (cf. Figure 4).

**Bandwidth Ordering (Fixed Reference Node).** In Section 3, we introduced a node ordering that allows storing the edges, *i.e.*, the topological information, very compactly. Obviously, we can use the same ordering for encoding the node positions (cf. Figure 4(a)). The positions of the first nodes are stored with absolute values whereas the positions of the following nodes are stored relative to the position of their preceding node. The advantage of this approach is that we do not need additional references from the list of node positions to the topological node ordering since they are identical. The disadvantage is that rather big deltas between consecutive nodes are still possible, *e.g.*, the position of nodes 3 and 4 differ considerably in Figure 4(a), thus requiring many bits for encoding the deltas.

**Bandwidth Ordering (Variable Reference Node).** The idea behind the approach in Figure 4(b) is to not always use the direct predecessor node from the bandwidth ordering as the reference node, but rather the best one out of the  $UB_{bdw}$  many predecessors. Note that if there is an edge between  $n_i$  and  $n_j$

where  $i < j$ , we can encode the position of  $n_j$  by using  $n_i$  as reference because the distance in the bandwidth ordering between  $n_i$  and  $n_j$  is smaller than or equal to  $UB_{bdw}$ . Figure 4(b) shows that the resulting deltas, *e.g.*, the ones for node 4 or 6, are much smaller than in Figure 4(a), and can therefore be stored in a more compact way. On the other hand, besides the delta information, we have to store  $\lceil \log_2(UB_{bdw} + 1) \rceil$  bits for each node for describing the reference node. Note that we choose as reference node not necessarily the one with the smallest distance to the current node, but the one where we need the smallest number of bits for encoding the delta values.

**Space Filling Curves.** The idea of the approach in Figure 4(c) is to order the nodes according to a space-filling curve, *e.g.* Z-curve [8], or Hilbert-curve [9]. The disadvantage of this approach is that the node ordering is not related to the topological ordering. Thus, we have to store at each node a reference to the corresponding node in the topological ordering, consuming  $n \times \lceil \log_2(n + 1) \rceil$  additional bits.

**Clustering.** Based on clustering, we can find central points within our point sets that can be used as reference points (cf. Figure 4(d)). A suitable algorithm for our needs is CLARANS [10]. Based on a certain distance measure, CLARANS determines a user-defined number  $k$  of representative objects. Each database object is assigned to exactly one representative. Based on an efficient heuristic CLARANS tries to minimize the sum of all distances from the database objects to their representatives. Although we could use some  $L_p$  distance measure, it is more appropriate to use the number of bits which are necessary for encoding the deltas. The  $k$  representative objects are stored absolutely whereas for the remaining  $(n - k)$  objects deltas can be stored. Similar to the approaches in Figure 4(c) and (e), we need  $n \times \lceil \log_2(n + 1) \rceil$  bits for references to the topological information.

**Minimum Spanning Trees (Complete Graph).** The idea of the approach in Figure 4(e) is to compute the minimum spanning tree for the set of nodes. If we label the nodes of the graph with their topological unique numbers, we can store the minimum spanning trees as a Prufer-tree [11]. The overhead for connecting the minimum spanning tree to the topological road network is also  $n \times \lceil \log_2(n + 1) \rceil$  bits. Figure 4 indicates that in this approach the deltas are smaller than in the other approaches.

**Minimum Spanning Trees (Road Network Topology).** The idea in this approach is to restrict the minimum spanning tree to the current road network (cf. Figure 4(f)). The motivation behind this approach is that we need many fewer bits for connecting the geometrical data to the topological data than the approach above. On the other hand, the quality of the resulting minimum spanning trees is expected to be almost as good as on the complete graph. For instance, in Figure 4 the two approaches differ only in one edge. In areas where we have many physical dividers like rivers or mountains, the two spanning trees might differ more. But in the majority of real world road networks, especially in cities, the two minimum spanning trees are likely to be similar. A minimum



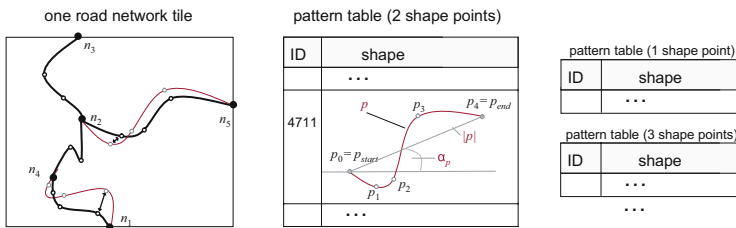
spanning tree on top of the road network can be stored efficiently by using a bit vector for the edges. Note that in Section 2.1, we defined not only a node ordering but also an ordering of the edges, *i.e.*, first all the edges following node  $n_1$ , then the edges following node  $n_2$ , etc.. In the edge bit vector, we set a bit if the corresponding edge is part of the minimum spanning tree. Thus, we can connect the geometrical node positions to the topological ordering by  $m = |E|$  additional bits. Note that for a typical sparse road graph consisting of 100 to 1000 nodes  $m \ll n \times \lceil \log_2(n + 1) \rceil$  holds.

### 4.2 Compression of Shapes

In this section, we describe the compression of the road shapes. The shapes are usually represented by a sequence of shape points which are piecewise linearly connected. Each point consists of a pair of integer values representing its longitude and latitude. As the node positions are fixed (cf. Section 4.1), the remaining question is how to store the  $n$  additional shape points of a segment. In the following, we first describe the general idea of our new lossy compression technique, followed by technical details regarding the distance measure and the clustering algorithm used.

**General Idea.** The new idea behind our approach is that we do not compress shapes individually but rather try to compress shapes by exploiting similarities between them. A shape consists of two endpoints and  $n$  additional 2-dimensional points. Formally, it is defined as follows:

**Definition 1.** A shape  $s = \langle s_0, s_1, \dots, s_n, s_{n+1} \rangle$  is defined as a sequence of  $n + 2$  2-dimensional points for which  $s_{start} = s_0 \neq s_{end} = s_{n+1}$  and  $s_{start}.x \leq s_{end}.x$  holds. The norm  $|s|$  of a shape is defined as  $|s| = |s_{end} - s_{start}|$  and the angle  $\alpha_s$  is defined as  $\alpha_s = \arctan \frac{s_{end}.y - s_{start}.y}{s_{end}.x - s_{start}.x}$ .



**Fig. 5.** General idea of lossy shape compression

Besides the above defined values  $s$ ,  $\alpha_s$ , and  $|s|$ , Figure 5 shows the general idea of our approach. Basically our approach works in the following steps:

- Group segments together according to their number of shape points. This grouping is based on the complete database and is not restricted to individual tiles.

- For each group find suitable representative shapes and store them normalized in a global pattern table.
- Replace shapes in the database by references to pattern elements if thereby a maximum allowed error is not exceeded, *e.g.*, in Figure 5 shape  $p$  represents  $(n_2, n_5)$  and  $(n_4, n_1)$ .
- Translate, rotate, and scale a referenced pattern shape in such a way that the end points of the transformed pattern are identical to the end points of the original shape. This step is done during decompression, *e.g.*, in the run of a navigation system. Note that the positions of the start and end nodes are given exactly by the approach described in Section 4.1. These node positions can be used for computing the translation, rotation, and scaling factors on-line, *i.e.*, they do not have to be stored along with the pattern reference.

The grouping of elements according to the number of their shape points is a very simple step compared to finding suitable representatives for each group of shapes. This second step is based on a suitable distance measure which forms the foundation of the clustering of the shapes and the extraction of meaningful representatives. In the following two subsections, we will introduce a non-symmetric distance measure and an appropriate clustering algorithm.

**An Appropriate Distance Measure.** For the purpose of compressing shapes by referring to appropriate reference shapes, we now motivate the use of a non-symmetric distance measure. Although the resulting distance measure is not metric, we can use it for clustering. Before defining the distance measure and describing its properties, we define the normalization  $p^{-s}$  of a shape  $p$  with respect to (w.r.t.) a shape  $s$ . The start and end points of  $p^{-s}$  and  $s$  are identical (cf. Figure 6).

**Definition 2.** Let  $s = \langle s_0, s_1, \dots, s_n, s_{n+1} \rangle$  and  $p = \langle p_0, p_1, \dots, p_n, p_{n+1} \rangle$  be two shapes. Then, the normalization  $p^{-s} = \langle p_0^{-s}, p_1^{-s}, \dots, p_n^{-s}, p_{n+1}^{-s} \rangle$  of  $p$  w.r.t.  $s$  is defined by

$$p_i^{-s} = \frac{|s|}{|p|} \begin{pmatrix} \cos(\alpha_s - \alpha_p) & -\sin(\alpha_s - \alpha_p) \\ \sin(\alpha_s - \alpha_p) & \cos(\alpha_s - \alpha_p) \end{pmatrix} \cdot (p_i - p_0) + s_0,$$

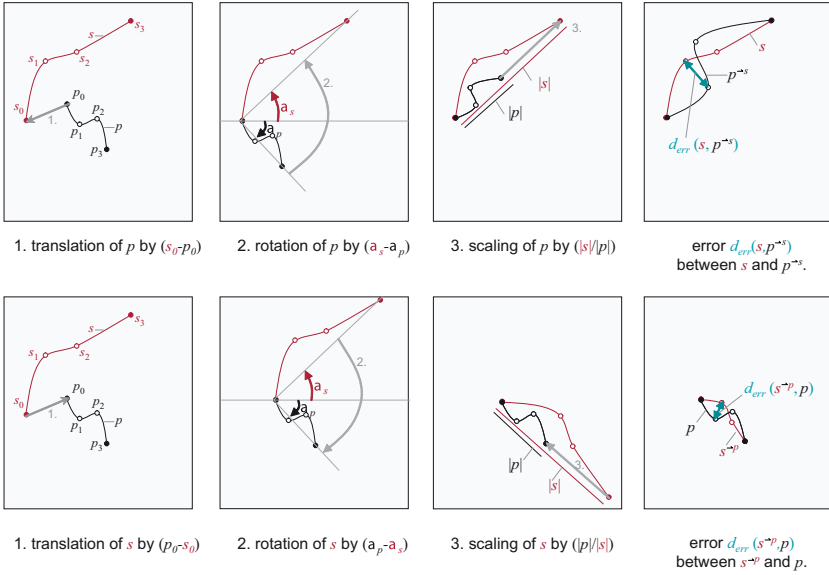
for all  $i = 0 \dots n + 1$ . The error  $d(s, p^{-s})$  between  $s$  and  $p^{-s}$  is given by the maximum distance of two shape points of  $s_i$  and  $p_i^{-s}$ ,

$$d_{err}(s, p^{-s}) = \max\{L_2(s_i, p_i^{-s}) \mid i = 1 \dots n\}.$$

Figure 6 shows that the introduced error differs if we normalize  $p$  w.r.t.  $s$  or if we normalize  $s$  w.r.t.  $p$ , *i.e.*,  $d_{err}(s, p^{-s}) \neq d_{err}(p, s^{-p})$ . Basically, the following property holds:

**Lemma 1.**  $d_{err}(s, p^{-s}) = \frac{|s|}{|p|} \cdot d_{err}(p, s^{-p})$

*Proof.* If we translate, rotate, and scale  $p$  and  $s^{-p}$  by  $(s_0 - p_0)$ ,  $(\alpha_s - \alpha_p)$ , and  $\frac{|s|}{|p|}$ , then  $p$  is transformed into  $p^{-s}$  and  $s^{-p}$  into  $s$ . Furthermore, for all  $i = 0, \dots, n + 1$  the vectors  $(p_i - s_i^{-p})$  are transformed into the vectors  $(p_i^{-s} - s_i)$ .



**Fig. 6.** The normalization  $p^{-s}$  of  $p$  w.r.t.  $s$  (above) and the normalization  $s^{-p}$  of  $s$  w.r.t.  $p$  (below)

Note that rotation and translation do not change the length of these vectors. Due to the theorems on intersecting lines, the length of the transformed vectors  $(p_i^{-s} - s_i)$  is  $\frac{|s|}{|p|}$  times the length of the original ones, proving the lemma.

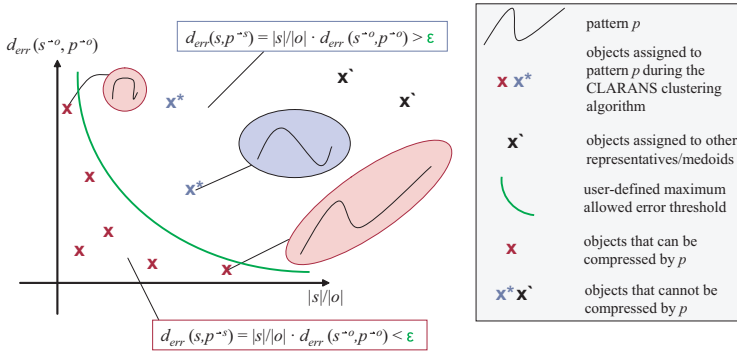
Lemma 1 shows that the error is much bigger when normalizing a shape  $s$  to a large shape  $p$  than to a small one. Based on Definition 2, we can now introduce our non-symmetric distance measure  $d_{sim}$ .

**Definition 3.** Let  $s$  and  $p$  be two shapes. Then the distance  $d_{sim}(s, p)$  between  $s$  and  $p$  is defined by  $d_{sim}(s, p) = d_{err}(s, p^{-s})$ .

We now cluster the shapes based on this distance measure. For each distance computation during the run of the clustering algorithm, we have to carry out one translation, rotation, and scaling operation. Typically, the number of distance operations during clustering is much greater than the number of shape objects in the database. In order to reduce the overhead induced by the transformation steps, we normalize each element w.r.t. a fixed but arbitrary shape  $o$ . This requires only one normalization step for each shape. Based on the normalized shapes  $p^{-o}$  and  $s^{-o}$ , we can compute the values  $d_{sim}(s, p) = d_{err}(s, p^{-s})$  and  $d_{sim}(p, s) = d_{err}(p, s^{-p})$ .

**Lemma 2.** Let  $s, p, o$  be three shapes with the same number of shape points. Then,  $d_{err}(s, p^{-s}) = \frac{|s|}{|o|} \cdot d_{err}(s^{-o}, p^{-o})$  and  $d_{err}(p, s^{-p}) = \frac{|p|}{|o|} \cdot d_{err}(s^{-o}, p^{-o})$ .

*Proof.* The start and end points of the shapes  $s^{-o}$  and  $p^{-o}$  are identical. In order to transform these shapes to the shapes  $s$  and  $p^{-s}$  we have to carry out



**Fig. 7.** Shapes of one cluster represented by pattern  $p$

a translation, rotation, and scaling by  $(s_0 - o_0)$ ,  $(\alpha_s - \alpha_o)$ , and  $\frac{|s|}{|o|}$ . Therefore, similar to the proof of Lemma 1,  $d_{err}(s, p^{-s}) = \frac{|s|}{|o|} \cdot d_{err}(s^{-o}, p^{-o})$  holds. Based on Lemma 1,  $d_{err}(p, s^{-p}) = \frac{|p|}{|o|} \cdot d_{err}(s^{-o}, p^{-o})$  can be derived from  $d_{err}(p, s^{-p}) = \frac{|p|}{|s|} \cdot d_{err}(s, p^{-s})$ .

**An Appropriate Clustering Algorithm.** In [12] an interesting approach is introduced which generates meaningful representatives based on the clustering algorithm OPTICS [13]. Because OPTICS, like many other approaches, requires metric distance functions, the approach in [12] is not useable. The idea of our approach is based on CLARANS [10]. CLARANS tries to group objects into a user-defined number of  $k$  clusters by maximizing a certain quality criterion. Usually this criterion is related to the sum of all distances of the objects to their representatives. In our case, we simply count the objects which can be represented by the  $k$  patterns. CLARANS stops if after a certain number of iterations, this quality criterion, *i.e.*, the number of representable shapes, does not increase any more. The higher the resulting quality is, the higher is the compression ratio. Figure 7 demonstrates that not all objects within a cluster represented by a certain shape  $p$  can be compressed by this shape. Only those shapes  $s$  can be compressed which are either very similar to  $p$  or which are very small. If  $|s|$  is very small,  $s$  can be represented by  $p$ , even if they are not intuitively similar. Thus, small patterns can more easily be compressed than large ones (cf. Lemma 1). Note that the overall compression ratio also depends heavily on the user-defined error-threshold  $\epsilon$ . If this value is large, *i.e.*, the user allows rather big errors, the compression ratio will be higher than for very small values.

## 5 Experimental Evaluation

In this section, we present the experimental evaluation demonstrating the characteristics and benefits of our approach. The evaluation is based on real-world

European road network data. We used two different data sets. *DB 1* consists of 10,000 arbitrarily chosen tiles containing between 163 and 296 nodes each. *DB 2* is always equal to a set of shapes all having the same number of shape points. Unless otherwise mentioned, the number of shapes in *DB 2* is 100,000, and each shape consists of 4 shape points. The experiments were carried out on a 3 GHz PC having 2 GB RAM running Linux. The algorithms were coded in C/C++.

## 5.1 Topological Compression

Table 2 depicts the results of the topological compression for some arbitrarily chosen, individual tiles of *DB 1*.  $LB_{bdw}$  is the lower bound and  $UB_{bdw}$  is the upper bound of the minimum bandwidth  $min_{bdw}$ .  $Original = 3 \times 8 + n \times \lceil \log_2(d_{max} + 1) \rceil + m \times \lceil \log_2(n + 1) \rceil$  denotes the number of bits needed for storing the graph according to Table 1 ( $n = |V|$  and  $m = |E|$ ). In our new approach, we have to spend one additional byte for encoding the value  $UB_{bdw}$  so that we know how many bits are required for the encoding of an edge. Therefore, the reduced number of bits for our topological compression approach is given by  $Reduced = 4 \times 8 + n \times \lceil \log_2(d_{max} + 1) \rceil + m \times \lceil \log_2(UB_{bdw} + 1) \rceil$ . Finally,  $\frac{Original - Reduced}{Original}$  indicates the compression ratio.

**Table 2.** Evaluation of the topological compression approach

Nodes n	Edges m	max. degree $d_{max}$	$LB_{bdw}$	$UB_{bdw}$	Time [sec.]	Original [Bit]	Reduced [Bit]	Ratio [%]
163	212	6	7	11	0.05	2209	1369	38.03
221	301	7	9	15	0.10	3095	1899	38.64
296	397	7	12	21	0.19	4485	2905	35.23

Table 2 shows that we can achieve a noteworthy compression ratio when using the node ordering corresponding to the value  $UB_{bdw}$  instead of an arbitrary node ordering. In most of our tests,  $UB_{bdw}$  was beneath 16, which allows us to encode an edge with only 4 bits instead of 8 or more bits. For the 10,000 tiles, we achieved an average compression ratio of 37.98%.

Although the runtime for computing a small enough upper bound value  $UB_{bdw}$  for each graph increases super linear with the number of nodes, our approach is still applicable since usually only a very limited number of nodes are within one tile. Note that the compilation of a complete North America or Western Europe database typically takes several days on much faster computers. The additional computational overhead for the  $UB_{bdw}$ -ordering stays far beyond 1% of the overall compilation time. Compared to an arbitrary node ordering, no additional time is required for decompression on the target system.

To sum up, our proposed algorithm for efficiently computing a small upper-bound value  $UB_{bdw}$  along with its node ordering is well suited to compress the topology of a sparse road network graph.

## 5.2 Geometrical Compression

**Node Compression.** In this section, we present the results of the geometrical compression of the node positions. All tests were carried out on *DB 1*. In all approaches introduced in Section 4.1, we encode a few positions absolutely using  $2 \times 4$  bytes and the remaining node positions are encoded by deltas to reference nodes. Note that in this paper, we focus on the finding of suitable reference nodes rather than on suitable delta encoding techniques. For encoding the deltas, in all our tests the variable-byte integer encoding of [1] was used. In Table 3 the number of bytes for each approach is given. *Ratio* reflects the normalized number of bytes w.r.t. the number of bytes required for the approach depicted in Figure 4(a). Compression times are not reported, since in all approaches, they stay considerably beneath the time required for the topological compression. In all presented approaches decompression is a straight-forward and simple task. The resulting decompression times are very similar and are therefore not reported.

Table 3 clearly shows that the overall best approach is based on minimum spanning trees computed on the graphs given by the road network. This approach requires considerably fewer bytes for referring to the topological information than the approach of the minimum spanning tree on the complete graph which leads to the overall smallest delta values. Table 3 shows that it is important to not spend  $\lceil \log_2(n + 1) \rceil$  bits for each node for referencing purposes. Interestingly, the approach that uses one out of  $UB_{bdw}$  predecessors as reference point (cf. Figure 4(b)) is also well suited. As already argued in Section 4.1, it is likely that a suitable representative for each node will be found among its  $UB_{bdw}$  predecessor nodes. Note that this approach also needs only  $n \times \lceil \log_2(UB_{bdw} + 1) \rceil$  bits for connecting the topological and geographical information. Approaches based on traditional spatial databases techniques, like space-filling curves or clustering, are far less effective.

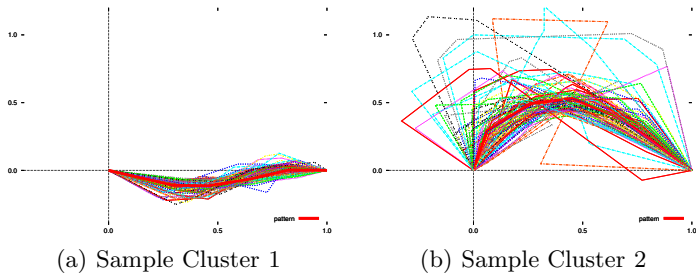
To sum up, the minimum spanning tree based on the road network is the best way to compress the geometry of the nodes.

**Shape Compression.** In this subsection, we discuss the characteristics of our new shape compression approach that exploits similarities between shapes.

Figure 8 depicts two sample clusters from a clustering of *DB 2* resulting in 1024 clusters. Sample cluster 1 contains 99 shapes from which 77 can be substituted by a reference to the medoid of the cluster (cf. bold line). Sample

**Table 3.** Evaluation of node compression approaches

	Bandwidth		Clustering			Space Filling Curve		Min. Span. Tree	
	1	$UB_{bdw}$	$k = 5$	$k = 20$	$k = 50$	$z - curve$	<i>hilbert</i>	<i>full</i>	<i>roads</i>
Absolte [Byte]	8	8	50	160	400	8	8	8	8
Delta [Byte]	1431	895	1487	823	653	1037	1012	824	855
Reference [Byte]	0	132	248	248	248	248	248	248	39
All [Byte]	1439	1035	1785	1231	1301	1293	1268	1080	902
Ratio [%]	0	28.1	-24	14.5	9.6	10.1	11.9	24.9	37.3

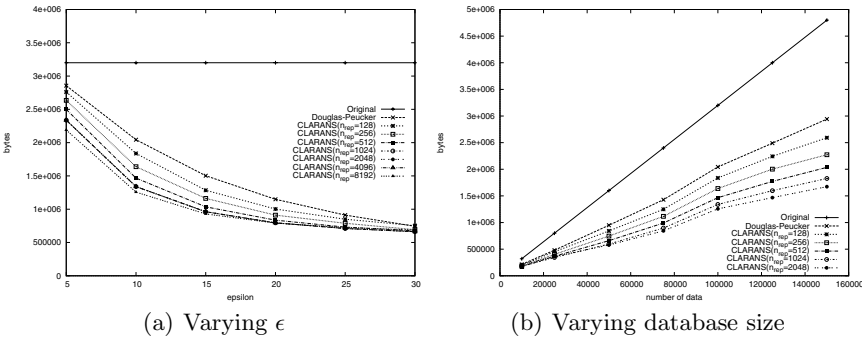


**Fig. 8.** The content of two sample clusters. All database shapes are normalized w.r.t. a shape  $o$  with  $o_{start} = (0, 0)$  and  $o_{end} = (1, 0)$ . Parameter:  $|DB| = 100,000$ ,  $n_{sp} = 4$ ,  $\epsilon = 10$ ,  $n_{rep} = 1024$ .

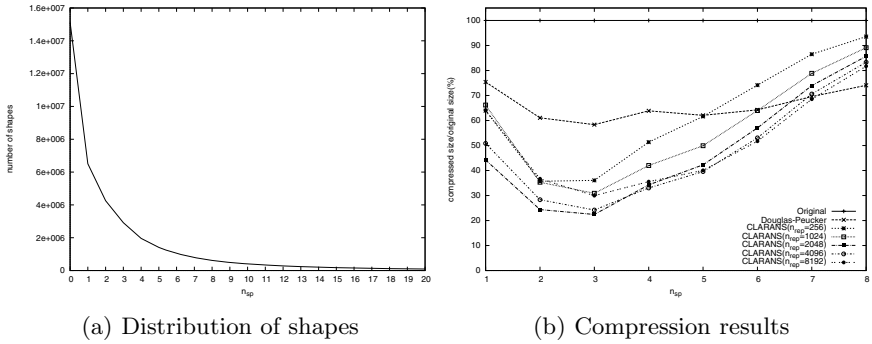
cluster 2 contains 95 shapes from which 58 can be substituted by a reference. Note that even if  $d_{err}(p^{-o}, s^{-o})$  is very high (cf. dashed line in Figure 8(b)), it is still possible that shape  $s$  can be compressed by pattern  $p$ , as long as  $|s|$  is very small.

Figure 9 compares the compression ratio achieved with our new technique to the traditional Douglas-Peucker (DP) approach. The picture shows that especially for small  $\epsilon$ -values, i.e., high precision maps, our approach clearly outperforms DP. This holds even if the number of representatives  $n_{rep}$  used is very small, e.g., 128. The higher the number of representatives, the better the compression ratio. For  $n_{rep}$  equal to 1024, i.e., 1% of the database are used as representatives, and for 150,000 database objects (cf. Figure 9(b)), our new approach leads to a 39% higher compression ratio than DP.

Because it is required to keep the pattern tables in main memory,  $n_{rep}$  should be as small as possible. Note that 90% of all shapes contain ten or fewer shape points (cf. Figure 5.2(a)). If only these shapes are compressed, only 10 pattern tables are needed. Assuming there are 1024 representatives in each of these



**Fig. 9.** Compression dependent on  $\epsilon$  and the database size. (a) Parameter:  $|DB| = 100,000$ ,  $n_{sp} = 4$ , (b) Parameter:  $\epsilon = 10$ ,  $n_{sp} = 4$ .



**Fig. 10.** Varying number of shape points ( $n_{sp}$ )

tables, then the main memory footprint would be  $10 \times \frac{1+10}{2} \times 8 \text{ Byte} \times 1024 = 440 \text{ KByte}$ , which is still acceptable even for embedded systems. Figure 5.2(b) shows that especially for small  $n_{sp}$  values our approach outperforms DP considerably. If the number  $n_{sp}$  of shape points gets very high, *e.g.*, 8, it is getting difficult to find suitable representatives due to the complexity and variety of the shapes.

To sum up, our new similarity based compression algorithm is well suited for the lossy compression of shapes, especially if the number of shape points is small.

## 6 Summary

In this paper, we presented new approaches for the compression of digital road networks consisting of both topological and geometrical information. For the compression of the topological information, we adapted existing solutions for the minimum bandwidth problem. The proposed approach allows encoding each edge with a small and constant number of bits. Besides this lossless compression of topological information, we also presented a lossless compression of the positions of nodes. The presented solution is based on minimum spanning trees that are computed on sparse road network graphs. Finally, for the lossy compression of road shapes, we exploited similarities between all shapes in the database. Based on a suitable non-symmetric distance measure, the clustering algorithm CLARANS groups shapes together trying to minimize a new compression related quality criterion. Our experimental evaluation on real-world European road network data demonstrates that the presented techniques lead to an overall size reduction of more than 30% compared to state of the art approaches currently used in digital map databases.

In our future work, we plan to apply our lossy geometrical shape compression to arbitrary polygons stored in digital map databases like lakes, parks, and build-up areas.



## References

1. Williams, H.E., Zobel, J.: Compressing integers for fast file access. *The Computer Journal* 42(3), 193–201 (1999)
2. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer* 10(2), 112–122 (1973)
3. Papadimitriou, C.H.: The np-completeness of the bandwidth minimization problem. *Computing* 16, 263–270 (1976)
4. Garey, M., Graham, R., Johnson, D., Knuth, D.: Complexity results for bandwidth minimization. *SIAM Journal of Applied Mathematics* 34(3), 477–495 (1978)
5. Caprara, A., Salazar-Gonzalez, J.-J.: Laying out sparse graphs with provably minimum bandwidth. *Inform. Journal on Computing* 17(3), 356–373 (2005)
6. Campos, V. Piñana, E., Martí, R.: Adaptive memory programming for matrix bandwidth minimization. Technical Report, University of Valencia, Spain (2006)
7. Gibbs, N., Poole, W., Stockmeyer, P.: An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM J. Numer. Anal.* 13(2), 235–251 (1976)
8. Orenstein, J.: A comparison of spatial query processing techniques for native and parameter spaces. In: *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data, Atlantic City, New Jersey, United States*, pp. 343–352. ACM Press, New York, NY, USA (1990)
9. Jagadish, H.V.: Linear clustering of objects with multiple attributes. In: *SIGMOD '90: Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pp. 332–342. ACM Press, New York, NY, USA (1990)
10. Ng, R.T., Han, J.: Clarans: A method for clustering objects for spatial data mining. *IEEE Transactions on Knowledge and Data Engineering* 14(5), 1003–1016 (2002)
11. Pruefer, H.: Neuer beweis eines satzes uber permutationen. *Archiv fur Mathematik und Physik* 27, 142–144 (1918)
12. Brecheisen, S., Kriegel, H.P., Kroger, P., Pfeifle, M.: Visually mining through cluster hierarchies. In: Jonker, W., Petković, M. (eds.) *SDM 2004. LNCS*, vol. 3178, pp. 400–412. Springer, Heidelberg (2004)
13. Ankerst, M., Breunig, M., Kriegel, H.-P., Sander, J.: Optics: ordering points to identify the clustering structure. In: *SIGMOD '99: Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, pp. 49–60. ACM Press, New York, NY, USA (1999)

# Traffic Density-Based Discovery of Hot Routes in Road Networks<sup>\*</sup>

Xiaolei Li, Jiawei Han, Jae-Gil Lee, and Hector Gonzalez

University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

**Abstract.** Finding hot routes (traffic flow patterns) in a road network is an important problem. They are beneficial to city planners, police departments, real estate developers, and many others. Knowing the hot routes allows the city to better direct traffic or analyze congestion causes. In the past, this problem has largely been addressed with domain knowledge of city. But in recent years, detailed information about vehicles in the road network have become available. With the development and adoption of RFID and other location sensors, an enormous amount of moving object trajectories are being collected and can be used towards finding hot routes.

This is a challenging problem due to the complex nature of the data. If objects traveled in organized clusters, it would be straightforward to use a clustering algorithm to find the hot routes. But, in the real world, objects move in unpredictable ways. Variations in speed, time, route, and other factors cause them to travel in rather fleeting “clusters.” These properties make the problem difficult for a naive approach. To this end, we propose a new density-based algorithm named *FlowScan*. Instead of clustering the moving objects, road segments are clustered based on the density of common traffic they share. We implemented *FlowScan* and tested it under various conditions. Our experiments show that the system is both efficient and effective at discovering hot routes.

## 1 Introduction

In recent years, analysis of moving object data [7] has emerged as a hot topic both academically and practically. In particular, the tracking of moving objects in road networks is becoming quite popular. GPS devices embedded in vehicles or RFID sensors on the streets can track a vehicle as it moves throughout the city traffic grid. There are many useful applications with such data. For instance, the OnStar system in General Motors vehicles notifies police of the vehicle’s GPS location when a crash is detected. E-ZPass sensors (using RFID technology) automatically pay tolls so traffic is not disturbed. GPS navigation systems offer driving directions in real-time. On a more aggregate level, average speeds or

---

<sup>\*</sup> The work was supported in part by Boeing company and the U.S. National Science Foundation NSF IIS-05-13678/06-42771, and NSF BDI-05-15813. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of the funding agencies.

traffic density is used to update driving time estimates in real-time or warn police of potential problem areas.

In this work, we address the problem of discovering *hot routes* in a road network. Informally, a hot route is a general traffic flow pattern. For example, “Many people in Oakland travel westbound on the Bay Bridge to reach downtown San Francisco at 7:30am.” The set of hot routes offers direct insight into the city’s traffic patterns. City officials can use them to improve traffic flow. Store owners and advertisers can use them to better position their properties. Police officials can use them to maximize patrol coverage.

*Example 1.* Figure 1(a) shows live traffic data<sup>1</sup> in the San Francisco Bay Area on a weekday at approximately 7:30am local time. Different colors show different levels of congestion (e.g., red/dark is heavy congestion). 511.org in the Bay Area gathers such data in real-time from RFID transponders located inside vehicles<sup>2</sup>. A likely hot route in Figure 1(a) is  $A \rightarrow B$  (i.e., highway CA-101).  $A$  is near the San Francisco International Airport.  $B$  is near the San Mateo Bridge. Figure 1(b) shows a closeup view of location  $B$ . Three additional locations  $x$ ,  $y$ , and  $z$  are shown. Without actually observing the flow of traffic, it is unclear whether  $y \rightarrow x$  is a hot route, or  $y \rightarrow z$ , or  $x \rightarrow z$ . FlowScan aims to solve this problem. ■

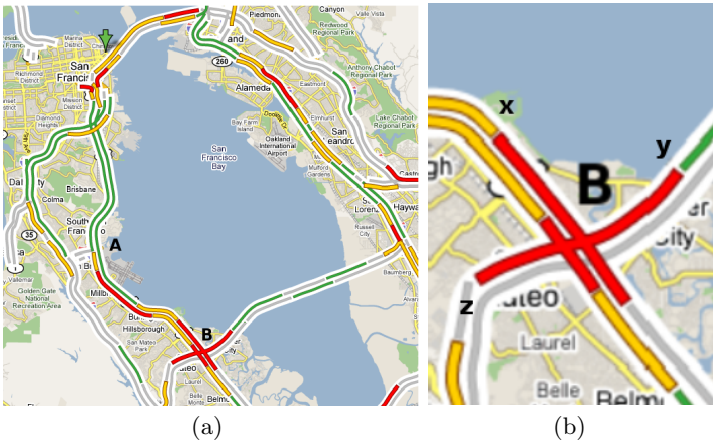


Fig. 1. Snapshot of San Francisco Bay Area traffic

At first glance, this may seem like an easy problem. A quick look at Figure 1(a) shows the high traffic roads in red. With some domain knowledge, we know that San Francisco, Oakland, and other densely populated regions are likely to be sources and destinations of traffic since many people live and work there. However, such domain knowledge is not always available. Additionally,

<sup>1</sup> <http://maps.google.com>

<sup>2</sup> <http://www.bayareafastrak.org>

real world traffic is a very *complex* data source. Objects do not travel in organized clusters. Two objects traveling from the same place to another place may take just slightly different routes at slightly different speeds and times. Random traffic conditions, such as a traffic accident or a traffic light, can cause even more deviations. Furthermore, hot routes do not have to be disjoint. Highways or major roads are popular pathways and several hot routes can share them. As a result, the mining algorithm must be robust to the variations within a hot route and amongst a set of hot routes.

We now state our problem as follows: *Given a set of moving object trajectories in a road network, find the set of hot routes.* A road network is represented by a graph  $G(V, E)$ .  $E$  is the set of directed edges, where each one represents the smallest unit of road segment.  $V$  is the set of vertices, where each one represents either a street intersection or important landmark.  $T$  is the set of trajectories, and each trajectory consists of an ID (*tid*) and a sequence of edges that it traveled through:  $(tid, \langle e_1, \dots, e_k \rangle)$ , where  $e_i \in E$ . Objects can only move on  $E$  and must travel the entirety of an edge.  $T$  is assumed to be collected from a similar time window; otherwise, different time windows might blur meaningful hot routes.

Informally, a *hot route* is a general path in the road network which contains heavy traffic. It represents a general flow of the objects in the network. Formally, it is a sequence of edges in  $G$ . The edges need not to be adjacent in  $G$ , but they should be near each other. Further, a sequence of edges in a hot route should *share a high amount of traffic between them*.

The rest of the paper is organized as follows. Section 2 gives an overview of the proposed solution and also alternative approaches. Section 3 lists some typical traffic behaviors and how they can confuse the naïve approaches. Section 4 describes the algorithm. Section 5 shows experiment results. Related work is discussed in Section 6. And finally, we conclude the study in Section 7.

## 2 Solution Overview

FlowScan extracts hot routes using the density of traffic on edges and sequences of edges. Intuitively, an edge with heavy traffic is potentially a part of a hot route. Edges with little or no traffic can be ignored. Also, two near-by edges that share a high amount of traffic between them are likely to be a part of the same hot route. This implies that the objects traveled from one edge to the other in a sequence. And lastly, a chain of such edges is likely to be a hot route.

We also list some possible alternative methods from related fields.

**Alternative Method 1:** Moving object clustering [6] discovers groups of objects that move together. The trajectory of each cluster can be marked as a hot route. We call this class of approaches *AltMoving*.

**Alternative Method 2:** Simple graph linkage is another possible approach. One could gather all the edges in  $G$  with heavy traffic and connect them via their graph connectivity. Then, each connected component is marked as a hot route. We call this class of approaches *AltGraph*.

**Alternative Method 3:** Trajectory clustering [10] discovers groups of similar sub-trajectories from the whole trajectories of moving objects. Each resultant cluster is marked as a hot route. We call this class of approaches *AltTrajectory*.

*FlowScan* and the three alternative methods offer very different approaches to the same data. One could view them in a spectrum. At one end of the spectrum are *AltMoving* and *AltTrajectory* where attention is paid to the individual objects. This is helpful in problems where the goal is to identify behaviors of individuals. At the other end of the spectrum is *AltGraph* where attention is paid to the aggregate. That is, objects' trajectories are aggregated into summaries and analysis is performed on the summary. This is helpful in problems where the goal is to learn very general information about the data. *FlowScan* can be viewed as an intermediate between these two extremes. The behaviors of the individuals (specifically, the common traffic between sequences of edges) are retained and affect high-level analysis about aggregate behavior.

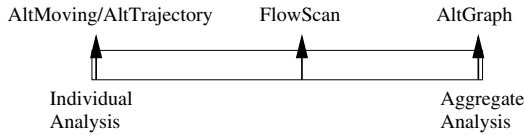


Fig. 2. Spectrum view of *FlowScan* and alternative methods

### 3 Traffic Behavior in Road Networks

In this section, we list some common real world traffic behaviors and examine how *FlowScan* and the alternative approaches can handle them.

#### 3.1 Traffic Complexity

A major characteristic of real world traffic is the amount of complexity. Instead of neat clusters, objects travel with different speeds and times even when they are on the same route. For example, in a residential neighborhood, many people leave for work in the morning and travel to the business district using approximately the same route. However, it is very unlikely that a group will leave at the same time and also travel together all the way to their destination. Various events (e.g., traffic light) can easily split them up.

Algorithms in the *AltMoving* class will not work very well with such complex data. Clusters in the technical sense only last for a short period of time or short distance. The same is true for *AltTrajectory* if speed/time is encoded into the trajectories. These algorithms lack *aggregate analysis* and as a result, they are likely to find too many short clusters and miss the overall flow. *FlowScan* connects road segments by the amount of traffic they share. So even if the objects change slightly or if the objects do not travel in compact groups, the amount of common traffic between consecutive edges in a hot route will still be high.

### 3.2 Splitting/Joining Hot Routes

Figure 3 shows a sample city traffic grid. The shade on each road segment indicates the amount of traffic on it; the darker the shade, the heavier the traffic. Suppose the two correct hot routes are  $A \rightarrow B \rightarrow C$  and  $A \rightarrow B \rightarrow D$ . Figure 3 shows a splitting of traffic at node  $B$ : some objects which moved from  $A$  to  $B$  go to  $C$  while others go to  $D$ . There are also other objects which move from  $C$  to  $D$  and vice-versa. This situation is very common in real world traffic.  $B$  could be the location in Chicago where I-90/94 splits into I-90 and I-94.

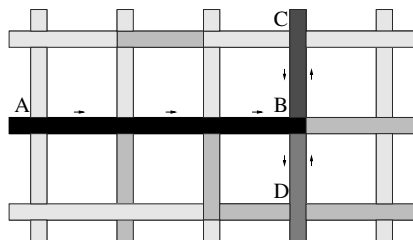


Fig. 3. Splitting hot routes:  $A \rightarrow B \rightarrow C$  and  $A \rightarrow B \rightarrow D$

With AltGraph, since all edges from  $A$  to  $C$  and  $D$  are connected, they would be incorrectly identified as a single big hot route. Notice that there is no *individual analysis* in AltGraph. With AltTrajectory,  $A \rightarrow B$  and  $B \rightarrow C$  will not be joined together because the physical similarity between them is low (i.e., hard left turn). Likewise for  $A \rightarrow B \rightarrow D$ . This flaw exists for the joining of traffic as well. In other words, if the arrows in Figure 3 were reversed, it would illustrate the problem of two hot routes joining at  $B$ .

In FlowScan, edges  $B \rightarrow C$  and  $B \rightarrow D$  will not be connected directly because they do not share any traffic. This is because physically, objects have to choose between the two edges and cannot travel on both. Further,  $A \rightarrow B$  will be connected to both  $B \rightarrow C$  and  $B \rightarrow D$  because it shares traffic with both.

### 3.3 Overlapping Hot Routes

In addition to splits or joins, two hot routes may overlap each other. Figure 4 shows an example with two distinct hot routes:  $A \rightarrow B$  and  $B \rightarrow C$ . Situations like this are common. Suppose  $B$  is a parking garage used by nearby residents during the night and incoming workers during the day. In the morning, residents drive out of the parking garage ( $B \rightarrow C$ ) and other people arrive from various locations to park ( $A \rightarrow B$ ).

Consider how AltGraph will handle this situation. Since  $A \rightarrow B$  and  $B \rightarrow C$  are connected in  $G$  at node  $B$ , the two hot routes will be joined together incorrectly. This is due to the lack of *individual analysis* on the edges during linking. The same happens with AltTrajectory, though for a different reason.  $A \rightarrow B$  and

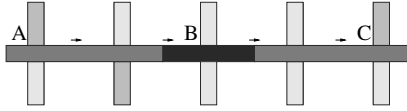


Fig. 4. Overlapping hot routes:  $A \rightarrow B$  and  $B \rightarrow C$

$B \rightarrow C$ 's shapes are similar and will be clustered together. With FlowScan, consecutive edges within a hot route must share a minimum number of common objects. If such edges were parts of different hot routes, this condition will not be satisfied, and thus, a single erroneous hot route will not be formed.

### 3.4 Slack Within Hot Routes

Figure 5 shows a hot route with some slight slack. A hot route exists from  $A$  to  $B$  in the grid. At the intermediate locations, objects are faced with different choices in order to reach  $B$ . Suppose the choices are essentially equivalent in terms of distance and speed and that traffic is split equally between them.

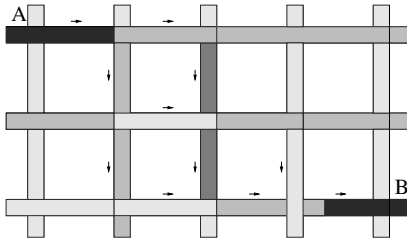


Fig. 5. Slack within a hot route:  $A \rightarrow B$

Consider how AltMoving will handle this deviation. Suppose the distance between the equivalent paths is larger than the maximum intra-cluster distance. This will cause the cluster at  $A$  to break into several smaller clusters when it reaches  $B$ . Next, consider AltGraph. Suppose the partitioning of traffic reduced the density on the intermediate edges to be below the “heavy” threshold. This would break the graph connectivity condition and miss the hot route from  $A$  to  $B$ . A similar error could occur with AltTrajectory if the traffic becomes too diluted between  $A$  and  $B$  or if the shapes become too dissimilar. With FlowScan, edge connectivity in  $G$  is not a required condition. Edges are connected in the hot route if they share common traffic and if they are near each other. Here, as long as  $A$  is within a given distance from  $B$ , the hot route will remain intact.

## 4 Density-Based Hot Route Extraction

In this section, we will give formal definitions of FlowScan, which uses traffic density information in road networks to discover hot routes.

### 4.1 Traffic-Density Reachability

**Definition 1 (Edge Start/End).** Given a directed edge  $r$ , let the  $start(r)$  be the starting vertex of the edge and  $end(r)$  be the ending vertex.

**Definition 2 (ForwardNumHops).** Given edges  $r$  and  $s$ , the number of forward hops between  $r$  and  $s$  is the minimum number of edges between  $end(r)$  and  $end(s)$  in  $G$ . It is denoted as  $ForwardNumHops(r, s)$ .

Recall that  $G$  is directed. This implies that an edge that is incident to  $start(r)$  in  $G$  will not have a  $ForwardNumHops$  value of 0 unless it is also incident to  $end(r)$ .

**Definition 3 (Eps-neighborhood).** The Eps-neighborhood of an edge  $r$ , denoted by  $N_{Eps}(r)$ , is defined by  $N_{Eps}(r) = \{s \in E \mid ForwardNumHops(r, s) \leq Eps\}$  where  $Eps \geq 0$ .

The Eps-neighborhood of  $r$  contains all edges that are within  $Eps$  hops away from  $r$ , in the direction of  $r$ . Semantically, this captures the flow of traffic and represents where objects are within  $Eps$  hops after they exit  $r$ . Figure 6 shows the 1-neighborhood of edge  $r$ .

Note that having the forward direction in the Eps-neighborhood makes the relation non-symmetric. In Figure 6 for example, the two edges in the circle are in the 1-neighborhood of  $r$ , but  $r$  is not in the 1-neighborhood of either of them. In fact, the only time when the Eps-neighborhood relation is symmetric is when two edges form a cycle within themselves. This is usually rare in road networks with one exception, and that is when one considers two sides of the same street. Figure 7 shows an example. In it,  $r_0$  is in the 1-neighborhood of  $r_1$  and vice-versa, because they form a cycle. This will happen for all two-way streets in the network. Though typically, an object will not travel on both sides of the same street within a trajectory. It could only happen with U-turns or if one end of the street is a dead end.

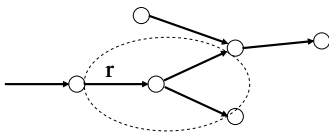


Fig. 6. 1-neighborhood of  $r$

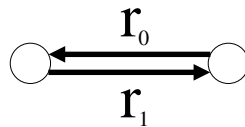


Fig. 7. Two sides of the same street

We did not choose a spatial distance function (e.g., Euclidean distance), because the number of hops better captures the notion of “nearness” in a transportation network. Consider a single road segment in the transportation network. If it were a highway, it might be a few kilometers long. But if it were a city block in downtown, it might be just a hundred meters. However, since an object has to travel the entirety of that edge, the two adjacent edges are the same “distance” apart no matter how long physically that intermediate edge is.



**Definition 4 (Traffic).** Let  $traffic(r)$  return the set of trajectories that contains edge  $r$ . Recall trajectories are identified by unique IDs.

**Definition 5 (Directly traffic density-reachable).** An edge  $s$  is directly traffic density-reachable from an edge  $r$  wrt two parameters, (1)  $Eps$  and (2)  $MinTraffic$ , if all of the following hold true.

1.  $s \in N_{Eps}(r)$
2.  $|traffic(r) \cap traffic(s)| \geq MinTraffic$

Intuitively, the above criteria state that in order for an edge  $s$  to be directly traffic density-reachable from  $r$ ,  $s$  must be near  $r$ , and  $traffic(s)$  and  $traffic(r)$  must share some non-trivial common traffic. The “nearness” between the two edges is controlled by the size of the  $Eps$ -neighborhood. This directly addresses the slack issues in Section 3.4. As long as the slack is not larger than  $Eps$ , two edges will stay directly connected via this definition.

The second condition of two edges sharing traffic is intuitive. It is also at the core of FlowScan. The flaw of methods in the AltGraph class is that aggregation on the edges has erased the identities of the objects. As a result, two edges with high traffic on them and near each other will look the same regardless if they actually *share* common traffic. By having the second condition rely on the common traffic, one can get a better idea of how objects actually move in the road network.

Directly traffic density-reachable is not symmetric for pairs of edges because the  $Eps$ -neighborhood is not symmetric. Though, for the same reasons that two edges might be in the  $Eps$ -neighborhood of each other, two edges could be directly traffic density-reachable from each other.

**Definition 6 (Route traffic density-reachable).** An edge  $s$  is route traffic density-reachable from an edge  $r$  wrt parameters  $Eps$  and  $MinTraffic$  if:

1. There is a chain of edges  $r_1, r_2, \dots, r_n, r_1 = r, r_n = s$ , and  $r_i$  is directly traffic density-reachable from  $r_{i-1}$ .
2. For every  $Eps$  consecutive edges (i.e.,  $r_i, r_{i+1}, \dots, r_{i+Eps}$ ) in the chain,  $|traffic(r_i) \cap traffic(r_{i+1}) \cap \dots \cap traffic(r_{i+Eps})| \geq MinTraffic$ .

Definition 6 is an extension of Definition 5. It states that two edges are route reachable if there is a chain of directly reachable edges in between and that if one were to slide a window across this chain, edges inside every window share common traffic. The sliding window directly address the overlapping behavior as described in Section 3.3. At the boundaries of two overlapping hot routes, the second condition will break down and thus break the overlapping hot routes into two. The  $Eps$  parameter is being reused here to control the width of a sliding window through the chain. The reuse is justified because their semantics are similar, but one could just as well use a separate parameter.

The reason for using a sliding window is based on our observation that a trajectory can contribute to only a portion of a hot route. This better matches real world hot route behavior. For example, a hot route exists from the suburb

to downtown in the morning. Figure 8 shows an illustration. However, most people do not travel the entirety of the hot route. More often, they live and work somewhere in between the suburb and downtown. But in the aggregate, a hot route exists between the two locations.

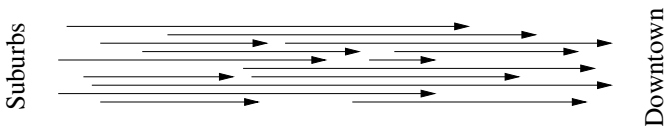


Fig. 8. Route traffic density-reachable

### 4.2 Discovering Hot Routes

The hot route discovery process follows naturally from Definition 6. It is an iterative process. Roughly, one starts with a random edge, expands it to a hot route, and repeats until no more edges are left. The question is then with which edge(s) should each iteration begin. To this end, we introduce the concept of a **hot route start**, which is the first edge in a hot route. Intuitively, an edge is a hot route start if none of its preceding directly traffic density-reachable neighbors are part of hot routes.

**Definition 7 (Hot Route Start).** *An edge  $r$  is a hot route start wrt  $MinTraffic$  if the following condition is satisfied.*

$$\left| traffic(r) \setminus \bigcup \{ traffic(x) \} \right| \geq MinTraffic$$

where  $\{x \mid end(x) = start(r) \wedge |traffic(x)| \geq MinTraffic\}$ .

The question is whether all hot routes begin from a hot route start. The following lemma addresses this.

**Lemma 1 (Hot Route Start).** *Hot routes must begin from a hot route start.*

**Proof:** There are two ways for a hot route to begin on an edge  $r$ . The first way is when  $MinTraffic$  or more objects start their trajectory at  $r$ . In this case, none of these objects will appear in  $start(r)$ 's adjacent edges because they simply did not exist then. As a result, the set difference will return at least  $MinTraffic$  objects and thus marking  $r$  as a hot route start. The second way is when  $MinTraffic$  or more objects converge at  $r$  from other edges. The source of traffic on  $r$  is exactly the set of edges adjacent to  $start(r)$ . Suppose one of these edges,  $x$ , contains more than  $MinTraffic$  objects on it. In this case,  $x$  is part of another hot route, and the objects that moved from  $x$  to  $r$  should not contribute to  $r$ . However, if it does not contain more than  $MinTraffic$  objects, it cannot be in a hot route and its objects are counted towards  $r$ . If more than  $MinTraffic$  objects are counted towards  $r$ , then it is the start of a hot route. ■

### 4.3 Algorithm

Definitions 6 and 7 form the foundation of the hot route discovery process. A simple approach could be to initialize a hot route to a hot route start and iteratively add all route traffic density-reachable edges to it. By repeating this process for all hot route starts in the data, one can extract all the hot routes. The question is then how to efficiently find all route traffic density-reachable edges given an existing hot route. If new edges are added in no particular order, then one would have to search through all existing edges in the hot route at every iteration. This is very inefficient. Further, if the hot route splits, it could become tricky if it is in the middle.

To alleviate this problem, we restrict the growth of a hot route to be only at the *last* edge. A hot route is a sequence of edges so the last edge is always defined. By growing the hot route at the end one edge at a time, only the *Eps*-neighborhood of the last edge needs to be extracted. This is much more efficient than extracting the *Eps*-neighborhoods of all edges in the hot route. This is still a complete search because all possible reachable edges are examined but just with some order. It is essentially a breadth-first search of the road network. Then for each neighboring edge, the *route* traffic density-reachability condition is checked against the last few edges of the hot route (*i.e.*, window). If the condition is satisfied, the edge is appended to the hot route; otherwise, the edge is ignored.

Sometimes, the number of directly traffic density-reachable edges from the last edge in the hot route is larger than one. There are two causes for this. The first cause is multiple edges within one hot route. This can happen when *Eps* is larger than 0, and multiple edges of the hot route are in the *Eps*-neighborhood. This can be detected by checking to see if the *start()*'s and *end()*'s match across edges. In this case, only the *nearest* edge is appended to the hot route. The other edges will just be handled in the next iteration. The second cause is when a hot route *splits*. In this case, the current hot route is duplicated, and a different hot route is created for each split. The difference between these two cases can be detected by checking the directly traffic density-reachability condition between edges in the *Eps*-neighborhood.

The overall algorithm proceeds as follows. First, all hot route starts are extracted from the data. This is done by checking Definition 7 for every edge in  $G$ . This step has linear complexity because only individual edges with their *Eps*-neighborhoods are checked. Then, for every hot route start, the associated hot routes are extracted. Algorithm 1 shows a pseudo-code description.

One point of concern is efficiency. Suppose the adjacency matrix or list of the road network fits inside main memory. Then, searching the graph is quite efficient. Retrieving the list of TID's at each edge will require disk I/O but traversing the graph will not. However, suppose the adjacency matrix is too big to fit inside main memory. In this case, we introduce two additional indexing structures to help the search process. Figure 9 shows an illustration.

All vertices of the road network are stored in a 2D index, *e.g.*, R-tree (Vertex Index Tree). All edges are stored on disk (Edge Table). Each edge record consists of the edge ID and its starting and ending vertices (each vertex is an  $(x,$

**Algorithm 1.** FlowScan**Input:** Road network  $G$ , object trajectory data  $T$ ,  $Eps$ ,  $MinTraffic$ .**Output:** Hot routes  $R$ 


---

```

1: Initialize  $R$  to  $\{\}$ 
2: Let  $H$  be the set of hot route starts in  $G$  according to  $T$ 
3: for every hot route start  $h$  in  $H$  do
4:    $r = \text{new Hot\_Route}$  initialized to  $\langle h \rangle$ 
5:   Add  $\text{Extend\_Hot\_Routes}(r)$  to  $R$ 
6: end for
7: Return  $R$ 

```

---

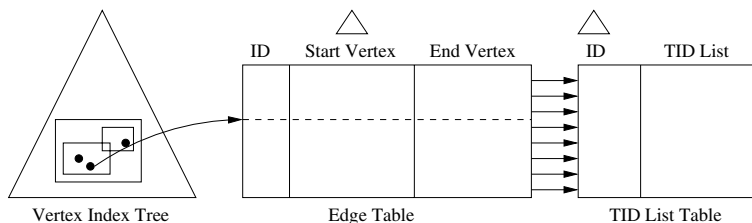
Procedure  $\text{Extend\_Hot\_Routes}(\text{hot route } r)$ 

```

1: Let  $p$  be the last edge in  $r$ 
2: Let  $Q$  be the set of directly traffic density-reachable neighbors of  $p$ 
3: if  $Q$  is non-empty then
4:   for every split in  $Q$  do
5:     if route traffic density-reachable condition is satisfied then
6:       Let  $r'$  be a copy of  $r$ 
7:       Append split's edges to  $r'$ 
8:        $\text{Extend\_Hot\_Routes}(r')$ 
9:     end if
10:  end for
11: else
12:   Return  $r$ 
13: end if

```

---

**Fig. 9.** Indexing structures of FlowScan

y) tuple). Using these two data structures, one can retrieve adjacent neighbors of an edge by querying the R-tree on the appropriate vertex and then retrieving the corresponding edges in the Edge Table. The R-tree is quite useful for finding adjacent neighbors of a specific edge since the coordinates of the adjacent neighbors tend to be close to that of the edge.

We note that the Edge Table may be accessed repeatedly to retrieve adjacent neighbors. This operation can be done more efficiently by exploiting locality. Specifically, if the physical locality of edges in the road network is preserved in the Edge Table, one can reduce the amount of disk I/Os. To this end, we create a *clustering* index on the Starting Vertex attribute of the Edge Table.

Assuming that each value is 4 bytes, each edge record is then 20 bytes. Then, if a page is 4K, it will contain approximately 200 edges. The intuition is that these 200 edges will be physically close to each other in the road network. Because FlowScan traverses through *Eps*-neighborhoods, it is highly likely that an edge and its neighbors will be stored on the same page in the Edge Table. In this case, disk I/O will be reduced because the page has already been fetched.

**Lemma 2 (Completeness).** *The set of hot routes discovered by FlowScan is complete and unique wrt. Eps and MinTraffic.*

**Proof:** The above assertion is easy to see because the construction algorithm uses the definition word-for-word, specifically Definition 6, to build the hot routes. Thus, given a hot route start, the set of hot routes extending from it is guaranteed to be found. The question is more about if the set of hot route starts found is complete. Because every edge in a hot route must satisfy the *MinTraffic* condition, there must be a “first” in a sequence. The set of hot route starts is simply these “firsts.” Lastly, ordering is not a factor in FlowScan, because no marking or removal is done to  $G$ . Thus, it does not matter in which order  $H$  is processed. ■

#### 4.4 Determining Parameters

There are two input parameters to the FlowScan algorithm: *Eps* and *MinTraffic*. The first parameter, *Eps*, controls how lax FlowScan can be between directly reachable edges. A value of 0 is too strict since it enforces strict spatial connectivity. A small value in the range of 2–5 is usually reasonable. In a metropolitan area, this corresponds to 2–5 city blocks; and in a rural area, this corresponds to 2–5 highway exists.

As for *MinTraffic*, this is often application or traffic dependent. “Dense” traffic in a city of 50,000 people is very different from “dense” traffic in a city of 5,000,000 people. In cases where domain knowledge dictates a threshold, that value can be used. If no domain knowledge is available, one can rely on statistical data to set *MinTraffic*. It has been shown that traffic density (and many other behaviors in nature) usually obeys the power law. That is, the vast majority of road segments have a small amount of traffic, and a relative small number have extremely high density. One can plot a frequency histogram of the edges and either visually pick a frequency as *MinTraffic* or use the parameters of the exponential equation to set *MinTraffic*.

## 5 Experiments

To show the effectiveness and efficiency of FlowScan, we test it against various datasets. FlowScan was implemented in C++ and all tests were performed on a Intel Core Duo 2 E6600 machine running Linux.

## 5.1 Data Generation

Due to the lack of real-world data, we used a network-based data generator provided by [13]. It uses a real-world city road network as the road network and generates moving objects on it. Objects are affected by the maximum speed on the road, the maximum capacity of the road, other objects on the road, routes, and other external factors.

The default generator provided generates essentially random traffic: an object's starting and end locations are randomly chosen within the network. In order to generate some interesting patterns, we modified how the generator chooses starting and end locations. Within a city network, "neighborhoods" are generated. Each neighborhood is generated by picking a random node and then expanding by a preset radius (3–5 edges). Moving objects are then restricted to start and end in neighborhoods.

Hot routes form naturally because of the moving object's preference for the quickest path. As a result, bigger roads (e.g., highways) are more likely to be chosen by the moving objects. However, if too many objects take a highway or a road, it will reach capacity and actually slow down. In such cases, objects will choose to re-route and possibly create secondary hot routes.

## 5.2 Extraction Quality

**General Results.** To check the effectiveness of FlowScan, we test it against a variety of settings. First, we present the results for two general cases. Figure 10 shows several routes extracted from 10,000 objects moving in the San Francisco bay area. 10 neighborhoods of radius 3 each were placed randomly in the map. *Eps* and *MinTraffic* were set to 2 and 300, respectively. Each hot route is drawn in black with an arrow indicating the start and a dot indicating the end. The gray lines in the figures indicate all traffic observed in the input data (not the entire city map).

Even though the neighborhoods were completely random, we get realistic hot routes in this experiment. The hot routes in Figure 10(a) and 10(b) are CA-101 connecting San Francisco and San Jose, a major highway in the area. Figures 10(c) and 10(d) correspond to the Golden Gate Bridge connecting the city of San Francisco to the north. One of the random neighborhoods must have been across the bridge so objects had no choice but to use the bridge. Figure 10(e) shows a hot route connecting Oakland to that same neighborhood across the Richmond-San Rafael Bridge. Lastly, Figure 10(f) corresponds to a hot route connecting approximately Hayward to San Jose via I-880.

Next, Figure 11 shows three hot routes extracted from 5,000 objects moving in the San Joaquin network. Three neighborhoods were picked in this network, each with radius of 3. *Eps* and *MinTraffic* were set to 2 and 400, respectively. In Figures 11(a) and 11(b), the horizontal portions of the hot routes correspond to I-205. In Figure 11(b), the vertical portion corresponds to I-5. Both these roads are major interstate highways. The roads in Figure 11(c) are W. Linne Rd

<sup>3</sup> <http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/>

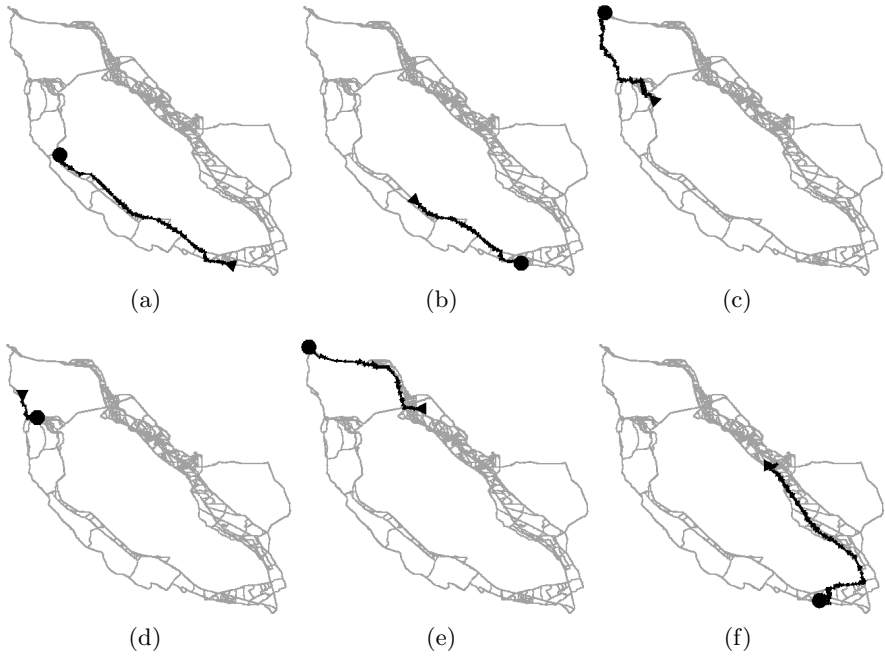


Fig. 10. Hot routes in San Francisco data map

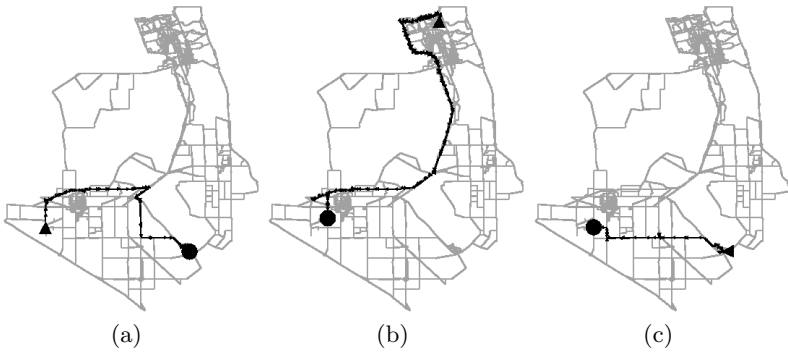


Fig. 11. Hot routes in San Joaquin data map

and Kason Rd. By looking at the city map, we observe that they make up the quickest route between the two neighborhoods.

**Splitting Hot Route Behavior.** We also tested FlowScan with some specific traffic behaviors. First, we test the case of a hot route splitting into two. This data set is generated by setting the number of neighborhoods in a road network to three and fixing the start node to be in one of the three. Because start and

destination neighborhoods cannot be the same, this forces the objects (1000 of them) to travel to one of two destinations. And because the objects like to travel on big roads (due to speed preference), they will usually leave the starting neighborhood using the same route regardless of the final destination and split sometime later.

Figures 12(a) and 12(b) shows the two hot routes extracted from the data. Both hot routes start at the green arrow at the lower right, move to the middle, and the split according to their final destinations. In Figure 12(c), the result from an AltGraph algorithm is shown. All edges that exceed the *MinTraffic* threshold (100) are connected if they are adjacent in the road network. Obviously, the two hot routes are connected together because the underlying objects are not considered. Figure 12(d) shows the result from a AltTrajectory algorithm [10]. In it, 14 clusters were found. Because shape is a major factor in trajectory clustering, the routes were broken into different clusters. The split is “detected” simply due to the hard left-turn shape, but the routes are not intact. One could post-process the results and merge near-by clusters, but this could run into the same problems as AltGraph since individual trajectories are ignored.

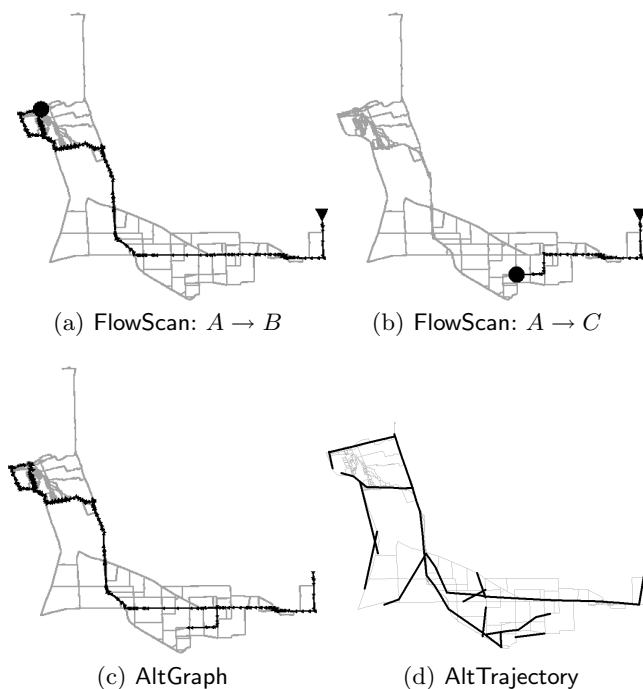


Fig. 12. Splitting hot routes

**Overlapping Hot Route Behavior.** Next, we test the case of two hot routes overlapping. That is, one starts at the same place as where the other one ends. To generate this data set, we also set the number of neighborhoods to three. Let



them be known as  $A$ ,  $B$ , and  $C$ . Then, for half of the objects, their paths are  $A \rightarrow B$ ; and for the other half, their paths are  $B \rightarrow C$ . We set the radius of neighborhood  $B$  to 0 to ensure that the two hot routes overlap.

Figure 13 shows the results of this test. As the graphs show, two hot routes were extracted. Figure 13(c) shows a result with an AltGraph algorithm. Although  $B \rightarrow C$  (not shown) is correctly extracted in that algorithm,  $A \rightarrow B$  is not. It is incorrectly linked together with  $B \rightarrow C$  and erroneously forms  $A \rightarrow B \rightarrow C$ . This is because individual identities are not considered in the algorithm. Figure 13(d) shows the result of AltTrajectory. 13 clusters were discovered. Again, the routes are not intact. But more seriously, the trajectories near  $B$  are clustered into a single cluster because their shapes are similar.

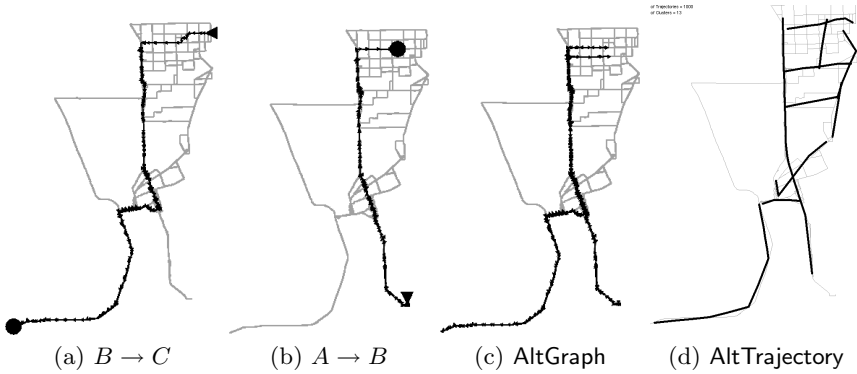


Fig. 13. Overlapping hot routes

### 5.3 Efficiency

Finally, we test the efficiency of FlowScan with respect to the number of objects. Figure 14 shows the running time as the number of objects increases from 2,000 to 10,000 with *MinTraffic* set to 10%. All objects were stored in memory, and time to read the input data is excluded. As the curve shows, running

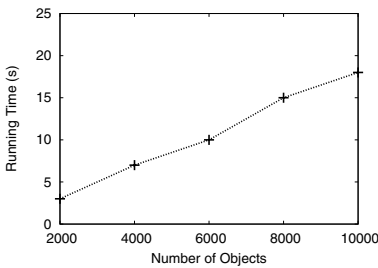


Fig. 14. Efficiency with respect to number of objects

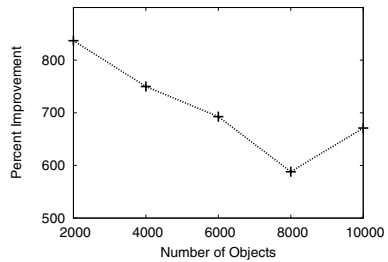


Fig. 15. Disk I/O improvement of clustered index on Edge Table

time increases linearly with respect to the number of objects. Next, we test the difference in disk I/O using a clustered Edge Table vs. an unclustered Edge Table. Figure 15 shows the result. Pages were set to 4K each and a buffer of 10 pages was used. We excluded the I/Os of the Vertex Index Tree and the TID List Table since they are the same in both cases. The figure shows the percent improvement of the clustered Edge Table. It is a significant improvement ranging from 588% to over 800%. This value is relatively stable because the percent improvement depends more on the structure of the network than the number of objects.

## 6 Related Work

Work in moving object clustering is closely related to FlowScan. Some examples include [6,218]. In [8], objects are grouped using a traditional clustering algorithm at each time snapshot and then linked together over time to form moving clusters. The assumption is that clusters will be stable across short time periods. In real world traffic, one can see how a traffic light or a incoming traffic from a highway on-ramp can easily breakup clusters between consecutive snapshots. As a result, [8] and similar approaches will likely find too many short clusters and miss the overall flow.

Work in the AltTrajectory class is also related. They include trajectory clustering [18,10] and trajectory modeling [11,9]. In both types, the focus is on individual objects, not aggregate. Further, most algorithms deal with free-moving objects and consider the shape in clustering. This is irrelevant in road networks. Also, the common traffic between sub-trajectories is ignored in the analysis. This could cause problems when hot routes merge or split or make drastic turns.

Data mining in spatiotemporal data is also related to our work. One class of problems mines sequential patterns of events (e.g., temperature) at spatial locations [15]. Another problem is co-location mining [14,17,19]. A co-location rule states a set of locations that often occur together with respect to a neighborhood function. These work have a similar spirit of discovering frequent patterns, but they are different in that the input is not trajectory data. General data mining in sequential pattern mining [12] is another related area. Hot routes are similar to sequential patterns in trajectories. However, spatial information and traffic information are not considered in traditional data mining.

General moving object database research has work related to indexing [13,5,7] and similarity search [16,3]. But the focus of such work is on the raw edges, shapes, locations, etc. FlowScan focuses on a higher level problem.

Our definitions of density is similar in spirit to density-based clustering (DBSCAN [4]), also used in [10]. But the nature of the data is very different. A typical clustering algorithm is concerned with discovering clusters of points in spatial data, while FlowScan is concerned with discovering hot routes in traffic.

## 7 Conclusion and Future Work

In this study, we have examined the problem of discovering hot routes in road networks. Due to the complexity of the data, this is a problem not easily solved by existing algorithms in related areas. We show several typical traffic behaviors that are tricky to handle. To this end, we propose a new algorithm, **FlowScan**, which uses the density of traffic in sequences of road segments to discover hot routes. It is a robust algorithm that can handle the complexities in the data and we verify through extensive experiments. By comparing against other approaches, we see the advantages of this approach.

One important aspect of the trajectory data we did not utilize in **FlowScan** is the non-spatiotemporal information about the trajectories. The type of the vehicle is one such example. The hot routes of sedans are sure to be different from the hot routes of transport trucks. Other attributes on the data facilitates a multi-dimensional approach to this problem. By knowing the correlations between hot routes and other attributes, one can enhance the usefulness of the discovered information.

## References

1. Brinkhoff, T.: A framework for generating network-based moving objects. In: *GeoInformatica'02* (2002)
2. Cadez, I.V., Gaffney, S., Smyth, P.: A general probabilistic framework for clustering individuals and objects. In: *KDD'00* (2000)
3. Chen, L., Ozsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: *SIGMOD'05* (2005)
4. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases. In: *KDD'96* (1996)
5. Frentzos, E.: Indexing objects moving on fixed networks. In: Hadzilacos, T., Manolopoulos, Y., Roddick, J.F., Theodoridis, Y. (eds.) *SSTD 2003*. LNCS, vol. 2750, Springer, Heidelberg (2003)
6. Gaffney, S., Smyth, P.: Trajectory clustering with mixtures of regression models. In: *KDD'99* (1999)
7. Güting, G.H., Schneider, M.: *Moving Objects Databases*. Morgan Kaufmann, San Francisco (2005)
8. Kalnis, P., Mamoulis, N., Bakiras, S.: On discovering moving clusters in spatio-temporal data. In: Bauzer Medeiros, C., Egenhofer, M.J., Bertino, E. (eds.) *SSTD 2005*. LNCS, vol. 3633, Springer, Heidelberg (2005)
9. Kostov, V., Ozawa, J., Yoshioka, M., Kudoh, T.: Travel destination prediction using frequent crossing pattern from driving history. In: *ITSC'05* (2005)
10. Lee, J., Han, J., Whang, K.: Trajectory clustering: A partition-and-group framework. In: *SIGMOD'07* (2007)
11. Liao, L., Fox, D., Kautz, H.: Learning and inferring transportation routines. In: *AAAI'04* (2004)
12. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.-C.: Mining sequential patterns by pattern-growth: The prefixspan approach. *TKDE'04* (2004)

13. Pfoser, D., Jensen, C.S.: Indexing of network constrained moving objects. In: GIS'03 (2003)
14. Shekhar, S., Huang, Y.: Discovering spatial co-location patterns: A summary of results. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, Springer, Heidelberg (2001)
15. Tsoukatos, I., Gunopulos, D.: Efficient mining of spatiotemporal patterns. In: Jensen, C.S., Schneider, M., Seeger, B., Tsotras, V.J. (eds.) SSTD 2001. LNCS, vol. 2121, Springer, Heidelberg (2001)
16. Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In: ICDE'02
17. Yoo, J.S., Shekhar, S.: A partial join approach to mining co-location patterns. In: GIS'04 (2004)
18. Zen, H., Tokuda, K., Kitamura, T.: A viterbi algorithm for a trajectory model derived from hmm with explicit relationship between static and dynamic features. In: ICASSP '04 (2004)
19. Zhang, X., Mamoulis, N., Cheung, D.W., Shou, Y.: Fast mining of spatial collocations. In: KDD'04 (2004)

# Spatio-temporal Network Databases and Routing Algorithms: A Summary of Results

Betsy George\*, Sangho Kim, and Shashi Shekhar

Department of Computer Science and Engineering, University of Minnesota  
200 Union St SE, Minneapolis, MN 55455, USA  
{bgeorge,sangho,shekhar}@cs.umn.edu  
<http://www.spatial.cs.umn.edu/>

**Abstract.** Spatio-temporal networks are spatial networks whose topology and parameters change with time. These networks are important due to many critical applications such as emergency traffic planning and route finding services and there is an immediate need for models that support the design of efficient algorithms for computing the frequent queries on such networks. This problem is challenging due to potentially conflicting requirements of model simplicity and support for efficient algorithms. Time expanded networks which have been used to model dynamic networks employ replication of the network across time instants, resulting in high storage overhead and algorithms that are computationally expensive. In contrast, proposed time-aggregated graphs do not replicate nodes and edges across time; rather they allow the properties of edges and nodes to be modeled as a time series. Since the model does not replicate the entire graph for every instant of time, it uses less memory and the algorithms for common operations (e.g. connectivity, shortest path) are computationally more efficient than those for time expanded networks. One important query on spatio-temporal networks is the computation of shortest paths. Shortest paths can be computed either for a given start time or to find the start time and the path that leads to least travel time journeys (best start time journeys). Developing efficient algorithms for computing shortest paths in a time varying spatial network is challenging because these journeys do not always display greedy property or optimal substructure, making techniques like dynamic programming inapplicable. In this paper, we propose algorithms for shortest path computations in both contexts. We present the analytical cost models for the algorithms and provide an experimental comparison of performance with existing algorithms.

**Keywords:** time-aggregated graphs, shortest paths, spatio-temporal data bases.

## 1 Introduction

The underlying data of interest for many significant applications such as transportation networks is structured as a spatio-temporal network, which consists

---

\* Corresponding author.

of a finite collection of points (i.e. nodes) with location information, the line-segments (i.e. edges) connecting the points, and the time-varying attributes attached to the elements. For example, a spatio-temporal network database for a traveler's trip planning may store the intersections as nodes, the road segments as edges, and time dependent travel time attached to the road segments. In the case of evacuation planning, time dependent capacity may be added to the road segments as another important attribute.

Related work in the field of databases falls into three broad categories (1) Spatial network databases, (2) Graph Databases, and (3) Spatio-temporal databases. The recent release of Oracle (version 10g) includes a network data model to store and maintain the connectivity of link-node networks and supports basic features such as shortest path [14]. The Network Analyst extension of ArcMap from ESRI supports a network geodatabase and provides basic algorithms (e.g., shortest path, service area, closest facility, etc.) [7]. However, these products do not address the time variance of spatial networks, which is crucial in applications such as route computations and emergency planning.

Graph databases [5,6,7,19,22,24] also primarily deal with spatial networks that do not vary with time. Research in graph databases that accounts for temporal variations perform computations over a snapshot of the network [4,9,18], and does not consider the interplay between the edge travel times and the existence of edges. For example, Ding [4] proposed a model that addresses the time-dependency by associating a temporal attribute to every edge and node of the network so that its state at any instant of time can be retrieved. This model performs path computations over a snapshot of the network. Since the network can change over the time taken to traverse these paths, this computation might not give realistic solutions. The model does not propose an algorithm for the least travel time paths.

Although the need for live traffic information is increasing, there has been little work on the modeling and algorithms for spatio-temporal network databases. Chorochronos [12], studied various aspects of spatio-temporal databases including ontology, modeling, and implementation. However, the researchers have yet to study spatio-temporal networks in this framework.

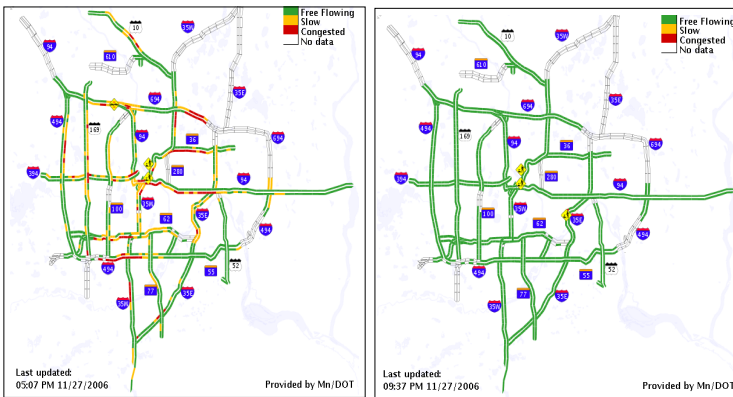
Research in Operations Research is based on the time expanded network [10,11,13,15,17,21]. This model duplicates the original network for each discrete time unit  $t = 0, 1, \dots, T$  where  $T$  represents the extent of the time horizon. The expanded network has edges connecting a node and its copy at the next instant in addition to the edges in the original network, replicated for every time instant. This significantly increases the network size and is very expensive with respect to memory. Because of the increased problem size due to replication of the network, the computations become expensive.

As the first step towards the study of spatio-temporal network databases, we previously proposed a spatio-temporal network model named the time aggregated graph [8]. In this paper, we introduce a case study of this model using routing algorithms. The proposed algorithms (SP-TAG and BEST) compute the shortest path in the given network for a given start time at the source node and

the least travel time route over the entire time period. The proposed model and algorithms are evaluated with a real world static graph appended with a synthetically generated travel time series.

### 1.1 An Illustrative Application Domain

Transportation networks are the kernel framework of many advanced transportation systems such as the Advanced Traveler Information System and Intelligent Vehicle Highway Systems. Transportation networks are spatio-temporal in nature and require significant database support to handle the storage of their large amounts of multi-dimensional data. Many important applications based on transportation networks, including travelers' trip planning, consumer business logistics, and evacuation planning need to be built upon spatio-temporal network databases. For example, commuters try to find a suitable time to start their commute so that they spend the least time in traffic. Figure 1 illustrates traffic sensor networks on urban highways which measure congestion levels at two different times (e.g. 5:07pm and 9:37pm) illustrating possible changes in shortest route travel times at different times of the day. With the increasing use of sensor networks to monitor traffic data on spatial networks and the subsequent availability of time-varying traffic data, it becomes important to incorporate this data in the models and algorithms related to transportation networks. However, existing spatio-temporal databases do not offer adequate support for spatio-temporal networks.



**Fig. 1.** Sensor networks periodically report time-variant traffic volumes on Twin Cities highways (Best viewed in color, Source: Mn/DOT)

The problem of finding best start time has similar applications in freight delivery services, one of whose main concerns is to reduce logistic costs such as fuel consumption. Another important application is in emergency traffic management. Emergencies caused by natural or manmade disasters can result in atypical

demands on a transportation network, resulting in severe congestion. Emergency managers may be interested in using spatio-temporal network databases to understand non-equilibrium traffic dynamics and to make informed decisions about evacuation route planning.

## 1.2 Broad Challenges

A time-variant graph is a graph whose edge and node properties and topological structure are time dependent. For example, traffic volume on urban highways varies over the time of day, which leads to a variation in travel time. In addition to network parameter values, the network topology can also change with time due to the unavailability of certain road segments during some periods of time due to repair or natural calamities. Conventional graph algorithms cannot easily be applied to the snapshot graphs at discrete time instants to evaluate frequent queries without accounting for relationships among snapshots. However, time-variant graphs raise many challenges for database research. Due to their potentially large and evergrowing sizes, a storage-efficient representation is critical to reduce and possibly eliminate redundant information across different time-points. Second, new data model concepts need to be investigated to represent and classify potentially new alternative semantics for common graph operations such as shortest-path and connectivity. For example, a shortest path between a given pair of nodes may have at least two interpretations, one for a given start time-point and the other for the shortest travel-time for any start time in a given time interval. A third challenge is the design of efficient and correct query processing strategies and algorithms since some of the commonly assumed graph-properties may not hold for spatio-temporal graphs. For example, consider the optimal substructure (required in dynamic programming, [2]) for shortest paths in a graph. While each prefix path (path from a source node to an intermediate node in an optimal path) is optimal in a static graph, it may not be optimal in a spatio-temporal graph due to a potential wait at an intermediate node.

**Our Contribution:** The paper describes a model for spatio-temporal networks called the time aggregated graph, that uses a time series to represent time-varying attributes. We propose algorithms to compute shortest paths for a fixed start time and the best start time (Best Start Time Algorithm) and consequently the least commute time paths. These problems are challenging since common algorithm design techniques like greedy design cannot always be applied. The Best Start Time algorithm uses a node cost time series instead of a scalar node cost. The entries in the time series are updated when a path of smaller cost is found. The algorithm iterates until every entry reaches a minimum value and hence does not depend on the greedy choice property. This removes the FIFO restriction from the edge travel times. We also present the experimental analysis of the best start time algorithm and the shortest path algorithm for a given start time [8].



### 1.3 Scope and Outline of the Paper

The paper presents a case study of time aggregated graphs using routing algorithms to compute shortest paths in two different contexts. Shortest paths can be computed from a given source node for a fixed start time and at the best start time which minimizes the travel time over the entire time horizon.

The rest of the paper is organized as follows. For the sake of completeness, Section 2 provides a brief description of the time aggregated graph model that is used to represent spatio-temporal networks. This section also describes the shortest path algorithm for a given start time. Section 3 describes the proposed algorithm to compute the best start time at a given source node for any destination node. In Section 4, we present the experimental design and the performance analysis. In Section 5 we conclude and describe the direction of future work.

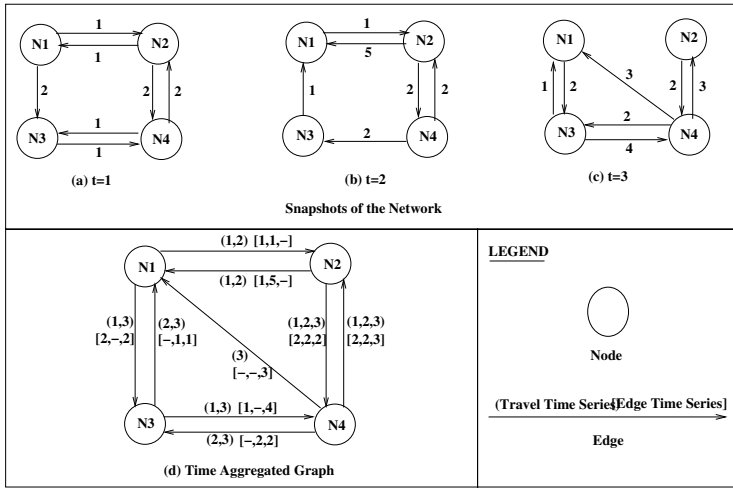
## 2 Basic Concepts

Spatial networks that show time-dependence serve as the underlying networks for many applications such as routing in transportation networks. Traditionally graphs have been extensively used to model spatial networks (e.g. road networks) [19]; weights assigned to nodes and edges are used to encode additional information. In a real world scenario, it is not uncommon for these network parameters to be time-dependent. It is important to be able to formulate computationally efficient and correct algorithms for the shortest path computation that take into account the dynamic nature of the networks. Models of these networks need to capture the possible changes in topology and values of network parameters with time and provide the basis for the formulation of computationally efficient and correct algorithms for the frequent computations like shortest paths.

Given a set of frequent queries posed by an application on a spatial network and the pattern of variations of the spatial network with time, we need to find a model that supports efficient and correct algorithms for computing the query results, while trying to minimize the storage and cost of computation. In this section we discuss the basics of the model used to represent time dependent spatial networks called “Time Aggregated Networks” [8]. The algorithms presented in this paper are formulated based on this model. Time aggregated graphs can not only capture the time-dependence of network parameters, but also account for the possibility of edges and nodes being absent during certain instants of time.

### 2.1 The Conceptual Model

A graph  $G = (N, E)$  consists of a finite set of nodes  $N$  and edges  $E$  between the nodes in  $N$ . If the pair of nodes that determines the edge is ordered, the graph is directed; if it is not, the graph is undirected. In most cases, additional information is attached to the nodes and edges. In this section, we discuss how



**Fig. 2.** Network at Various Time Instants and the Time Aggregated Graph

the time dependence of these edge/node parameters are handled in the proposed time-aggregated graph model.

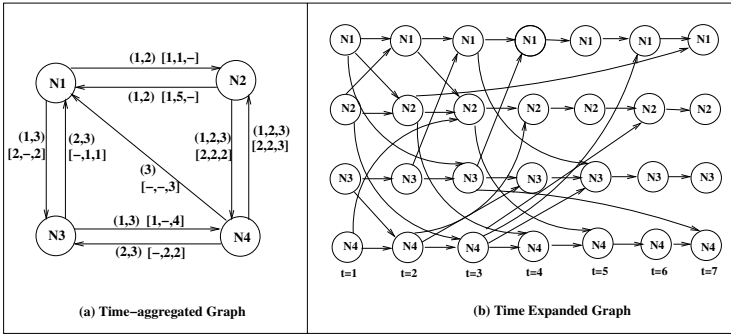
We define the time-aggregated graph as follows.

$$taG = (N, E, TF, f_1 \dots f_k, g_1 \dots g_l, w_1 \dots w_p | f_i : N \rightarrow \mathbb{R}^{TF}; g_i : E \rightarrow \mathbb{R}^{TF}; w_i : E \rightarrow \mathbb{R}^{TF})$$

where  $N$  is the set of nodes,  $E$  is the set of edges,  $TF$  is the length of the entire time interval,  $f_1 \dots f_k$  are the mappings from nodes to the time-series associated with the nodes,  $g_1 \dots g_l$  are mappings from edges to the time series associated with the edges, and  $w_1 \dots w_p$  indicate the time dependent weights (eg. travel times) on the edges.

Each edge has an attribute, called an edge time series that represents the time instants for which the edge is present. This enables the time aggregated graph to model the topological changes of the network with time. We assume that each edge travel time has a positive minimum and the presence of an edge at time instant  $t$  is valid for the closed interval  $[t, t + \sigma]$ .

Figure 2(a,b,c) shows a network at three time instants. The network topology and parameters change over time. For example, the edge N2-N1 is present at time instants  $t = 1, 2$ , and disappears at  $t = 3$ , and its weight changes from 1 at  $t = 1$  to 5 at  $t = 2$ . The time aggregated graph that represents this dynamic network is shown in Figure 2(d). In this figure, edge N2-N1 has two attributes, each being a series. The attribute (1, 2) represents the time instants at which the edge is present and  $[1, 1, -]$  is the weight time series, indicating the weights at various instants of time. Though this model can include spatial properties at nodes and edges, these properties are not incorporated in the algorithms presented in this paper. Figure 3(a) shows the time aggregated graph (corresponding to Figure 2(a),(b),(c)) and a time expanded graph that represent the same



**Fig. 3.** Time-aggregated Graph vs. Time Expanded Graph

scenario. Edge weights in a time expanded graph are not explicitly shown as edge attributes; instead they are represented by edges that connect the copies of the nodes at various time instants. For example, the weight 1 of edge N2-N1 at  $t = 1$  is represented by connecting the copy of node N2 at  $t = 1$  to the copy of node N1 at time  $t = 2$ . The time expansion for the example network needs to go through 7 steps since the latest edge traversal in the network ends at  $t = 7$ . The traversal of the edge N3-N4 that starts at  $t = 3$  ends at  $t = 7$ , the travel time of the edge being 4 units. The number of nodes is larger by a factor of  $T$ , where  $T$  is the number of time instants and the number of edges is also larger in number compared to the time-aggregated graph. If the value of  $T$  is very large in a spatial network, it would result in enormously large time expanded networks and consequently slow computations.

**Comparison of Storage Costs with Time Expanded Networks:** According to the analysis in [20], the memory requirement for a time expanded network is  $O(nT) + O(n + mT)$ , where  $n$  is the number of nodes,  $m$  is the number of edges in the original graph, and  $T$  is the length of the travel time series. The memory requirement for the time-aggregated graphs would be  $O(n + m)T$ , assuming an adjacency list representation of the graph. Each edge has a travel time series associated with it, instead of a scalar cost as in the case of a static graph.

This comparison shows that the memory usage of time-aggregated graphs is less than that of time expanded graphs by a factor of  $O(nT)$ .

**2.2 Shortest Path Computation for Time Aggregated Graphs (SP-TAG Algorithm)**

In time dependent networks, the shortest path and its traversal time are dependent on the start time at the source node. Here we give an outline of the algorithm that computes the shortest path for a given start time in a time-dependent network. The algorithm uses the time aggregated graph to represent the network. The application of a greedy strategy in the shortest path computation (which is a popular choice in most optimization problems) in a time-aggregated graph

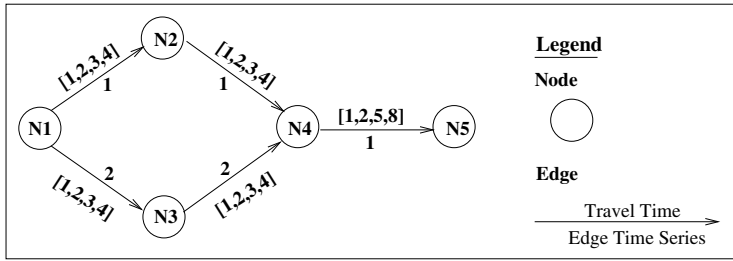


Fig. 4. Optimal Sub-structure of Shortest Paths

faces a challenge. Not all shortest paths display the optimal sub-structure, as illustrated by Figure 4. For the sake of simplicity, the travel times are constant in this example. It can be seen that a shortest path (N1-N3-N4-N5) from N1 to N5 for the start time  $t = 1$ , which takes 5 time units, does not display optimal substructure. The path from N1 to N4 following the above path is not optimal (shortest path being N1-N2-N4). Although such paths that do not display optimal sub-structure could exist, it can be proved that there is at least one optimal path which satisfies the optimal sub-structure property [8]. This result enables us to use a greedy approach to compute the shortest path. The algorithm, called the SP-TAG algorithm, uses greedy strategy to find the shortest path for a fixed start time. Every node has a cost associated with it which represents the travel time to reach the node from the source node. The algorithm picks the node with the least cost and updates the costs of its adjacent nodes. While finding the adjacent nodes, each edge is selected at its earliest available time instant ( $min_t$  operation in the algorithm description). A trace of the algorithm is given in Table 1. The table entries are the costs associated with each node (representing the arrival times at the node) at each iteration. The node marked as “closed” is the node with the minimum cost selected for expansion. The travel times are assumed to follow the FIFO property.

**Lemma 1:** The SP-TAG algorithm is correct.

**Proof:** As Figure 4 illustrates, the shortest path fails to have optimal structure due to a potential wait at the intermediate node ( $u$ ), after reaching this node traversing the optimal path from  $s$  to  $u$ . Consider the optimal path from  $s$  to  $u$ . Append this path to the path  $u - d$  (allowing a wait at the intermediate node  $u$ ) from the optimal path. This would be still the shortest path from  $s$  to  $d$ . Otherwise, it would contradict the optimality of the original shortest path.

**Lemma 2:** The time complexity of the SP-TAG algorithm is  $O(m(\log T + \log n))$  where  $T$  is the number of time instants,  $n$  is the number of nodes and  $m$  is the number of edges in the time aggregated graph.

---

**Algorithm 1. Shortest Path (SP-TAG) Algorithm**

---

**Input:**

- 1)  $G(N, E)$ : a graph  $G$  with a set of nodes  $N$  and a set of edges  $E$ ;  
 Each node  $n \in N$  has a property:  
     *Node Presence Time Series* : series of positive integers;  
 Each edge  $e \in E$  has two properties:  
     Edge Presence Time Series,  
     Travel\_time series : series of positive integers;  
      $\sigma_{u,v}(t)$  - travel time of edge  $uv$  at time  $t$ .
- 2)  $s$ : Source node,  $s \subseteq N$ ; 3)  $d$ : Destination node,  $d \subseteq N$ ;
- 4)  $t_{start}$ : Start Time;

**Output:** Shortest Route from  $s$  to  $d$  for  $t_{start}$

**Method:**

```

c[s] = t_start; ∀v ≠ s, c[v] = ∞;
// c[u] is the cost at the node u.
Insert s in priority queue Q.
while Q is not empty do {
    u = extract_min(Q);
    for each node v adjacent to u do {
        t = min_t((u, v), c[u]);
        if t + σu,v(t) < c[v] {
            c[v] = t + σu,v(t); parent[v] = u;
            if v is not in Q, insert v in Q;
        }
    }
    update Q;
}
}
}

```

Output the route from  $s$  to  $d$ .

---

**Proof:** The cost model analysis assumes an adjacency list representation of the graph with two significant modifications. The edge time series is stored in the sorted order. Attached to every adjacent node in the linked list are the edge time series and the travel time series.

For every node extracted from the priority queue  $Q$ , there is one edge time series look up and a priority queue update for each of its adjacent nodes. The time complexity of this step is  $O(\log T + \log n)$ . The asymptotic complexity of the algorithm would be

$$O(\sum_{v \in N} [degree(v) \cdot (\log T + \log n)]) = O(m(\log T + \log n)).$$

The time complexity of the SP-TAG shortest path algorithm based on a time expanded network is  $O(nT \log T + mT)$  [3]. It can be seen that the algorithm based on a time-aggregated graph is faster if  $\log n < T \log T$ .

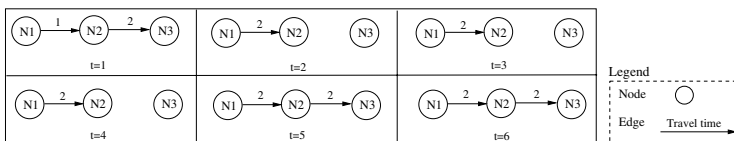
**Table 1.** Trace of the SP-TAG Algorithm for the Network shown in Figure 4

Iteration	N1	N2	N3	N4	N5
1	1 (closed)	$\infty$	$\infty$	$\infty$	$\infty$
2	1	2 (closed)	3	$\infty$	$\infty$
3	1	2	3 (closed)	3	$\infty$
4	1	2	3	3 (closed)	6
5	1	2	3	3	6 (closed)

### 3 Case Study: Best Start Time Shortest Paths

The time dependency of network parameters affects the connectivity and the shortest paths between nodes in a spatial network. As illustrated in Figure 5, the travel time from node N1 to node N3 changes with the start time. If the travel starts at  $t = 1$ , the commute time would be 6 units. A journey that starts at  $t = 1$  reaches N2 at  $t = 2$  and waits at N2 until edge N2-N3 becomes available at  $t = 5$ , thus taking a total travel time of 6 units to reach node N3. The travel on the same route would take 4 units if the start time is moved to  $t = 4$ . This shows that the shortest paths in a time-dependent network vary with time, which adds an interesting dimension to shortest path computation. A path that takes the smallest travel time for a source-destination traversal over the entire time horizon (called 'Best Start Time shortest Path') can be computed. This is significant since it suggests that it is possible to reduce the travel time for the same source-destination pair if the travel starts at the "right" time instant.

The formulation of algorithms to compute the paths that take the least commute time becomes non-trivial since most of the techniques that are used in static networks might not be applicable in dynamic scenarios. Since the network changes in its parameter values and the topology, meeting the requirements of efficiency and correctness can pose challenges. The potential waits at intermediate nodes can increase the total journey time even if an initial part of the path turns out to be optimal. Figure 5 shows a spatial network that changes with time. The figure shows the snapshots of the network at various instants of time, and the edges are marked with the travel times. It is significant to note that the prefix journeys of the best start time shortest path journey are not always optimal since some optimal prefix journeys can lead to longer waits at intermediate nodes. The best start time for a journey from node N1 to Node N3 is  $t = 4$ , which takes 4 time units. The optimal path from N1 to N3 that starts at  $t = 4$



**Fig. 5.** Network at various instants

is not optimal for the intermediate node N2. The best start time for a path from N1 to N2 is  $t = 1$ , which proves to be sub-optimal for a journey from N1 to N3. The lack of an optimal substructure in the best start time shortest paths rules out the possibility of using a greedy strategy in the algorithm design.

We propose an algorithm that computes the best start time based on a node-cost time series. The proposed algorithm uses the time aggregated network model to represent a time dependent spatial network.

### 3.1 BEst Start Time Shortest Path (BEST) Algorithm

While computing the best start time, each node needs to keep track of the travel times to the destination for every start time instant. The proposed algorithm attributes each node with a time series, with  $i^{th}$  entry representing the current, least travel time to the destination node for the start time  $t_i$ . Due to the lack of optimality of prefix paths and lack of ordering of nodes based on the costs (ie. travel times), nodes cannot be selected and “closed” based on a minimum scalar cost. The algorithm uses an iterative, label correcting approach [1] and each entry in a node time series is modified according to the following condition.

$$C_u[t] = \text{minimum}\{C_u[t], \sigma_{uv}(t) + C_v[t + \sigma_{uv}(t)]\} \text{ where, } uv \in E \quad (1)$$

$C_u[t]$  - Travel time from  $u \in N$  to the destination for the start time  $t$ .

$\sigma_{uv}(t)$  - Travel time of the edge  $uv$  at time  $t$ .

The algorithm maintains a list of all nodes that change its cost according to the condition and terminates when there is no further improvement indicated by an empty list. Though the list can be implemented using several data structures, studies on static networks [25,1] have shown that the Two-Q implementation [16] of label correcting algorithms performs the best on road networks.

The search starts at the destination node and proceeds to update the remaining nodes, finally finding the best start time shortest paths from all nodes to the destination. Figure 6 illustrates the trace of the algorithm on a small network. In this example, the destination node is the node N4. The node cost series  $C_4$  is initialized to  $[0, 0, 0, 0, 0]$  and the cost series  $C_i, i = 1, 2, 3$  are initialized to  $[\infty, \infty, \infty, \infty, \infty]$ . The nodes that have N4 in their adjacency lists (that is, all nodes  $N_i$  such that  $N_i N4 \in E$ ), N2 and N3 are relaxed according to condition (1). These nodes are added to the queue since there is a change in their cost series. The steps continue until the queue is empty, indicating that there is no further cost improvement at any of the nodes. At every iteration, the node that contributes to a cost improvement is stored in a descendant array to facilitate the trace of the shortest paths when the algorithm terminates. At the termination, the cost time series has the travel times for every start time  $t = 1, 2 \dots T$ . For example, the cost time series of node N1 shows that the travel times from N1 to N4 for start times  $t = 1$  is 4 time units, while the best start time at this node is  $t = 4$ , which results in a travel time of 2 time units and a best start time shortest path N1-N2-N4. N1-N2 takes 1 time unit at  $t = 4$ , reaches N2 at  $t = 5$  and continues on N2-N4 at  $t = 5$ , reaching N4 at  $t = 6$ , taking a total travel time of 2 time units. A more detailed trace is shown in Table 2.

**Algorithm 2.** BEST Algorithm

**Input:**

$G(N, E)$ : a graph  $G$  with a set of nodes  $N$  and a set of edges  $E$ ;  
 Each node  $n \in N$  has a property:  
     *Node Presence Time Series* : series of positive integers;  
 Each edge  $e \in E$  has two properties:  
     Edge Presence Time Series,  
     Travel\_time series : series of positive integers;  
 $\sigma_{u,v}(t)$  - travel time of edge  $uv$  at time  $t$ .

**Output:**

Best Start Time shortest route from  $s$  to  $d$ ;  
 Initialize;  
 While Queue not Empty  
      $v = \text{Dequeue}()$ ;  
     For every node  $u$  such that  $uv \in E$   
         For every entry in the cost series  $C_u$  of  $u$   
             if  $C_u(t) > \sigma_{uv}(t) + C_v(t + \sigma_{uv}(t))$   
                 Update  $C_u(t)$ ;  
                 Enqueue( $u$ );  
                 Update the descendant array of  $u$ .  
 Find the minimum entry in the node time series.  
 Return the BestStartTime and the ShortestRoute;

**Table 2.** Trace of the BEST Algorithm for the Network shown in Figure 6

Iteration	N1	N2	N3	N4	Queue
1	$\infty \dots \infty$	$\infty \dots \infty$	$\infty \dots \infty$	[0, 0, 0, 0, 0]	N1
2	$\infty \dots \infty$	[1, 1, 2, 2, 1]	[4, 4, 2, 4, 3]	[0, 0, 0, 0, 0]	N2, N3
3	$\infty \dots \infty$	[1, 1, 2, 2, 1]	[2, 3, 2, 4, 3]	[0, 0, 0, 0, 0]	N3
4	[4, 3, 3, 2, 3]	[1, 1, 2, 2, 1]	[2, 3, 2, 4, 3]	[0, 0, 0, 0, 0]	N1
5	[4, 3, 3, 2, 3]	[1, 1, 2, 2, 1]	[2, 3, 2, 4, 3]	[0, 0, 0, 0, 0]	-

**Lemma 3:** The algorithm terminates and computes the best start time paths from every node to the destination.

**Proof:** The algorithm terminates because there is a positive minimum for the travel time over every path, for every pair of nodes in the network since the edge weights (travel times) are positive and each such path has a finite number of edges. The updates on the costs according to condition(1) will generate the optimal travel times from a node to the destination at the termination of the algorithm. This can be proved by induction on the number of edges on the path. The base condition would be for paths with two edges, say from any node  $u$  to the destination node  $d$ . Every path with two edges from  $u$  to  $d$  will transit to some node  $v$  and then traverse the edge to  $d$  which takes the least time. If we assume the inductive hypotheses is true for every path with  $k$  edges, the



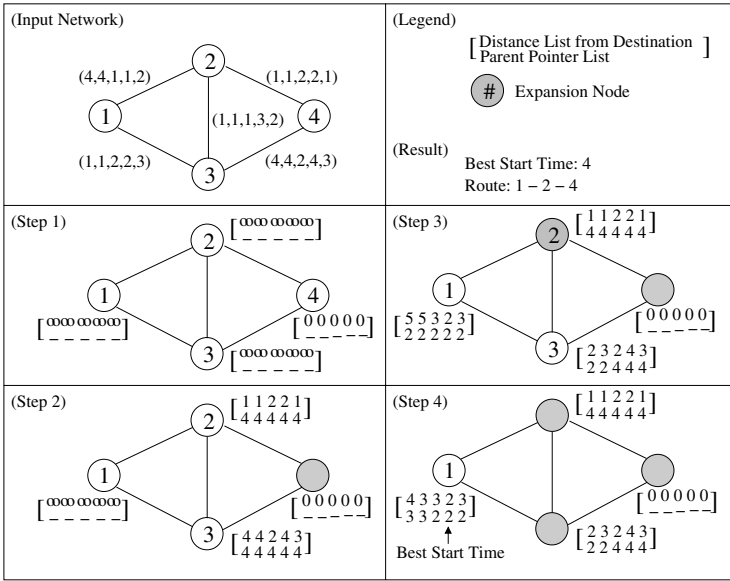


Fig. 6. Trace of the BEST Algorithm

minimality must hold for a path from  $u$  with  $(k + 1)$  edges since we can reach node  $u$  that with a minimal  $k$ -edge path and append  $uv$  with travel time  $\sigma_{uv}(t)$ .

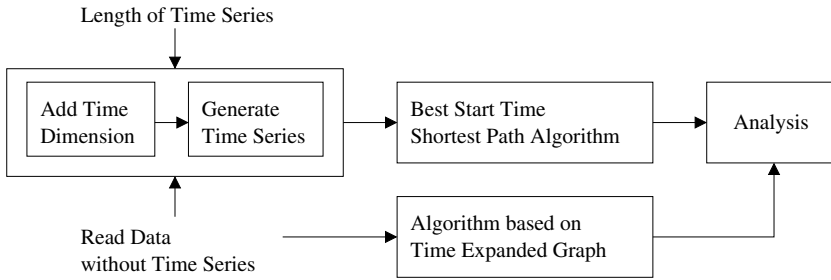
**Lemma 4:** The computational complexity of the BEST algorithm is  $O(n^2mT)$ , where  $n$  is the number of nodes,  $m$  is the number of edges and  $T$  is the length of the time series.

**Proof:** The worst case computational complexity of the label correcting algorithm based on Two-Q data structure is  $O(n^2m)$  when the node costs and edge weights are scalar quantities [1]. In the BEST algorithm, the relaxation step operates on a time series (node cost and edge weight) of length  $T$ . Hence the computational complexity of the algorithm is  $O(n^2mT)$ .

### 4 Experimental Analysis

In this section, the experimental analysis of the BEST algorithm and the SP-TAG algorithm are provided. The purpose of the performance evaluation of the algorithm is to compare the run-times with algorithms based on a time-expanded graph.

**Experiment Design.** Figure 7 illustrates the experiment design to compare the performance of the proposed algorithm and the algorithm based on a time expanded network. Time expanded graphs make copies of the original network



**Fig. 7.** Experiment Design

for every time instant under consideration. The model used for the proposed algorithm is time-aggregated graphs. In our experiments the following were selected as the independent parameters: 1) network size represented by number of nodes; and 2) the length of the time interval in terms of number of time instants. The data sets have two main components: (1) the network data that consists of the graph structure and (2) the travel time series. The networks chosen are road maps from the Minneapolis downtown area with radii of .5 mile, 1 mile, 2 miles and 3miles. This is appended with travel time series of various lengths. The travel time series were synthetically generated. This data was fed to both a time expanded graph generator, which generates the expanded graph encoding the travel time information. An algorithm for computing the shortest path for a given start time was run on this graph. The SP-TAG algorithm was run on the same dataset and the results were compared. The time expanded graph was then used to find the start time that results in the least travel time and the results were compared to the results from the BEST algorithm.

The experiments were conducted on a SUN Solaris workstation with 1.77GHz CPU, 1GB RAM and UNIX operating system. Each experimental result reported in the following sections is the average over 5 experiment runs with networks generated using the same input parameters, but with different destination nodes.

#### 4.1 Experimental Results and Anlysis

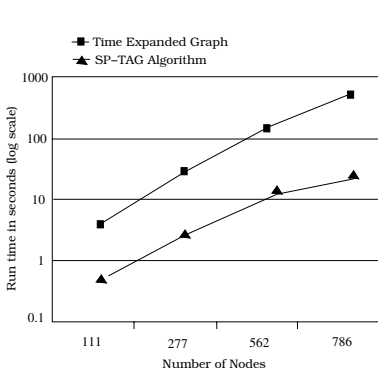
We wanted to answer three questions: (1) How does the network size (number of nodes, number of edges) affect the performance of the algorithms? (2) How does the length of the time series affect the performance of the algorithms? (3) How do the the two representations, time expanded graph and time aggregated graph, compare with respect to algorithm performance?

*Experiment 1: How does the network size affect the performance of the algorithms?*

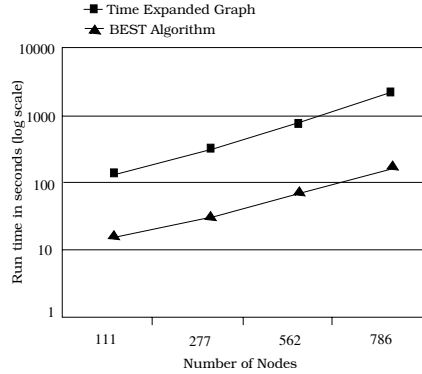
The purpose of the first experiment was to evaluate how the network size in terms of the number of nodes affects the performance of the algorithms. We fixed the length of the travel time series, and varied the network size to observe

**Table 3.** Description of Datasets

Dataset	Radius	No: of Nodes	No: of Edges
1	0.5 mile	111	287
2	1 mile	277	674
3	2 miles	562	1443
4	3 miles	786	2106



**Fig. 8.** SP-TAG Algorithm: Run-time With Respect to Network Size



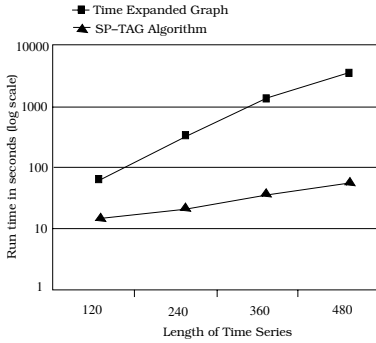
**Fig. 9.** BEST Algorithm: Run-time With Respect to Network Size

the run times of both the fixed start time(SP-TAG) and best start time(BEST) algorithms and time-expanded graph based algorithms.

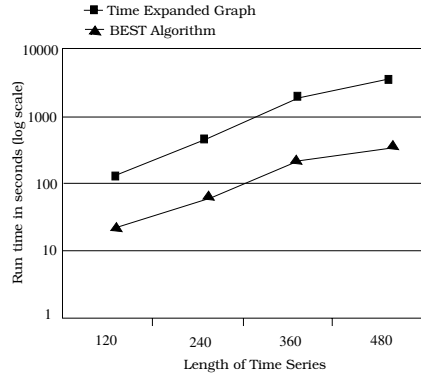
The experiment was done with four datasets that represent the road maps from the Minneapolis downtown area of .5 mile, 1 mile, 2 mile and 3mile radius. The length of the time series was fixed at 240. The number of nodes and edges in these datasets are provided in Table 3. Figure 8 shows the run-time of the fixed start time algorithm based on the time aggregated graph and the performance of the algorithm based on the time expanded graph. The SP-TAG algorithm runs faster than the time-expanded graph based algorithm in all cases; further, its run-time seems to increase at a slower rate. Figure 9 shows the performance of the BEST algorithm and that of the time expanded graph algorithm. The run time of the BEST algorithm is much lower than that of the time expanded graph algorithm.

*Experiment 2: How does the length of the time series affect the performance of the algorithms?*

In the second experiment, we evaluated how the number of time instants affects the performance of the algorithms. We fixed the network size, and varied the length of the time series to observe the run-time. The number of time instants was varied from 120 to 480 and the network size parameters were fixed at 562 nodes and 1443 edges. As seen in Figure 10, the SP-TAG algorithm performs



**Fig. 10.** SP-TAG Algorithm: Run-time With Respect to Length of Time series



**Fig. 11.** BEST Algorithm: Run-time With Respect to Length of Time series

better. Figure 10 shows the performance of the BEST algorithm and that of the time expanded graph algorithm. As the length of the time series increases, the number of copies of the entire network required in the case of the time expanded graph increases, resulting in a considerable increase in the size of the entire network, leading to almost exponential increases in run time.

*3: How do the the two representations, time expanded graph and time aggregated graph, compare with respect to algorithm performance?*

Based on the results of Experiments (1) and (2), it can be seen that algorithms based on the time aggregated graph perform better than those based on the time expanded graph.

## 5 Conclusions and Future Work

Spatio-temporal networks form a key part of critical applications such as emergency planning and there is a great need for database support in this area. The paper describes a model to represent a spatio-temporal network and proposes two algorithms for shortest path computations. The formulation of these algorithms is based on a model for spatio-temporal networks called time-aggregated graphs. In addition to the algorithm that computes the shortest path for a given start time, we also addressed the time-dependence of shortest paths in networks by formulating an algorithm that computes shortest paths which result in the least travel time over the entire time period. We also present an experimental analysis of the best start time (BEST) algorithm and the fixed start time algorithm (SP-TAG) (which was proposed in [8]). Experiments show that the algorithms based on time aggregated graphs significantly reduce the computational cost compared to similar algorithms based on time expanded networks.

We plan to evaluate the performance of the algorithms using real-traffic datasets shortly. We recently acquired a dataset for interstate highway I-66.

This data contains time-stamped occupancy, speed and volume collected from a number of stations on I-66 on November 6, 2006 using the Advanced Interactive Traffic Visualization System [23]. We anticipate that this evaluation will give new insights into the average case run time of the algorithms, which we expect to be significantly better than the worst case complexity, especially in the case of the BEST algorithm based on a label correcting approach. We are also planning to extend our experiments with Google traffic data and traffic archive data collected by the Traffic Management Center at the University of Minnesota.

The time aggregated graphs can accommodate the time-varying capacities of the road networks. The proposed algorithms need to be extended to give optimal solutions subject to the constraints of time-varying capacities. This would extend the use of the algorithms to domains such as evacuation planning in emergency management, where capacity constraints in the network pose significant challenges. We plan to include spatial attributes at nodes and edges and incorporate necessary changes in the algorithms. We plan to incorporate the algorithms as building blocks that find the shortest paths in the CCRP evacuation planner [13]. We will also explore other graph problems in the context of time aggregated graphs.

## Acknowledgments

We are particularly grateful to the members of the Spatial Database Research Group at the University of Minnesota for their helpful comments and valuable suggestions. We would also like to express our thanks to Kim Koffolt for improving the readability of this paper.

This work was supported by the NSF SEI grant and Minnesota Department of Transportation. The content does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred.

## References

1. Cherkassky, B.V., Goldberg, A.V., Radzik, T.: Shortest Paths Algorithms: Theory and Experimental Evaluation . *Mathematical Programming* 73, 129–174 (1996)
2. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: *Introduction to Algorithms* (Chapter 26, Flow Networks). MIT Press, Cambridge, MA, USA (2002)
3. Dean, B.C.: Algorithms for minimum-cost paths in time-dependent networks. *networks* 44(1), 41–46 (2004)
4. Ding, Z., Guting, R.H.: Modeling temporally variable transportation networks. *Proc. 16th Intl. Conf. on Database Systems for Advanced Applications* , 154–168 (2004)
5. Erwig, M.: *Graphs in Spatial Databases*. PhD thesis, Fern Universität Hagen (1994)
6. Erwig, M., Guting, R.H.: Explicit graphs in a functional model for spatial databases. *IEEE Transactions on Knowledge and Data Engineering* 6(5), 787–804 (1994)
7. ESRI. *ArcGIS Network Analyst* (2006), <http://www.esri.com/software/arcgis/extensions/>

8. George, B., Shekhar, S.: Time-aggregated Graphs for Modeling Spatio-Temporal Networks - An Extended Abstract. Proceedings of Workshops at International Conference on Conceptual Modeling (2006)
9. Hamre, T.: Development of Semantic Spatio-temporal Data Models for Integration of Remote Sensing and in situ Data in Marine Information System. PhD thesis, University of Bergen, Norway (1995)
10. Kaufman, D.E., Smith, R.L.: Fastest paths in time-dependent networks for intelligent vehicle highway systems applications. *IVHS Journal* 1(1), 1–11 (1993)
11. Kohler, E., Langtau, K., Skutella, M.: Time-expanded graphs for flow-dependent transit times. In: Proc. 10th Annual European Symposium on Algorithms, pp. 599–611 (2002)
12. Sellis, T., Koubarakis, M., Frank, A., Grumbach, S., Güting, R.H., Jensen, C., Lorentzos, N.A., Manolopoulos, Y., Nardelli, E., Pernici, B., Theodoulidis, B., Tryfona, N., Schek, H.-J., Scholl, M.O.: Spatio-Temporal Databases: The CHOROS Approach. In: Sellis, T., Koubarakis, M., Frank, A., Grumbach, S., Güting, R.H., Jensen, C., Lorentzos, N.A., Manolopoulos, Y., Nardelli, E., Pernici, B., Theodoulidis, B., Tryfona, N., Schek, H.-J., Scholl, M.O. (eds.) *Spatio-Temporal Databases*. LNCS, vol. 2520, Springer, Heidelberg (2003)
13. Lu, Q., George, B., Shekhar, S.: Capacity Constrained Routing Algorithms for Evacuation Planning: A Summary of Results. In: Bauzer Medeiros, C., Egenhofer, M.J., Bertino, E. (eds.) *SSTD 2005*. LNCS, vol. 3633, Springer, Heidelberg (2005)
14. Oracle. Oracle Spatial 10g, An Oracle White Paper. August (2005), <http://www.oracle.com/technology/products/spatial/>
15. Orda, A., Rom, R.: Minimum weight paths in time-dependent networks. *networks* 21, 295–319 (1991)
16. Pallottino, S.: Shortest-Path Methods: Complexity, Interrelations and New Propositions. *Networks* 14, 257–267 (1984)
17. Pallottino, S., Scutella, M.G.: Shortest path algorithms in transportation models: Classical and innovative aspects. *Equilibrium and Advanced transportation Modelling*, 245–281 (1998)
18. Rasinmäki, J.: Modelling spatio-temporal environmental data. In: 5th AGILE Conference on Geographic Information Science, Palma, Balearic Islands, Spain (April 2002)
19. Shekhar, S., Chawla, S.: *Spatial Databases: Tour*. Prentice-Hall, Englewood Cliffs (2003)
20. Sawitzki, D.: Implicit Maximization of Flows over Time. Technical report, University of Dortmund (2004)
21. Dreyfus, S.E.: An appraisal of some shortest path algorithms. *Operations Research* 17, 395–412 (1969)
22. Shekhar, S., Liu, D.: CCAM: A Connectivity-Clustered Access Method for Networks and Networks Computations. *IEEE Transactions on Knowledge and Data Engineering*, 9 (January 1997)
23. Spatial Data Management Lab, Virginia Polytechnic Institute and State University. AITVS: Advanced Interactive Traffic Visualization System (2007), [http://spatial.nvc.cs.vt.edu/traffic\\_zhh/](http://spatial.nvc.cs.vt.edu/traffic_zhh/)
24. Stephens, S., Rung, J., Lopez, X.: Graph data representation in oracle database 10g: Case studies in life sciences. *IEEE Data Engineering Bulletin* 27(4), 61–66 (2004)
25. Zhan, F.B., Noon, C.E.: Shortest Paths Algorithms: An Evaluation Using Real Road Networks. *Transportation Science* 32, 65–73 (1998)

# Author Index

- Béguet, Jean 312  
Böhm, Christian 294  
Brochhaus, Christoph 57  
Castano, Silvana 185  
Chen, Arbee L.P. 20  
Chow, Chi-Yin 258  
Dolev, Nir 276  
Doytsher, Yerach 276  
Ester, Martin 112  
Frank, Richard 112  
George, Betsy 460  
Ghinita, Gabriel 221  
Gonzalez, Hector 441  
Gross-Amblard, David 312  
Hadjieleftheriou, Marios 348  
Han, Jiawei 441  
Harrington, Brian 130  
Hess, Guillermo Nudelman 185  
Huang, Xuegang 93  
Huang, Yan 130  
Huang, Zhiyong 366, 403  
Iochpe, Cirano 185  
Jensen, Christian S. 93, 366, 403  
Jin, Wen 112  
Jung, Sungwon 423  
Kalnis, Panos 221  
Kantere, Verena 385  
Kanza, Yaron 276  
Khoshgozaran, Ali 239  
Kim, Sangho 460  
Kriegel, Hans-Peter 75  
Kröger, Peer 75  
Kunath, Peter 75, 294  
Lafaye, Julien 312  
Lee, Byung Suk 330  
Lee, Chia-Chen 20  
Lee, Jae-Gil 441  
Li, Xiaolei 441  
Lu, Hua 93, 366, 403  
Mamoulis, Nikos 1, 348  
McKenney, Mark 167, 203  
Mokbel, Mohamed F. 258  
Mouratidis, Kyriakos 38  
Ooi, Beng Chin 366  
Oswald, Marcus 423  
Papadopoulos, Stavros 38  
Patroumpas, Kostas 148  
Pauly, Alejandro 203  
Pfeifle, Martin 423  
Potamias, Michalis 148  
Praing, Reasey 203  
Pryakhin, Alexey 294  
Reinelt, Gerhard 423  
Renz, Matthias 75  
Ruas, Anne 312  
Sacharidis, Dimitris 38  
Safrá, Eliyahu 276  
Sagiv, Yehoshua 276  
Šaltenis, Simonas 93  
Schneider, Markus 167, 203  
Schubert, Matthias 294  
Seidl, Thomas 57  
Sellis, Timos 148, 385  
Shahabi, Cyrus 239  
Shekhar, Shashi 460  
Skiadopoulos, Spiros 221  
Suh, Jonghyun 423  
Tao, Yufei 348  
Tran, Tri Minh 330  
U, Leong Hou 1  
Vo, Khoa T. 423  
Wu, Yi-Hung 20  
Xu, Linhao 403  
Yiu, Man Lung 1