Proceedings

of the Third Workshop on

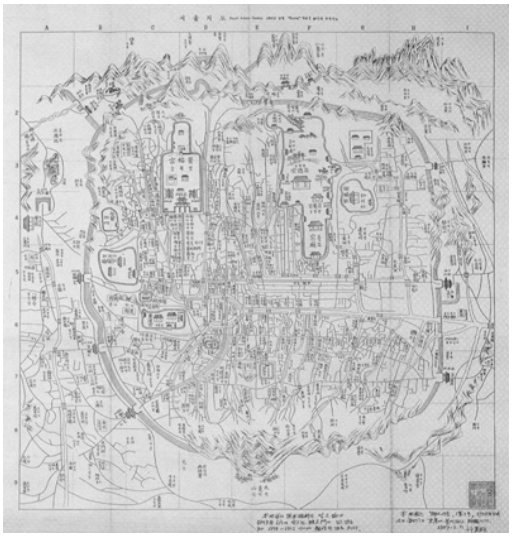
Spatio-Temporal Database Management

STDBM 06

September 11, 2006

Seoul, South Korea

in conjuction with VLDB 06



Edited by

Christophe Claramunt

Ki-Joune Li

Simonas Šaltenis

Workshop Co-Chairs

Christophe Claramunt, Naval Academy, France
Simonas Šaltenis, Aalborg University, Denmark
Ki-Joune Li, Pusan National University, Korea

Advisory Committee

Ralf Hartmut Güting, Fernuniversität Hagen, Germany
Christian S. Jensen, Aalborg University, Denmark
Mario A. Nascimento, University of Alberta, Canada
Yannis Manolopoulos, Aristotle University, Greece

Program Committee

Dave Abel, CSIRO, Australia
Walid G. Aref, Purdue University, USA
Lars Arge, University of Aarhus, Denmark
Elisa Bertino, Purdue University, USA
Michela Bertolotto, University College Dublin, Ireland
Thomas Brinkhoff, Oldenburg University of Applied Sciences, Germany
Edward P.F. Chan, University of Waterloo, Canada
Andrew U. Frank, Technical University of Vienna, Austria
Fabio Grandi, University of Bologna, Italy
Ralf Hartmut Güting, Fernuniversität Hagen, Germany
Erik Hoel, ESRI, USA
Kathleen Hornsby, University of Maine, USA
Christopher B. Jones, Cardiff University, UK
George Kollios, Boston University, USA
Ravikanth V. Kothuri, Oracle Corporation, USA
Mario A. Lopez, University of Denver, USA
Yannis Manolopoulos, Aristotle University, Greece
Claudia Bauzer Medeiros, UNICAMP, Brazil
Dimitris Papadias, UST Hong Kong, China
Jignesh M. Patel , University of Michigan, USA
Dieter Pfoser, CTI, Greece
Philippe Rigaux, Université Paris-Dauphine, France
John Roddick, Flinders University, Australia
Jörg Sander, University of Alberta, Canada

Michel Scholl, Cedric/CNAM, France
Bernhard Seeger, University of Marburg, Germany
Timos Sellis, NTUA, Greece
Richard Snodgrass, University of Arizona, USA
Jianwen Su, UC Santa Barbara, USA
Yannis Theodoridis, University of Piraeus, Greece
Theodoros Tzouramanis, University of the Aegean, Greece
Babis Theodoulidis, University of Manchester, UK
Goce Trajcevski, Northwestern University, USA
Nectaria Tryfona, Talent Information System SA, Greece
Vassilis Tsotras, UC Riverside, USA
Jeffrey S. Vitter, Purdue University, USA
Agnès Voisard, Fraunhofer ISST & FU Berlin, Germany
Peter Widmayer, ETH Zurich, Switzerland
Jef Wijsen, University of Mons-Hainaut, Belgium
Stephan Winter, University of Melbourne, Australia
Michael Worboys, University of Maine, USA
Donghui Zhang, Northeastern University, USA

Sponsor

Geospatial Information Technology Co. Ltd, South Korea

Preface

Managing spatially and temporally referenced data is becoming increasingly important, given the continuing advances in wireless communications, sensor technologies, and ubiquitous computing. New types of spatial databases and applications enabled by these technologies require data management technology to store, query, and analyze spatio-temporal data.

This volume contains papers presented at the Third Workshop on Spatio-Temporal Database Management, STDBM'06, co-located with the 32nd International Conference on Very Large Data Bases, VLDB 2006. This workshop follows the previous STDBM workshops respectively held in conjunction with VLDB'99 in Edinburgh and VLDB'04 in Toronto. The goal of this workshop is to bring together leading researchers and developers in the area of spatio-temporal databases in order to discuss and exchange novel research ideas and experiences with real world spatio-temporal databases.

The workshop received 14 submissions. All papers were evaluated by at least three of the 42 members of the program committee. At the end nine papers of high quality were accepted and are included in these proceedings.

We would like to thank the members of the workshop's steering committee for their valuable advice organizing the workshop, and program committee members for making their reviews in due time. We are thankful to our sponsor, Geospatial Information Technology Co. Ltd., for their support. We finally acknowledge the support provided by the VLDB organization, and of the CEUR organization for supporting the on-line publication of the proceedings.

Christophe Claramunt, Ki-Joune Li, and Simonas Šaltenis
STDBM'06 Organizers and PC co-chairs

Contents

Session 1: Spatio-Temporal Databases for Transportation Systems

Automatically and Efficiently Matching Road Networks with Spatial Attributes in Unknown Geometry Systems.
Ching-Chien Chen, Cyrus Shahabi Craig A. Knoblock,Mohammad Kolahdouzan 1

Update-efficient Indexing of Moving Objects in Road Networks.
Jidong Chen, Xiaofeng Meng,Yanyan Guo,Zhen Xiao 9

A Spatio-temporal Database Model on Transportation Surveillance Videos.
Xin Chen, Chengcui Zhang 17

Session 2: Spatio-Temporal Queries

Predicted Range Aggregate Processing in Spatio-temporal Database.
Wei Liao, Guifen Tang, Ning Jing, Zhinong Zhong 25

Additively Weighted Voronoi Diagrams for Optimal Sequenced Route Queries.
Mehdi Sharifzadeh, Cyrus Shahab 33

On Construction of Holistic Synopses under the Duplicate Semantics of Streaming Queries.
David Toman 41

Session 3: Spatio-Temporal Data Structures and Data

Mining Long, Sharable Patterns in Trajectories of Moving Objects.
Gyozo Gidófalvi, *Torben Bach Pedersen* 49

Fusion Based Methodology for Spatial Clustering.
Pavani Kuntala, *Vijay V. Raghavan* 59

A Storage Scheme for Multi-dimensional Databases Using Extendible Array Files.
Ekow J. Otoo, *Doron Rotem* 67

Automatically and Efficiently Matching Road Networks with Spatial Attributes in Unknown Geometry Systems

Ching-Chien Chen

Geosemble Technologies
2041 Rosecrans Ave, Suite 245
El Segundo, CA 90245
jchen@geosemble.com

Cyrus Shahabi

Craig A. Knoblock

University of Southern California
Department of Computer Science
Los Angeles, CA 90089
shahabi, knoblock@usc.edu

Mohammad Kolahdouzan*

Yahoo! Search Marketing
Burbank, CA
mohammad_ysm@yahoo.com

Abstract

Vast amount of geospatial datasets are now available through numerous public and private organizations. These datasets usually cover different areas, have different accuracy and level of details, and are usually provided in the vector data format, where the latitude and longitude of each object is clearly specified. However, there are scenarios in which the spatial attributes of the objects are intentionally transformed to a different, and usually unknown, (alien) system. Moreover, it is possible that the datasets were generated from a legacy system or are represented in a native coordinate system. An example of this scenario is when a very accurate vector data representing the road network of a portion of a country is obtained with unknown coordinate. In this paper, we propose a solution that can efficiently and accurately find the area that is covered by this vector data simply by matching it with the (possibly inaccurate and abstract) data with known geocoordinates. In particular, we focus on vector datasets that represent road networks and our approach identifies the exact location of the vector dataset of alien system by comparing the distribution of the detected road intersection points between two datasets. Our experiment results show that our technique can match road vector datasets that are composed of thousands of arcs in a relatively short time with 91% precision and 92.5% recall for the matched road feature points.

1. Introduction

With the rapid improvement of geospatial data collection techniques, the growth of Internet and the implementation of Open GIS, a large amount of geospatial data are now readily available on the web. The examples of well-known vector datasets are US Census TIGER/Line files[1] (covering most roads over the United States), NAVSTREETS from NAVTEQ,[2] VPF data from NGA (U.S. National Geospatial-Intelligence Agency), [3] and DLG data from USGS (U.S. Geological Survey).[4] The Yahoo Map Service,[5] Google Map Service,[6] Microsoft TerraService[7] [1] are good examples of map or satellite imagery repositories. These datasets usually cover different areas, have different accuracy and level of details, and some of them are provided in the vector data format, where the latitude and longitude of each vector object is clearly specified. However, there are scenarios in which the spatial attributes of the vector objects are intentionally transformed to a different, and usually unknown, (alien) system. Moreover, it is also possible that the datasets were generated from a legacy system or are represented in a native coordinate system.

Figure 1 illustrates a scenario where we want to locate the area of a USGS raster topographic map (as shown in

** This work was done when the author was working at University of Southern California as a Post-Doc Research Associate.*

Proceedings of the 3rd Workshop on STDBM
Seoul, Korea, September 11, 2006

[1] http://www.census.gov/geo/www/tiger/

[2] http://www.navteq.com/

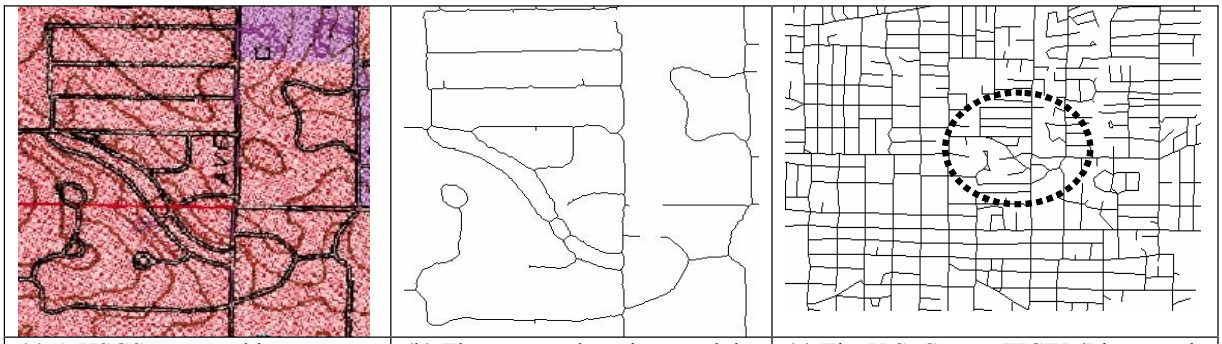
[3] http://www.nga.mil/

[4] http://tahoe.usgs.gov/DLG.html

[5] http://maps.yahoo.com/

[6] http://maps.google.com

[7] http://terraserver-usa.com/



(a) A USGS topographic map	(b) The extracted road network in a native coordinate system	(c) The U.S. Census TIGER/Lines road network

Figure 1: Two road networks cover overlapping areas

Figure 1(a)). The map covers partial area of St. Louis County, MO in U.S.A., but its exact geocoordinates are inaccessible. The map is first processed to extract road network (as shown in Figure 1(b)) by using a text/graphics separation technique developed by us [2] . Obviously, the extracted road network is in a native coordinate system (i.e., raster pixel x/y in Cartesian system). Meanwhile, a larger road network covering this county is publicly available from US Census TIGER/Lines. By identifying matched features between the two road networks, the system can then automatically infer the geocoordinate of the extracted map road network (as the area highlighted in Figure 1(c)).

There have been a number of efforts to automatically or semi-automatically detect matched features across different road vector datasets [3, 4, 5, 6, 7]. Given a feature point from one dataset, these approaches utilize different matching strategies to discover the corresponding point within a predetermined distance (i.e., a localized area). This implies that these existing algorithms only handle the matching of vector datasets in the same geometry system (i.e., the same coordinate system[8]). Hence, to the best of our knowledge, no general method exits to resolve the matching of two vector data in unknown geometry systems. Furthermore, processing large vector datasets often requires significant CPU time. Our methodology, described in this paper, is able to automatically and efficiently handle the matching of diverse and potentially large vector datasets, independent of the coordinate system used. In particular, we focus on vector datasets that represent road networks.

The basic idea of our approach is to find the transformation T between the layout (with relative distances) of the feature point set on one road network and the corresponding feature point set on the other road network. This transformation achieves global matching between two feature point sets by locating the common point pattern among them. More precisely, the system can detect feature points from both road networks. The distribution of detected feature points from each road network forms a particular (and probably unique) point pattern for the road network. In order to improve the running time, our approach exploits auxiliary spatial information to reduce the search space for the transformation T. Once the matched points across different road networks are identified, the system can then utilize this transformation to map the road network of alien geometry (coordinate) system into a known coordinate system (e.g., geodetic coordinate system).

To illustrate the usefulness of our approach, consider the matching of two road networks: one is in an unknown coordinate system but with more accurate geometry and the other has rich attributes and a known coordinate system but with poor geometry. Applying our matching algorithm to these two road networks can result in a superior road network that combines the accuracy of the road geometry from one vector dataset and rich attributes from the other. Furthermore, these matched points can be used as control points to conflate these two road network datasets [6].

The remainder of this paper is organized as follows. Section 2 describes our approach in details. Section 3 provides experimental results. Section 4 discusses the related work and Section 5 concludes the paper by discussing our future plans.

2. Proposed Approach

In this section, we first describe our overall approach to match two road networks. Then, we describe the details of our techniques.

2.1 Approach Overview

Intuitively, matching road networks relies on the process of matching the road segments from two vector datasets to find the corresponding road segments. However, this is a challenging task for two large road networks, especially when one of the road networks is in a different or unknown geometry system. To address this issue, we propose to match two datasets based on some feature points detected from the road networks. In particular, we

[8] In this paper, we use the terms "*geometry system*" and "*coordinate system*" interchangeably.

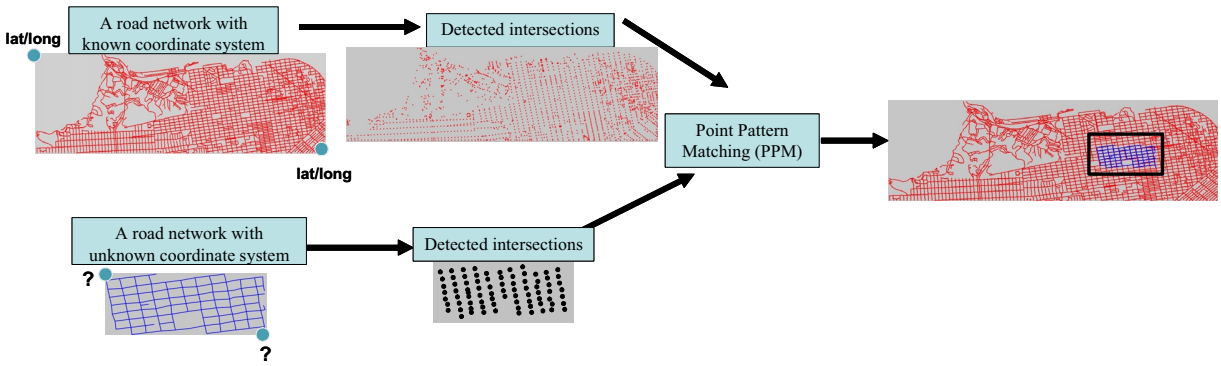


Figure 2: The overall approach

utilize road intersections as the feature points. Road intersections are good candidates for being matched, because road intersections are salient points to capture the major features of the road network and the road shapes around intersections are often well-defined. In addition, various GIS and computer vision researchers have shown that the intersection points on the road networks are good candidates to be identified as an accurate set of matched points [7, 8, 9, 10].

After detecting a set of intersection points from each road network separately, the remaining problem is how to match these intersection points effectively and efficiently to locate a common distribution (or pattern) from these intersections. Our system perceives the distribution of detected intersections from each road network as the *fingerprint* of the road network. Then our system finds the transformation T between the layout (with relative distances) of the feature point set on one road network data and the feature point set on the other road network. This transformation achieves global alignment between two intersection point sets by locating the common point pattern among them.

Figure 2 shows our overall approach. Using detected road intersections as input, the system locates the common point pattern across these two point sets by computing a proper transformation between them. The system can then utilize this transformation to map the road network in unknown geometry system into a known coordinate system. We describe our detailed techniques in the following sections.

2.2 Finding the feature points from vector datasets

The technique to detect road intersections from road network relies on the underlying road vector data representation. Typically, there are two common ways to represent the geometry of a road vector dataset: (1). The road network is composed of multiple road segments (polylines), and the line segments are split at intersections (as the example shown in Figure 3(a)). (2). The road network is composed of multiple road segments (polylines), but the line segments are *not* split (if not necessary) at intersections (see Figure 3(b)).

This generation of our system focus on handling vector data represented in the first way (i.e., the line segments are split at intersections), because most of the popular road vector datasets (such as US Census TIGER/Line files and NAVSTREETS from NAVTEQ) represent their datasets in such way. Based on this sort of road segment representation, the process of finding the intersection points from the road network is divided into two steps. First, the system examines all line segments in the vector data to label the endpoints of each segment as the candidate intersection points. Second, the system examines the connectivity of these candidate points to determine if they are intersection points. In this step, each candidate point is verified to see if there are more than two line segments connected at this point. If so, this point is marked as an intersection point and the directions of the segments that are connected at the intersection point are calculated. In practice, the search of road intersections from large road networks is supported efficiently by spatial access method R-tree [11].

2.3 Finding the matched feature points by Point Pattern Matching (PPM)

Now that we have described how to detect feature points from a road network, we now describe how this

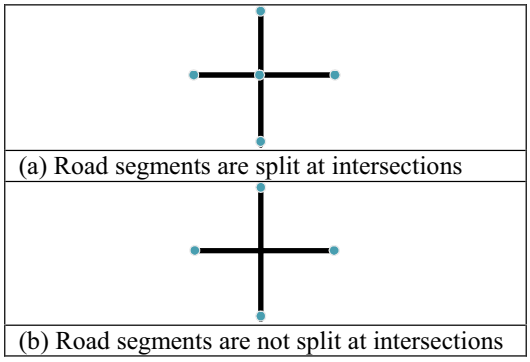


Figure 3: Different ways to represent a cross-shaped road network with one intersection

information can be used to automatically match two road networks. Let U= {u_i | u_i= (x_i, y_i), where (x_i, y_i) is the location of intersections of the first road network} and V= {v_j | v_j= (m_j, n_j), where (m_j, n_j) is the location of intersections of the second road network}. Our objective is to locate the set: {Rel_{Pat}={(u_i,v_j) | where u_i is the intersection on the first vector dataset and v_j is the corresponding intersection (if any) on the second dataset. That is, point u_i and v_j are formed by the same intersected road segments}. Consider identifying matched point pattern between two road networks. If the system can recognize the names of road segments that meet at intersections, it can use these road names to infer the set Rel_{Pat}. However, road vector data may not include the non-spatial attribute, road name. Instead, we propose our approach that relies on some prominent geometric information, such as the distribution of points, the degree of each point and the direction of incident road segments, to locate the matched point pattern. In other words, the problem of point pattern matching is at its core a geometric point sets matching problem. The basic idea is to find the transformation T between the layout (with relative distances) of the point set U and V.

The key computation of matching the two sets of points is calculating a proper transformation T, which is a 2D rigid motion (rotation and translation) with scaling. Because the majority of vector datasets are oriented such that north is up, we only compute the translation transformation with scaling. Without loss of generality, we consider how to compute the transformation where we map from a fraction α of the points of U to the points of V. The reason that only a fraction α of the points of U is considered is that one road vector dataset could be detailed while the other one is represented abstractly or there may be some missing/noisy points from each road network. The transformation T brings at least a fraction α of the points of U into a subset of V. This implies:

$\exists$ T and $U' \subseteq U$, such that $T(U') \subseteq V$, where $| U' | \geq \alpha | U |$ and $T(U')$ denotes the set of points that results from applying T to the points of U'. Or equivalently, for a 2D point (x, y) in the point set $U' \subseteq U$, $\exists$ T in the matrix form $\begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ Tx & Ty & 1 \end{bmatrix}$ (S_x and S_y are scale factors along x and y direction, respectively, while T_x and T_y are translation factors along x and y directions, respectively), such that

$$[x,\ y,\ 1] * \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ Tx & Ty & 1 \end{bmatrix} = [m\ ,\ n,\ 1]\ ,\ \text{where}\ \ | U' | \geq \alpha | U |$$

and the 2D point (m, n) belongs to the intersection point set V on the second vector dataset. With this setting, we do not expect point coordinates to match exactly because of finite-precision computation or small errors in the datasets. Therefore, when checking whether a 2D point p belongs to the point set V, we declare that $p \in V$, if there exists a point in V that is within Euclidean distance δ of p for a small fixed positive constant δ, which controls the degree of inaccuracy. The minimum δ such that there is a match for U' in V is called *Hausdorff distance*. Different computations of the minimum *Hausdorff distance* have been studied in great depth in the computational geometry literature [12]. We do not seek to minimize δ but rather adopt an acceptable threshold for δ. The threshold is relatively small compared to the average inter-point distances in V. In fact, this sort of problem was categorized as "Nearly Exact" point matching problem in [13].

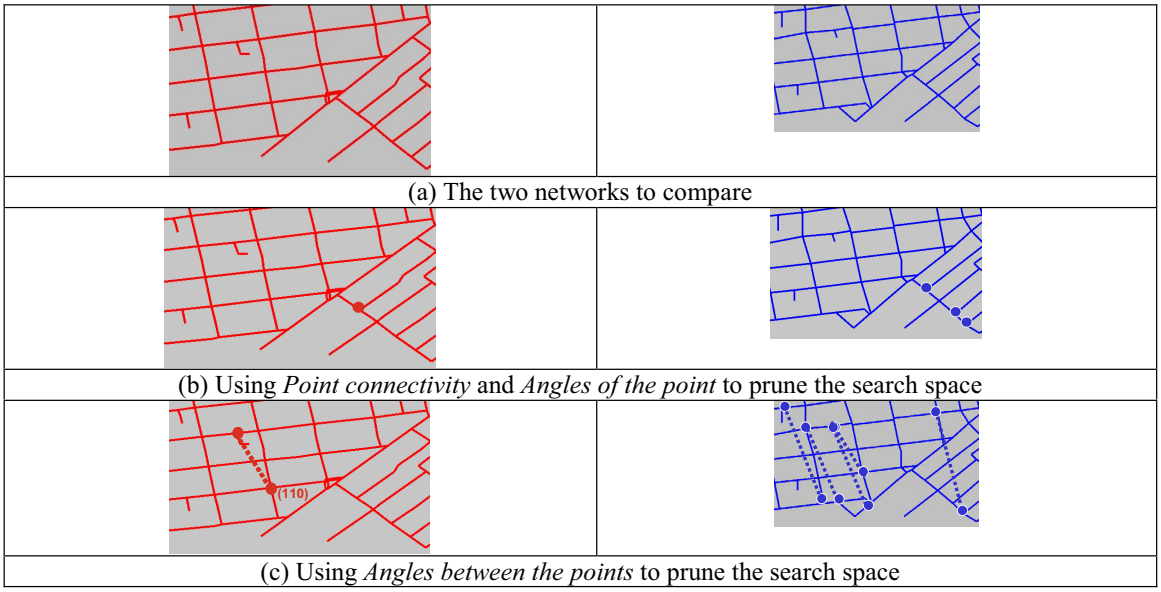
Given the parameters α and δ, to obtain a proper transformation T, we need to compute the values of the four unknown parameters S_x, S_y, T_x and T_y. This implies that at least four different equations are required. A straightforward (brute-force) method is first choosing a point pair (x_1, y_1) and (x_2, y_2) from U, then, for every pair of distinct points (m_1, n_1) and (m_2, n_2) in V, the transformation T' that map the point pair on U to the point pair on V is computed by solving the following four equations:

$S_x * x_1 + T_x = m_1$ $\qquad$ $S_y * y_1 + T_y = n_1$

$S_x * x_2 + T_x = m_2$ $\qquad$ $S_y * y_2 + T_y = n_2$

Each generated transformation T' is thus applied to the entire points in U to check whether there are more than $\alpha|U|$ points that can be aligned with some points on V within the threshold δ. This process is repeated for each possible point pair from U, which implies that it could require examining $O(|U|^2)$ pairs in the worst case. Since for each such pair, the algorithm needs to try all possible point pairs on V (i.e., $O(|V|^2)$) and spends $O(|U|\ log|V|)$ time to examine the generated transformation T', this method has a worst case running time of $O(|U|^3\ |V|^2\ log|V|)$. The advantage of this approach is that we can find a mapping (if the mapping exists) with a proper threshold δ, even in the presence of very noisy data. However, it suffers from high computation time. One way to improve the efficiency of the algorithm is to utilize randomization in choosing the pair of points from U as proposed in [14], thus achieving the running time of $O(|V|^2\ |U|\ log|V|)$. However, their approach is not appropriate for our datasets because it is possible one vector dataset is in detailed level while other vector dataset is represented abstractly.

In fact, in our previous work [15], we utilized the similar technique to match two point sets detected from a raster map and an image. More precisely, in [15], we proposed an enhanced point pattern matching algorithm to find the overlapping area of a map and an imagery by utilizing map-scale to prune the search space of possible point pattern matches (by reducing the numbers of potential matching point pairs needed to be examined). In the following sections, we focus on finding the matching



(a) The two networks to compare

(b) Using *Point connectivity* and *Angles of the point* to prune the search space

(c) Using *Angles between the points* to prune the search space

Figure 4: Comparing two road networks by using Geo-PPM

between different road networks and developing more efficient techniques by utilizing some additional spatial information that can be inferred from the road vector datasets. In addition, we also discuss how to prioritize the potential matching point pairs needed to be examined.

2.4 Enhanced PPM Algorithm: Prioritized Geo-PPM

Due to the poor performance of the brute-force point pattern matching algorithm mentioned in the previous section, PPM cannot be applied to large datasets where the number of points (or intersections) is in the order of thousands (such as the road networks covering large areas). Consequently, we utilize some auxiliary information that can be extracted from the road vector data to improve the performance of PPM for larger road networks. With the goal to reduce the numbers of potential matching point pairs needed to be examined, the intuition here is to exclude all unlikely matching point pairs. For example, given a point pair (x_1, y_1) and (x_2, y_2) in S_1, we only need to consider pairs (x'_1, y'_1) and (x'_2, y'_2) in S_2 as candidate pairs such that the real world distance and angle between (x_1, y_1) and (x_2, y_2) is close to the real world distance and angle between (x'_1, y'_1) and (x'_2, y'_2). In addition, (x'_1, y'_1) would be considered as a possible matching point for (x_1, y_1) if and only if they have similar connectivity and road directions. We categorize the auxiliary information we utilize to the following groups.

1. *Point connectivity*: We define the connectivity of a point as the number of the road segments that intersect at that point. Clearly, if datasets S_1 and S_2 have very close densities (i.e., number of intersections per one unit of area), a candidate matching point P'_1 in S_2 for a point P_1 in S_1 must have the same connectivity as P_1. Note that if the densities of the datasets are different (i.e., one dataset is detailed and the other one is represented abstractly), this condition will not be valid for a large portion of the intersections and may only be valid for major roads' intersections.

2. *Angles of the point:* The angles of a point are defined as the angles of the road segments that intersect at that point. Similar to the connectivity, a point P'_1 in S_2 can only be considered as a candidate for point P_1 in S_1 only if the two points have similar angles, or the difference between their angles is less than a threshold value. To illustrate, consider comparing two road networks as the example shown in Figure 4(a). Whenever the system chooses a point (as the point shown in the left figure of Figure 4(b)) in one road network, it only has to consider the candidate matched points with same connectivity and similar directions of intersected road segments from the other network (as some possible candidates marked in the right figure of Figure 4(b)). Note that if the densities of the datasets are different (i.e., one dataset is detailed and the other one is represented abstractly), this condition will not be valid for a large portion of the intersections and may only be valid for major roads' intersections.

3. *Angle between the points:* The angle between two points is defined as the angle of the straight line that connects the points. Clearly, a pair (P'_1,P'_2) can be considered as a possible candidate for the pair (P_1,P_2) only if the angle between P'_1 and P'_2 is similar to the angle between P_1 and P_2, or the difference between their angles is less than a threshold value. Note that this feature can only be utilized when the second dataset is not rotated and has the same direction as the first dataset. Consider the example shown in Figure 4(c). Whenever the system chooses a point pair (as the point pair shown in the left figure of Figure 4(c) and the angle between these two points is about 110 degree) in one road network, it only has to consider the candidate matched point pairs with the

similar angle (as some possible candidate point pairs marked as dash lines in the right figure of Figure 4(c)).

4. *Distance between the points:* The distance between two points is defined as the length of the straight line that connects the points in Euclidean space. Similar to the previous case, a pair (P'_1,P'_2) can be considered as a possible candidate for the pair (P_1,P_2) only if the length of the line connecting P'_1 and P'_2 is similar to the length of the line connecting P_1 and P_2, or the difference between the lengths is less than a threshold value. Note that this feature can only be utilized when the relationship between the geometry of the datasets is known and hence, the distances between objects in two datasets are comparable.

By applying the above conditions simultaneously, the Geo-PPM approach can be defined as a specialization of PPM where only the candidate pairs that have similar point connectivity, angles of the point, angles between the points, and distances between the points, will be considered. This will greatly reduce the size of the search space. However, this is still a very complex approach when the number of points in the datasets is in the order of thousands. Hence, we propose prioritized Geo-PPM that can dramatically reduce the complexity of Geo-PPM for large networks by examining the points that have the minimum number of candidates.

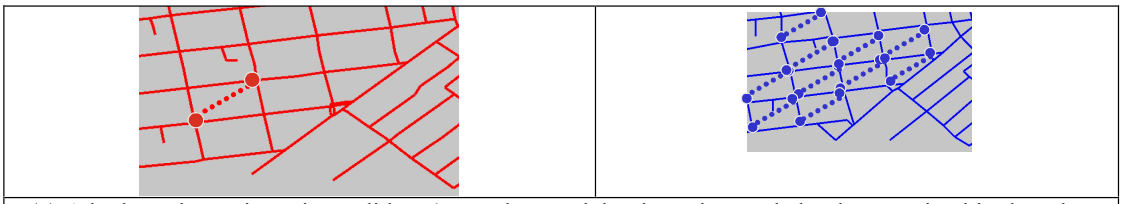
Prioritized Geo-PPM

The intuition behind prioritized Geo-PPM is to increase the possibility of examining the correct matching pair from the candidates by first examining the pairs of points that have the minimum number of candidates. Suppose that there are n_1, n_2, n_3 and n_4 points in the pool of candidates for points P_1, P_2, P_3, and P_4, respectively. This means that the number of possible candidate pairs for (P_1, P_2) and (P_3, P_4) that must be examined by Geo-PPM is n_1n_2 and n_3n_4, respectively. Note that the values of n_1 to n_4 could be very large, especially for urban areas where the road networks follow a grid pattern and hence, a large portion of the intersections have the same connectivity and angles. Also note that from these possible candidate pairs, only (a maximum of) one pair is the correctly matching one. Hence, by first examining the combination that contains the minimum number of points, we can significantly increase the possibility of finding the correct matching pair sooner. Consider the example shown in Figure 5. Our system can start the matching process by first examining the combination that contains the minimum number of points. As the point pair chosen in the left figure of Figure 5(b), it has less potential matching point pairs as shown in the right figure of Figure 5(b), comparing to the point pair examined in the left figure of Figure 5(a).

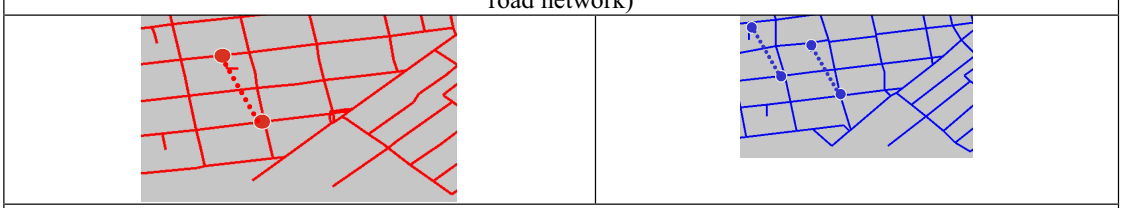
3. Evaluations

We performed several experiments with real world datasets to examine the performance of our prioritized Geo-PPM. We used three road networks obtained from USGS, NGA and US Census, covering the streets in the area of (-122.5015, 37.78) to (-122.3997, 37.8111). Figure 6(a) shows USGS road network with accurate geometry but with poor attributes. Figure 6(b) shows the US Census TIGER/Lines road network with rich attributes (e.g., road names, road classifications) but with poor geometry. Figure 6(c) shows the NGA road network with some specific attributes (e.g., road surface type). Also note that, as shown in the figure, while the data from USGS and US Census have almost similar granularity, the NGA data is an abstract level data (i.e., only major roads are stored). We manually transformed each dataset to unknown geometry systems by multiplying and subsequently adding different values to latitudes and longitudes of the vector objects in each dataset. Moreover, we filtered the south west quarter of the datasets to generate datasets with smaller sizes to examine how our approach behaves for different sizes of data.

Figure 7 shows the partial result of matched feature point sets for the USGS and US Census road networks. We also performed a quantitative analysis to measure the performance of our approach. Toward that end, we developed two metrics, precision and recall, to measure the performance of our Geo-PPM technique, since the accuracy of the matched points significantly affects the matching of the two road networks. Let the point pattern generated by Geo-PPM be defined as a set:

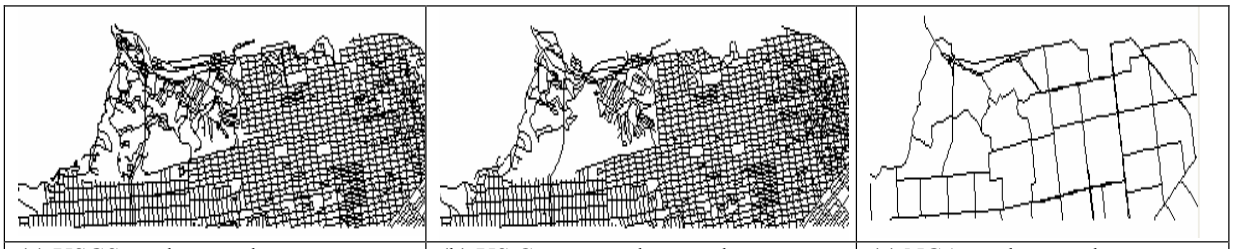


(a) A bad starting point pair candidate (several potential point pairs needed to be examined in the other road network)



(b) A better starting point pair candidate (only two potential point pairs needed to be examined in the other road network)

Figure 5: Picking up proper point pair by using Prioritized Geo-PPM



(a) USGS road network | (b) US Census road network | (c) NGA road network

Figure 6: Different road networks used in the experiments

Ret_{Pat}={(m_i, s_j) | where m_i is the intersections on the first vector dataset and s_j is the corresponding intersections located by prioritized Geo-PPM}

To measure the performance of Geo-PPM, we need to compare the set Ret_{Pat} with respect to the real matched point pattern set Rel_{Pat} (defined in Section 2.3).

Using this term, we define

$$\text{Precision}=\frac{|Ret_{pat} \cap Rel_{pat}|}{|Ret_{pat}|}$$

$$\text{Recall}=\frac{|Ret_{pat} \cap Rel_{pat}|}{|Rel_{pat}|}$$

Intuitively, precision is the percentage of correctly matched road intersections with respect to the total matched intersections detected by prioritized Geo-PPM. Recall is the percentage of the correctly matched road intersections with respect to the actual matched intersections. Table 1 shows the results of our experiments for three combinations of these datasets. As shown in the table, the average number of candidates (i.e., the number of points in the second dataset with the same connectivity and angles as compared to a point in the first dataset) varies between 371 and 637. This shows that the possibility of selecting 2 pairs from the candidate pool which are exactly matched to 2 points selected from the first dataset is very low, meaning that random selection of points in Geo-PPM will result to a very large number of possibilities and hence, to a very large processing time. For example, for the USGS+US Census combination, the possibility that randomly selected pair of points from the pool of candidates is exactly matched to the pair of points selected from the first dataset is $\frac{1}{637}\times\frac{1}{637}=\frac{1}{405769}$. However, as shown in the table, by utilizing prioritized Geo-PPM we could achieve an acceptable precision (i.e., over 80% for USGS+NGA data and over 90% for other cases) and recall (i.e., over 90%) by examining between 33 and 52 candidate pairs. This means that using the prioritized Geo-PPM, the possibility of selecting the actual matching pair is between $\frac{3}{100}$ to $\frac{2}{100}$, which is up to 4 orders of magnitude better than that of Geo-PPM.

4. Related Work

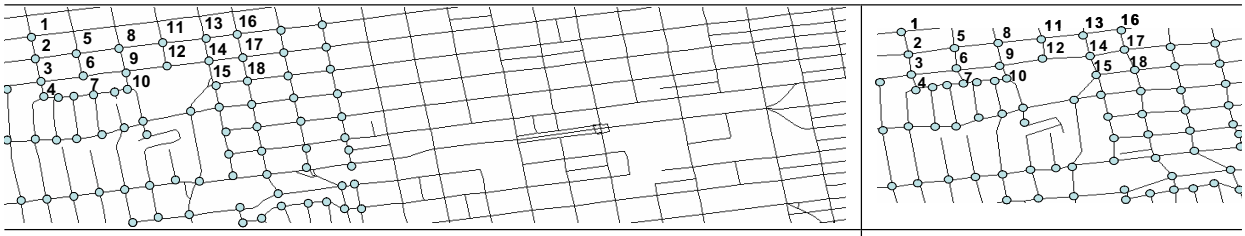
There have been a number of efforts to automatically or semi-automatically detect matched features across different road vector datasets [3, 4, 5, 6, 7]. Given a feature point from one dataset, these approaches utilize different matching strategies to discover the corresponding point within a predetermined distance (i.e., a localized area). This implies that these existing algorithms only handle the matching of vector datasets in the same geometry systems (i.e., the same coordinate system). Hence, to the best of our knowledge, no general method exits to resolve the matching of two vector data in unknown geometry systems. In addition, various GIS systems (such as ESEA MapMerger [9]) have been implemented to achieve the matching of vector datasets with different accuracies. However, most of the existing systems require manual interventions to transform two road networks into same geocoordinates beforehand. Thus, they are not suitable for handling road networks in unknown geometry systems, while our approach can match two road networks in unknown geometry systems.

Finally, our approach discussed in this paper utilizes a specialized point pattern matching algorithm to find the corresponding point pairs on both datasets. The geometric

Datasets	USGS+ US Census	USGS+ NGA	USGS+ US Census
Number of Intersections	2367 + 2456	2367 + 133	920 + 1035
Average Number of Candidates	637	514	371
Point Pairs Examined	43	52	33
Processing Time	946 sec.	48 sec.	132 sec.
Precision	91%	82%	95.8%
Recall	92.5%	96.5%	95.8%

Table 1: Experimental Results (on a PC with 3.2GHz CPU)

[9] http://www.esea.com/products/



(a) USGS road network (b) U.S. Census road network

Figure 7: The partial result of matched points from two road networks (some matched points are labelled in order to show the corresponding points)

point set matching in two or higher dimensions is a well-studied family of problems with application to area such as computer vision, biology, and astronomy [12, 14].

5. Conclusion and Future Work

In this paper, we proposed an efficient and accurate technique, termed prioritized Geo-PPM, to locate the matched points between two road network datasets when the spatial attributes of the datasets are in unknown systems. In our solution, we first select pairs of points in the first dataset with the minimum number of candidates (i.e., point with similar connectivity and angles) in the second dataset, and then perform our PPM method on these pairs. Although our technique matches road networks at the point level (not at the road segment level), it takes the road connectivity, road directions and global distribution of road intersections into consideration. Our experiments show that this approach provides acceptable precision and recall values by only examining a very small number of pairs.

We plan to extend our approach in several ways. First, we plan to examine prioritized Geo-PPM for even larger road networks and for different patterns of road networks (e.g., rural roads and urban roads), and consider the orientations of the road networks as well. Second, we intend to investigate the appropriate order of utilizing the auxiliary information described in Section 2.4. Third, we would like to perform comprehensive comparisons between our approach and the related techniques described in Section 4. Finally, we also plan to use these matched points as control points to integrate different road network datasets.

6. Acknowledgements

This research is based upon work supported in part by the National Science Foundation under Award No. IIS-0324955. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any of the above organizations or any person connected with them.

7. Reference

1. Barclay, T., Gray, J., and Stuz, D. *Microsoft TerraServer: A Spatial Data Warehouse*. in *the 2000 ACM SIGMOD International Conference on Management of Data*. 2000. Dallas, TX: ACM Press.
2. Chiang, Y.-Y. and Knoblock, C.A. *Classification of Line and Character Pixels on Raster Maps Using Discrete Cosine Transformation Coefficients and Support Vector Machines*. in *the 18th International Conference on Pattern Recognition*. 2006.
3. Walter, V. and Fritsch, D., *Matching Spatial Data Sets: a Statistical Approach.* International Journal of Geographic Information Sciences, 1999. 13(5): p. 445-473.
4. Ware, J.M. and Jones, C.B. *Matching and Aligning Features in Overlayed Coverages*. in *the 6th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS'98)*. 1998. Washington, D.C: ACM Press.
5. Cobb, M., Chung, M.J., Miller, V., Foley, H.I., Petry, F.E., and Shaw, K.B., *A Rule-Based Approach for the Conflation of Attributed Vector Data.* GeoInformatica, 1998. 2(1): p. 7-35.
6. Saalfeld, A., *Conflation: Automated Map Compilation.* International Journal of Geographic Information Sciences, 1988. 2(3): p. 217-228.
7. Chen, C.-C., Shahabi, C., and Knoblock, C.A. *Utilizing Road Network Data for Automatic Identification of Road Intersections from High Resolution Color Orthoimagery*. in *the Second Workshop on Spatio-Temporal Database Management, colocated with VLDB*. 2004. Toronto, Canada.
8. Chen, C.-C., Thakkar, S., Knoblok, C.A., and Shahabi, C. *Automatically Annotating and Integrating Spatial Datasets*. in *the 8th International Symposium on Spatial and Temporal Databases (SSTD'03)*. 2003. Santorini Island, Greece.
9. Habib, A., Uebbing, R., Asmamaw, A, *Automatic Extraction of Primitives for Conflation of Raster Maps*. 1999, The Center for Mapping, The Ohio State University.
10. Flavie, M., Fortier, A., Ziou, D., Armenakis, C., and Wang, S. *Automated Updating of Road Information from Aerial Images*. in *the American Society Photogrammetry and Remote Sensing Conference*. 2000. Amsterdam, Holland.
11. Guttman, A. *R-trees: a dynamic index structure for spatial searching*. in *the SIGMOD Conference*. 1984. Boston, MA.
12. Chew, L.P., Goodrich, M.T., Huttenlocher, D.P., Kedem, K., Kleinberg, J.M., and Kravets, D. *Geometric pattern matching under Euclidean motion*. in *the Fifth Canadian Conference on Computational Geometry*. 1993.
13. Cardoze, D.E. and Schulman, L.J. *Pattern Matching for Spatial Point Sets*. in *the IEEE Symposium on Foundations of Computer Science*. 1998.
14. Irani, S. and Raghavan, P., *Combinatorial and experimental results for randomized point matching algorithms.* Computational Geometry, 1999. 12(1-2): p. 17-31.
15. Chen, C.-C., Knoblock, C.A., Shahabi, C., Chiang, Y.-Y., and Thakkar, S. *Automatically and Accurately Conflating Orthoimagery and Street Maps*. in *the 12th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS'04)*. 2004. Washington, D.C: ACM Press.

Update-efficient Indexing of Moving Objects in Road Networks

Jidong Chen Xiaofeng Meng Yanyan Guo Zhen Xiao

School of Information, Renmin University of China
{chenjd, xfmeng, guoyy, xiaozhen}@ruc.edu.cn

Abstract

Recent advances in wireless sensor networks and positioning technologies have boosted new applications that manage moving objects. In such applications, a dynamic index is often built to expedite evaluation of spatial queries. However, development of efficient indexes is a challenge due to frequent object movement. In this paper, we propose a new update-efficient index method for moving objects in road networks. We introduce a dynamic data structure, called *adaptive unit*, to group neighboring objects with similar movement patterns. To reduce updates, an adaptive unit captures the movement bounds of the objects based on a prediction method, which considers the road-network constraints and stochastic traffic behavior. A spatial index (e.g., R-tree) for the road network is then built over the adaptive unit structures. Simulation experiments, carried on two different datasets, show that an adaptive-unit based index is efficient for both updating and querying performance.

Keywords Spatial-Temporal Databases, Moving Objects, Index Structure, Road Networks

1 Introduction

Recent advances in wireless sensor networks and positioning technologies have enabled a variety of new applications such as traffic management, fleet management, and location-based services that manage continuously changing positions of moving objects [11, 12]. In such applications, a dynamic index is often built to expedite evaluation of spatial queries. However, existing dynamic index structures (e.g. B-tree and R-tree) suffer from poor performance due to the large overhead of keeping the index updated with the frequently changing position data. Development of efficient indexes to improve the update performance is an important challenge.

Current work on reducing the index updates of moving objects mainly contains three kinds of approaches. First, most efforts [4, 9, 10, 15] focus on the update optimization of the existing multi-dimensional index structures especially the adaptation and extension of the R-tree [6]. To avoid the multiple paths search operation in the R-tree during the top-down update, recent work proposes the bottom-up approach [9, 10] and memo-based [15] structure to reduce the updates of the R-tree. Another method [4] exploits the change-tolerant property of the index structure to reduce the number of updates that cross the MBR boundaries of R-tree.

However, the indexes based on MBRs exhibit high concurrency overheads during node splitting, and each individual update is still costly. Therefore, some index methods based on a low-dimensional index structure (e.g. B^+-tree) are proposed [7, 16], which construct the second category of index methods. They combine the dimension reduction and linearization technique with a single B^+-tree to efficiently update the index structure.

The third kind of approaches use a prediction method with a time-parameterized function to reduce the index updates [12, 13, 14]. They describe a moving object's location by a linear function and the index is updated only when the parameters of the function change, for example, when the moving object changes its speed or direction. The MBRs of the index vary with the time as a function of the enclosed objects. However, the linear prediction is hard to reflect the movement in many real application and therefore leads to low prediction accuracy and frequent updates.

Though these index structures solve the problem of index updates to some extent, they are designed to index objects performing free movement in a two-dimensional space. We focus on the index update problem in real life environments, where the objects move within constrained networks, such as vehicles on roads. In such setting, the spatial property of objects' movement is captured by the network. Therefore, the

Proceedings of the third Workshop on STDBM Seoul, Korea, September 11, 2006

spatial location of moving objects can be indexed by means of the road-network index structure. For example, moving objects can be accessed by each road segment indexed by the R-tree. Since the road network seldom change and objects just move from one part to the other part of the network, the R-tree in this case remains fixed. Existing index work that handles network-constrained moving objects [1, 5, 11] is based on this feature. They separate spatial and temporal components of the moving objects' trajectories and index the spatial aspect by the network with a R-tree. However, they are mostly concerned with the historical movement and therefore they do not consider the problem of index updates.

In this paper, we address the problem of efficient indexing of moving objects in road networks to support heavy loads of updates. We exploit the constraints of the network and the stochastic behavior of the real traffic to achieve both high update and query efficiency. We introduce a dynamic data structure, called *adaptive unit* (AU for short) to group neighboring objects with similar movement patterns in the network. A spatial index (e.g., R-tree) for the road network is then built over the adaptive units to form the index scheme for moving objects in road networks. The index scheme optimizes the update performance for the following reasons: (1) An AU functions as a one-dimensional MBR in the TPR-tree [13], while it minimizes expanding and overlaps by considering more movement features. (2) The AU captures the movement bounds of the objects based on a prediction method, which considers the road-network constraints and stochastic traffic behavior. (3) Since the movement of objects is reduced to occur in one spatial dimension and attached to the network, the update of the index scheme is only restricted to the update of the AUs. We have carried out extensive experiments based on two datasets. The results show that an adaptive-unit based index not only improves the efficiency of each individual update but also reduces the number of updates and is efficient for both updating and querying performance.

The main contributions of this paper are:

- The introduction of Adaptive Units that optimize for frequent index updates of moving objects in road networks.
- An experimental evaluation and validation of the efficient update as well as query performance of the proposed index structure.

The rest of the paper is organized as follows. Section 2 surveys related work and introduces underlying model. Section 3 describes the structure and algorithms of adaptive units for efficient updates. Section 4 contains algorithm analysis and experimental evaluation. We conclude and propose the future work in Section 5.

2 Related Work and Underlying Model

2.1 Related Work

There are lots of efforts at reducing the need for index updates of moving objects. In summary, they can be classified into three categories.

First, most work focuses on the update optimization of existing multi-dimensional index structures especially the adaptation and extension of the R-tree [6]. The top-down update of R-tree is costly since it needs several paths for searching the right data item considering the MBR overlaps. In order to reduce the overhead, Kwon et al. [9] develop the Lazy Update R-tree, which is updated only when an object moves out of the corresponding MBR. With adding a secondary index on the R-tree, it can perform the update operation in the bottom-up way. Recently, by exploiting the change-tolerant property of the index structure, Cheng et al. [4] present the CTR-tree to maximize the opportunity for applying lazy updates and reduce the number of updates that cross MBR boundaries. [10] extends the main idea of [9] and generalizes the bottom-up update approach. However, they are not suitable to the case where consecutive changes of objects are large. Xiong and Aref [15] present the RUM-tree that processes R-tree updates in a memo-based approach, which eliminates the need to delete the old data item during an index update. Therefore, its update performance is stable with respect to the changes between consecutive updates. In our index structure, however, the R-tree remains fixed since it indexes the road network and only the adaptive units are updated.

The second type of methods are based on the dimension reduction technique [11] and a low-dimensional index [7, 16] (e.g. B^+-tree). The B^x-tree [7, 16] combine the linearization technique with a single B^+-tree to efficiently update the index structure. They uses space filling curves and a pre-defined time interval to partition the representation of the locations of the moving objects. This makes the B^+-tree capable to index the two-dimensional spatial locations of moving objects. Therefore, the cost of individual update of index is reduced. However, the B^x-tree imposes discrete representation and may not keep the precise values of location and time during the partitioning. For our setting, the two-dimensional spatial locations of moving objects can be reduced to the 1.5 dimensions [8] by the road network where objects move.

The techniques in third category use a prediction method represented as the time-parameterized function to reduce the index updates [12, 13, 14]. They store the parameters of the function, e.g. the velocity and the starting position of an object, instead of the real positions. In this way, they update the index structure only when the parameters change (for example, the speed or the direction of a moving object changes). The Time-Parameterized R-tree (TPR-tree) [13] and its variants (e.g. TPR*-tree) [12, 14] are

the examples of this type of index structures. They all use a linear prediction model, which relates objects' positions as a linear function of time.However, the linear prediction is hard to reflect the movement in many real application especially in traffic networks where vehicles change their velocities frequently. The frequent changes of the object's velocity will incur repeated updates of the index structure. Our technique also fall into this category and apply an accurate prediction method we proposed in [3] by considering more transportation features.

Several methods have been proposed for indexing moving objects in spatially constrained networks. Pfoser et al. [11] propose to convert the 3-dimensional problem into two sub-problems of lower dimensions through certain transformation of the networks and the trajectories. Another approach, known as the FNR-tree [5], separates spatial and temporal components of the trajectories and indexes the time intervals that each moving object spends on a given network link. The MON-tree approach [1] further improves the performance of the FNR-tree by representing each edge by multiple line segments (i.e. polylines) instead of just one line segment. However, they all focus on the historical movement and cannot support frequent index updates. To the best of our knowledge, there is no current index method to support efficient updates of moving objects in road networks.

2.2 Underlying Model

We use the GCA model we proposed in [3] to model the network and moving objects. A road network is modeled as a graph of cellular automata (GCA), where the nodes of the graph represent road intersections and the edges represent road segments with no intersections. Each edge consists of a cellular automaton (CA), which is represented, in a discrete mode, as a finite sequence of cells.

In the GCA, a moving object is represented as a symbol attached to the cell and it can move several cells ahead at each time unit. Intuitively, the velocity is the number of cells an object can traverse during a time unit. The motion of an object is represented as some (time, location) information. Generally, such information is treated as a trajectory.

3 The Adaptive Unit

3.1 Structure and Storage

Conceptually, an adaptive unit is similar to a one-dimensional MBR in the TPR-tree, that expands with time according to the predicted movement of the objects it contains. However, in the TPR-tree, it is possible that an MBR may contain objects moving in opposite directions, or objects moving at different speeds. As a result, the MBR may expand rapidly, which may create large overlaps with other MBRs. The AU avoids this problem by grouping objects having similar moving patterns. Specifically, for objects in the same network edge, we use a distance threshold and a speed threshold to cluster the adjacent objects with the same direction and similar speed. In comparison, the AU has no obvious enlargement because objects in the AU move in a cluster.

We now formally introduce the AU. An AU is a 8-tuple:

$$\begin{aligned} AU \quad = \quad & (\texttt{auID}, \texttt{objSet}, \texttt{upperBound}, \texttt{lowerBound}, \\ & \texttt{edgeID}, \texttt{enterTime}, \texttt{exitTime}, \texttt{auInitLen}) \end{aligned}$$

where `auID` is the identifier of the AU, `objSet` is a list that stores objects belonging to the AU, `upperBound` and `lowerBound` are upper and lower bounds of predicted future trajectory of the AU. The trajectory bounds will be explained in details in Section 3.3. We assume the functions of trajectory bounds as follows:

$$\begin{aligned} \texttt{upperBound}: \quad & D(t) = \alpha_u + \beta_u \cdot t \\ \texttt{lowerBound}: \quad & D(t) = \alpha_l + \beta_l \cdot t \end{aligned}$$

`edgeID` denotes the network edge that the AU belongs to, `enterTime` and `exitTime` record the time when the AU enters and leaves the edge and `auInitLen` represents the initial length of the AU.

In the road network, multiple AUs are associated with a network edge. Since AUs in the same edge are likely to be accessed together during query processing, we store AUs by clustering on their `edgeID`. That is, the AUs in the same edge are stored in the same disk pages. To access AUs more efficiently, we create an in-memory, compact summary structure called the *direct access table* for each edge. A direct access table stores the summary information of each AU on an edge (i.e. number of objects, trajectory bounds) and pointers to AU disk pages. Each AU corresponds to an entry in the direct access table, which has the following structure `(auID, upperBound, lowerBound, auPtr, objNum)`, where `auPtr` points to a list of AUs in disk storage and `objNum` is the number of objects included in the AU. In order to minimize I/O cost, we use the direct access table to filter AUs and only access the disk pages when necessary.

3.2 The Index Scheme

We build a spatial index (e.g., R-tree) for the road network over the adaptive units to form the index scheme for the network-constrained moving objects. The AU index scheme is a two-level index structure. At the top level, it consists of a 2D R-tree that indexes spatial information of the road network. On the bottom level, its leaves contain the edges representing road segments included in the corresponding MBR of the R-tree and point to the lists of adaptive units. The top level R-tree remains fixed during the lifetime of the index scheme (unless there are changes in the network). The index scheme is developed with the R-tree

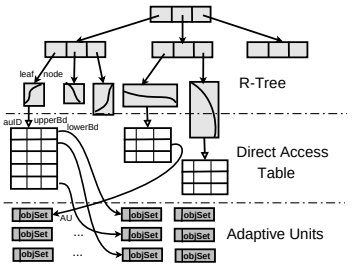
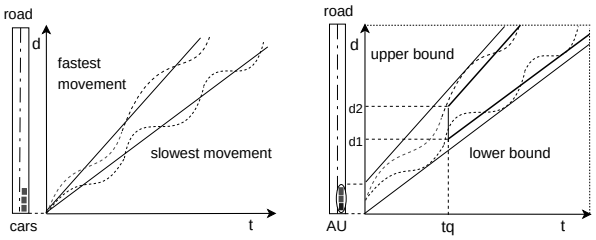


Figure 1: Structure of the AU index scheme



(a) Simulated trajectories (b) Trajectory bounds

Figure 2: The simulation-based prediction

in this paper, but any existing spatial index can also be used without changes.

Figure 1 shows the structure of the AU index scheme, which also includes the direct access table. The R-tree and adaptive units are stored in the disk. However, the direct access table is in the main memory since it only keeps the summary information of adaptive units. In the index scheme, each leaf node of the R-tree can be associated with its direct access table by its `edgeID` and the direct access table can connect to corresponding adaptive units by `auPtr` in its entries. Therefore, we only need to update the direct access table when AUs change, which greatly enhances the performance of the index scheme.

3.3 Optimizing for Updates

An important feature of the AU is that it groups objects having similar moving patterns. The AU is capable of dynamically adapting itself to cover the movement of the objects it contains. By tightly bounding enclosed moving objects for some time in the future, the AU alleviates the update problem of MBR rapid expanding and overlaps in the TPR-tree like methods.

For reducing the updates further, the AU captures the movement bounds of the objects based on a prediction method we proposed in [3], which considers the road-network constraints and stochastic traffic behavior. Since objects in an AU have similar movement, we then predict the movement of the AU, as if it were a single moving object. In the following, we describe the application and adaptation of the prediction method to the AU.

We use GCAs not only to model road networks, but also to simulate the movements of moving objects by the transitions of the GCA. Based on the GCA, the *Simulation-based Prediction (SP)* method to anticipate future trajectories of moving objects is proposed. The SP method treats the objects' simulated results as their predicted positions. Then, by the linear regression, a compact and simple linear function that reflects future movement of a moving object can be obtained. To refine the accuracy, based on different assumptions on the traffic conditions we simulate two future trajectories to obtain its predicted movement function. Specifically, we extend the CA model used in traffic flow simulation for predicting the future trajectories of objects by setting $P_d(i)$ to values that model different traffic conditions. In this setting, $P_d(i)$ is treated as a random variable to reflect the stochastic, dynamic nature of traffic system. By giving $P_d(i)$ two values (e.g. 0 and 0.1 in our experiments), we can derive two future trajectories, which describe, respectively, the fastest and slowest movements of objects. Finally, we translate the two regression lines, until all estimated future positions fall within to obtain the predicted trajectory bounds. The SP method is shown in Figure 2. Through the SP method, we obtain two predicted future trajectory bounds of objects. We apply this technique to the AU - a set of moving objects that have similar movement and are treated as one object.

The future trajectory bounds are predicted as soon as AU is created. The trajectory bounds will not be changed along the edge that the AU moves on until the objects in the AU move to another edge in the network. It is evident that the range of predicted bounds of AU will become wider with the time, which leads to lower accuracy of future trajectory prediction. However, if we issue another prediction when the predicted bounds are not accurate any more, the costs of simulation and regression are high. Considering that the movement of objects along one network edge is stable, we can assume the same trends of the trajectory bounds and adjust only the initial locations when the prediction is not accurate. Specifically, when the predicted position exceeds its actual position above the predefined accuracy, the AU treats its actual locations (the locations of the boundary objects) at that time as the initial locations of the two trajectory bounds and follow the same movement vector (e.g. slope of the bounds) as the previous bounds to provide more accurate predicted trajectory bounds. In this way, the predicted trajectory bounds can be effectively revised with few costs. Figure 2(b) shows the adaptation of the trajectory bounds. t_q is the time slice when actual locations of boundary objects in the AU exceeds the predicted bounds of the AU above precision threshold and the d_1,d_2 are the actual locations of the first object and last object respectively in the AU. The trajectory bounds are revised according to the actual locations and the original bounds' slopes. Therefore, without executing more prediction, the prediction accuracy of the objects' future trajectories can be kept high.

Since the R-Tree indexes the road network, it remains fixed, and the update of the AU index scheme restricts to the update of adaptive units. Specifically, an AU is usually created at the start of one edge and dropped at the end of the edge. Since the AU is a one-dimensional structure, it performs update operations much more efficiently than the two-dimensional indexes. We will describe these operations in details.

3.4 Update Operations

The update of an AU can be of the following form: creating an AU, dropping an AU, adding objects to an AU and removing objects from an AU.

Creating an AU

To create an AU, we first compose the $objSet$ – a list of objects traveling in the same direction with similar velocities, and in close-by locations. We then predict the future trajectories of the AU by simulation and compute its trajectory bounds. In fact, we treat the AU as one moving object (the object closest to the center of the AU) and predict its future trajectory bounds by predicting this object. The prediction starts when the AU is created and ends at the end the edge. Finally, we write the created AU to the disk page and insert the AU entry to its summary structure.

Dropping an AU

When objects in an AU move out of the edge, they may change direction independently. So we need to drop this AU and create new AUs in adjacent edges to regroup the objects. When the front of an AU touches the end of the edge, some objects in the AU may start moving out of the edge. However, the AU cannot be dropped because a query may occur at that time. Only after the last object in the AU enters another edge and joins another AU, can the AU be dropped. Dropping an AU is simple. Through its entry in direct access table, we find the AU and delete it.

Adding and removing objects from an AU

When an object leaves an AU, we remove this object from the AU and find another AU in the neighborhood to check if the object can fit that AU. If it can, the object will be inserted into that AU, otherwise, a new AU is created for this object. Specifically, when adding an object into an AU, we first find the direct access table of the edge that the object lies and, by its AU entry in the table, access the AU disk storage. Finally, we insert into the objects list of the AU and update the AU entry in the direct access table. Removing an object from an AU has the similar process.

Therefore, when updating an object in the AU index scheme, we first determine whether the object is leaving the edge and entering another one. If it is moving to another edge, we delete it from the old AU (if it is the last object in the old AU, the AU is also dropped) and insert it into the nearest AU or create a new AU in the edge it is entering. Otherwise, we do not update the AU that the object belongs to unless its position exceeds the bounds of the AU. In that case, we execute the same updates as those when it moves to another edge or only revise the predicted trajectory bounds of the AU. Factually, we find, from the experiment evaluation, that the chances that objects move beyond the trajectory bounds of its AU on an edge are very slim. The algorithm 1 shows the update algorithm of AUs.

Algorithm 1: Update($objID, position, velocity, edgeID$)

input : $objID$ is the object identifier, $position$ and $velocity$ are its position and velocity, $edgeID$ is the edge identifier where the object lies

Find AU where $objID$ is included before update;
if $AU.edgeID \neq edgeID$ *or (position* $<$ $AU.lowerBound$ *or position* $>$ $AU.upperBound$*)*
then
  `// The object moves to a new edge or exceeds bounds of its original AU`
  Find the nearest AU AU_1 for $objID$ on $edgeID$;
  if $GetNum(AU_1.objSet) < MAXOBJNUM$ *and* $ObjectFitAU(objID, position, velocity, AU_1)$ **then**
    InsertObject($objID, AU_1.auID, AU_1.edgeID$);
  else $AU_2 \leftarrow$ CreateAU($objID$,$edgeID$);
  if $GetNum(AU.objSet) > 1$ **then**
    DeleteObject($objID, AU.auID, AU.edgeID$);
  else DropAU($AU.edgeID, AU.auID$);
end

In summary, updating the AU-based index is easier than updating the TPR-tree. It never invoke any complex node splitting and merging. Moreover, thanks to the similar movement features of objects in an AU and the accurate prediction of the SP method, the objects are seldom removed or added from their AU on an edge, which reduces the number of index updates.

3.5 Query Algorithm

Query processing in the AU index scheme is straightforward. Given a query, we use the top level R-tree to get the edges involved and then scan the direct access tables of the edges. With the `upperBound` and the `lowerBound` in the direct access table, we can easily find AU entries that intersect the query, and then visit the disk pages to get more information about these AUs. For space limitation, we just take window range query for example. Given a range with (X_1, Y_1, X_2, Y_2), we first perform a spatial range search in the top level R-Tree to locate the edges (e.g. $e_1, e_2, e_3, \ldots$). For each selected edge e_i, we transform the original search (X_1, Y_1, X_2, Y_2) to a 1D search range (S_1, S_2) $(S_1 \leq S_2)$, where S_1 and S_2 are the relative distances from the start vertex along the edge e_i. In the case of multiple intersecting edges, we can divide the query range into several sub-ranges by edges and apply the transformation method to each edge. The method is also applicable to the various modes

that the query and edges intersect. Here, we only illustrate the case when the query window range only intersects one edge and compute its relative distances S_1 and S_2. It can be easily extended to other cases. Suppose $X_{start}, Y_{start}, X_{end}, Y_{end}$ are the start vertex coordinates and the end vertex coordinates of the edge e_i. According to Thales Theorem about similar triangles, we obtain S_1 and S_2 as follows:

$$\begin{aligned} r &= \sqrt{(X_{start} - X_{end})^2 + (Y_{start} - Y_{end})^2} \\ S_1 &= \frac{X_1 - X_{start}}{X_{end} - X_{start}} r \\ S_2 &= \frac{Y_1 - Y_{start}}{Y_{end} - Y_{start}} r \end{aligned}$$

The transformed query (S_1, S_2) is then executed in each of the AUs in the direct access table of the corresponding edge e_i. By the trajectory bounds of the AU, we can determine whether the transformed query intersects the AU, thus filtering the unnecessary AUs quickly. Finally, we access the selected AUs in disk storage and return the objects satisfying the query window. In summary, the query processing is efficient due to the grouping of similar objects in AUs and the dimensionality reduction of the query.

4 Performance Analysis

We evaluate the AU index scheme (denoted as "AU index") by comparing it with the TPR-tree and the AU index scheme when the direct access table is not used (denoted as "AU index without DT"). We measure their their update performance with the individual update, update frequency and total update costs and their query performance.

4.1 Datasets

We use two datasets for our experiments. The first is generated by the CA simulator, and the second by the Brinkhoff's Network-based Generator [2]. We use the CA traffic simulator to generate a given number of objects in a uniform network of size 10000×10000 consisting of 500 edges. Each object has its route and is initially placed at a random position on its route. The initial velocities of the objects follow a uniform random distribution in the range $[0, 30]$. The location and velocity of every object is updated at each time-stamp. The Brinkhoff's Network-based Generator is used as a popular benchmark in many related work. The generator takes a map of a real road network as input (our experiment is based on the map of Oldenburg including 7035 edges). The positions of the objects are given in two dimensional X-Y coordinates. We transform them to the form of (`edgeid`, `pos`), where `edgeid` denotes the edge identifier and `pos` denotes the object relative position on the edge. The generator places a given number of objects at random positions on the road network, and updates their locations at each time-stamp.

Table 1: Parameters and their settings

Parameters	Settings
Page size	4K
Node capacity	100
Numbers of queries	200
Numbers of mo(cars)	10K, ... , 50K, ... , **100K**
Numbers of updates	**100K**, ... , 500K, ... , 1M
Dataset Generator	CA Simulator, Network-based Generator

We implemented both the AU index scheme and the TPR-tree in Java and carried out experiments on a Pentium 4, 2.4 GHz PC with 512MB RAM running Windows XP. To improve the performance of the index structure, we employed a LRU buffer of the same size as the one used in the TPR-tree [13]. We summarize workload parameters in Table 1, where values in bold are default values.

4.2 Update Cost

We compare the cost of index update for the AU index and the TPR-tree in terms of the average individual update cost, update frequency and total update cost.

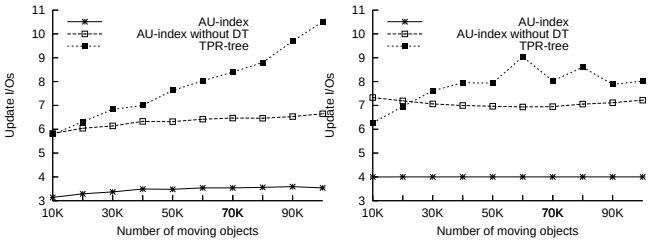
Individual Update Cost

We study the individual update performance of the index while varying the number of moving objects and updates. Figure 3 shows the average individual update cost when increasing the data size from 10K to 100K. Figure 4 shows how the performance varies over time. Clearly, updating the TPR-tree tends to be costly, and the problem is exacerbated when the data size increases. In each case of different data size and different number of updates, the AU index has much lower update cost than the TPR-tree. The main reason can be explained as follows. Each update of the TPR-tree involves the search of an old entry and a new entry, as well as the modification of the index structure (node splitting, merging, and the propagating of changes upwards). The cost increases with larger data size due to more overlaps among MBRs. The changes of index structure with the increase of data updates also affect the performance of the TPR-tree. However, the AU index has better performance because its update only restricts to the AU's update and as a one-dimensional access structure, the AU has few overlaps and incurs no cost associated with node splitting and the propagation of MBR updates.

The direct access table of the AU index has a significant contribution in improving update performance. This is because the search of the specific AU is accelerated by the in-memory structure.

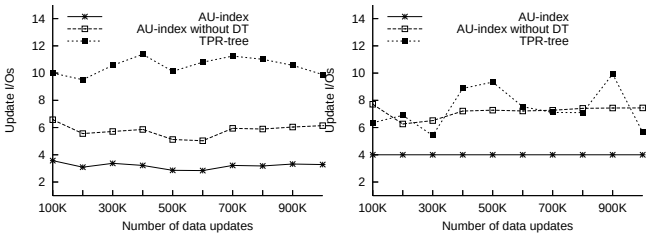
Update Frequency

Frequent updates of moving objects (a.k.a. data updates) may lead to frequent updates of index. When an object's position exceeds the MBR or AU, the index needs to be updated to delete the object from the old MBR or AU and insert it to another one. In this experiment, we measure the index update rate, which is



(a) Brinkhoff (b) CA

Figure 3: Individual Update Cost with Different Datasize



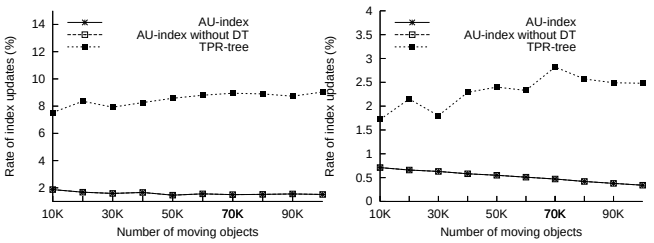
(a) Brinkhoff (b) CA

Figure 4: Individual Update Cost over Time

the ratio between number of index updates and number of data updates, for every 100K data updates and different data size. Figure 5 and 6 show that the update rate of the TPR-tree is nearly 4 to 5 times more than that of the AU index. The index update rate depends on the prediction method. In the AU index, the future positions of the object are predicted more accurately, so the object is likely to remain in its AU, which leads to fewer index updates.

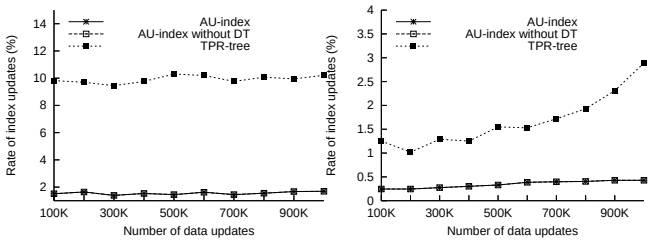
Total Update Costs

The total update costs depend on the update frequency and the average individual update cost, and it can reflect the index update performance more accurately. From both Figure 7 and 8, we can see that although the AU index has to deal with the creation and dropping of AUs, the TPR-tree incurs much higher update costs than the AU index and its performance deteriorates dramatically as data size increases. This is mainly due to the inaccuracy of the linear prediction model and the complex reconstruction of the TPR-tree (e.g. splitting and merging).



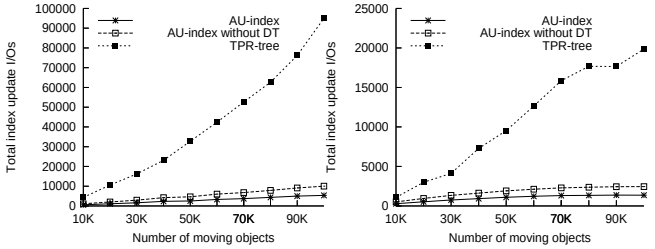
(a) Brinkhoff (b) CA

Figure 5: Index Update Frequency with Different Datasize



(a) Brinkhoff (b) CA

Figure 6: Index Update Frequency over Time



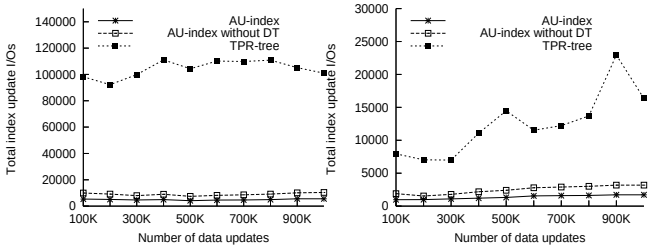
(a) Brinkhoff (b) CA

Figure 7: Total Update Cost with Different Datasize

For each data size, the update costs of the two indexes in the Brinkhoff's dataset are both higher than those in the CA dataset due to the higher complexity of road network and skewed spatial distribution of objects in the Brinkhoff's dataset.

4.3 Query Cost

We study the window range query performance of the TPR-tree and the AU index with different update settings. We increase the number of updates from 100K to 1M to examine how query performance is affected. We issued 200 range queries for every 100K updates in a 1M dataset. Figure 9 shows that the cost of the TPR-tree increases much faster as the number of updates increases. The cost of the AU index is considerably lower and is less sensitive to the number of updates. This is because the adaptive units in the AU index have much less overlaps than the MBRs in the TPR-tree, and the overlaps to a large extent determine the range query cost. Besides, as objects move apart, the amount of dead space in the TPR-tree increases,

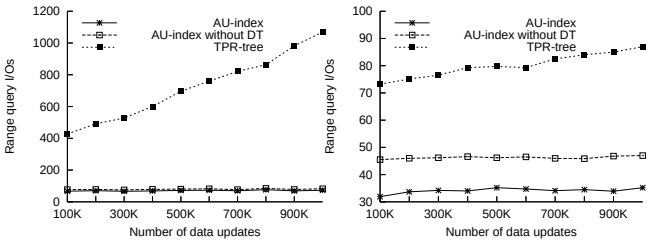


(a) Brinkhoff (b) CA

Figure 8: Total Update Cost over Time

which makes false hits more likely. Also, updates lead to the expanding and overlaps of MBRs, which further deteriorate the performance of the TPR-tree. For the AU index, the increase of the updates hardly affect the total number of AUs, and the chances of overlaps of different AUs are very slim.

We also study the query performance while varying the number of moving objects and query window size. For the space limitation, we do not report the experimental results. Also, in each case, the AU index has lower query cost than the TPR-tree and scales well.



(a) Brinkhoff (b) CA

Figure 9: Effect of Updates on Query Performance

5 Conclusions and Future Work

Indexing objects moving in a constrained network especially the road network is a topic of great practical importance. We focus on the index update issue for the current positions of network-constrained moving objects. We introduce a new access structure, adaptive unit that exploits as much as possible the characteristics of the movements of objects. The updates of the structure are minimized by an accurate prediction method which produces two trajectory bounds based on different assumptions on the traffic conditions. The efficiency of the structure also results from the possible reduction of dimensionality of the trajectory data to be indexed. Our experimental results performed on two datasets show that the efficiency of the index structure is one order of magnitude higher than the TPR-tree.

In the future, we will compare the update performance with the work of the R-tree-based updating optimization such as RUM-tree [15] and CTR-tree [4]. On the other hand, since the adaptive units contain the predicted future trajectories of moving objects, the predictive query algorithms can be developed naturally based on the adaptive unit-based index. Furthermore, we will extend the query algorithms to support the KNN query and continuous query for moving objects in the road network.

Acknowledgments

This research was partially supported by the grants from the Natural Science Foundation of China under grant number 60573091, 60273018; the Key Project of Ministry of Education of China under Grant No.03044; Program for New Century Excellent Talents in University (NCET); Program for Creative PhD Thesis in University. The authors would like to thank Jianliang Xu and Haibo Hu from Hong Kong Baptist University and Stéphane Grumbach from CNRS, LIAMA in China for many helpful advice and assistance.

References

[1] V. T. Almeida, R. H. Güting. Indexing the Trajectories of Moving Objects in Networks (Extended Abstract). In SSDBM, 2004, 115-118.

[2] T. Brinkhof. A framework for generating network-based moving objects. In GeoInformatica, 6(2), 2002, 153-180.

[3] J. Chen, X. Meng, Y. Guo, S. Grumbach, H. Sun. Modeling and Predicting Future Trajectories of Moving Objects in a Constrained Network. In MDM, 2006, 156 (MLASN workshop).

[4] R. Cheng, Y. Xia, S. Prabhakar, R. Shah. Change Tolerant Indexing for Constantly Evolving Data. In ICDE, 2005, 391-402.

[5] E. Frentzos. Indexing objects moving on Fixed networks. In SSTD, 2003, 289-305.

[6] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In SIGMOD, 1984, 47-57.

[7] C. S. Jensen, D. Lin, B. C. Ooi. Query and Update Efficient B+-Tree Based Indexing of Moving Objects. In VLDB, 2004, 768-779.

[8] G. Kollios, D. Gunopulos, V. J. Tsotras. On indexing mobile objects. In PODS, 1999, 261-272.

[9] D. Kwon, S. J. Lee, S. Lee. Indexing the current positions of moving objects using the lazy update R-tree. In MDM, 2002, 113-120.

[10] M. L. Lee, W. Hsu, C. S. Jensen, B. Cui, K. L. Teo. Supporting Frequent Updates in R-Trees: A Bottom-Up Approach. In VLDB, 2003, 608-619.

[11] D. Pfoser, C. S. Jensen. Indexing of network constrained moving objects. In ACM-GIS, 2003, 25-32.

[12] S. Saltenis, C. S. Jensen. Indexing of Moving Objects for Location-Based Service. In ICDE, 2002, 463-472.

[13] S. Saltenis, C. S. Jensen, S. T. Leutenegger, M. A. Lopez. Indexing the Positions of Continuously Moving Objects. In SIGMOD, 2000, 331-342.

[14] Y. Tao, D. Papadias, J. Sun. The TPR*-Tree: An Optimized Spatiotemporal Access Method for Predictive Queries. In VLDB, 2003, 790-801.

[15] X. Xiong, W. G. Aref. R-trees with Update Memos. In ICDE, 2006, 22.

[16] M. L. Yiu, Y. Tao, N. Mamoulis. The $B^{dual-Tree}$: Indexing Moving Objects by Space-Filling Curves in the Dual Space. To appear in Very Large Data Base Journal, 2006.

A Spatio-Temporal Database Model on Transportation Surveillance Videos

Xin Chen and Chengcui Zhang
Dept. of Computer and Information Sciences, University of Alabama at Birmingham,
Birmingham AL 35294, USA
{chenxin, zhang}@cis.uab.edu

Abstract

With the rapid growth of multimedia data, there is an increasing need for robust multimedia database model. Such a model should be able to index spatio-temporal data thus efficient access to data whose geometry changes over time can be provided. In this paper, a spatio-temporal multimedia database model for managing transportation surveillance video data is proposed. The objective is to build a spatio-temporal database schema for transportation surveillance videos, in which queries can be answered easily and efficiently. The proposed spatio-temporal model for transportation surveillance videos combines the strength of two general-purpose spatio-temporal multimedia database models - the Multimedia Augmented Transition Network model (MATN) and the Common Appearance Interval (CAI) model. While MATN model is good at modeling the replay of the multimedia presentation and the spatial-temporal relations of semantic objects in the video, it is not efficient in modeling or querying the trajectories of moving objects. In the mean time, while Common Appearance Interval (CAI) model can be used to better answer trajectory-based queries, it explicitly stores the spatial relations of pairs of objects in the model, which is considered redundant in transportation video databases. The proposed model bases its structure on MATNs and adopts the concept of CAI to segment transportation surveillance videos. Since this model is motivated by transportation surveillance applications, it has some domain specific features. It models each traffic light phase in a MATN-like network and models the corresponding video segment using CAIs. In addition, CAIs are further divided into sub-intervals and modeled by the sub-network structure in MATNs. In this paper, the proposed model, together with a brief introduction of the vehicle extraction/tracking/classification, is presented with its formal definition and some sample queries. The advantages of our model in comparison with other models are also demonstrated.

Proceedings of the 3rd Workshop on STDBM
Seoul, Korea, September 11, 2006

1. Introduction

While spatio-temporal applications (Intelligent Transportation Systems, health, climate changes, etc.) have only recently attracted researchers in this field, most of the existing work has concentrated on general-purpose spatio-temporal database models and query languages[2][3][5][6]. The existing models all have their advantages in modeling certain aspects of the data. However, none of them are "jack-of-all-trades" without any redundancy or lost of any efficiency. Since different spatio-temporal applications may have different emphasis on the properties and queries of the domain-specific data, there is a need for designing domain-specific spatio-temporal database model.

In building an intelligent transportation system, a large amount of transportation surveillance videos are collected via surveillance cameras. Various algorithms are proposed to analyze these video data. However, it is still a challenge to store and manage these videos with an efficient indexing and querying schema. In addition, it is necessary to build an integrated system that captures a comprehensive set of requirements which are needed in building a transportation surveillance video database. Such requirements include the extraction of vehicle objects from surveillance videos, vehicle tracking, vehicle classification, and spatio-temporal modeling that provides efficient data indexing and querying for transportation domain. To our best knowledge, there is no such system in this field that is sophisticated enough to manage video data efficiently in transportation surveillance domain. One of the research directions in spatio-temporal database management is to incorporate data streams and evaluate continuous queries over data streams [9] [10] [11]. For example, in [9], Mouza et al. proposes a data model for reporting continuous queries based on mobility pattern matching. Continuous queries are managed as a discrete process relying on events related to the moves of objects. Several general-purpose spatio-temporal models for video databases are proposed in [2][3][5][6]. However, since different spatio-temporal applications may have different emphasis on the properties and queries of the domain-

specific data, there is still a need for designing domain-specific spatio-temporal models. In this paper, we proposed a spatio-temporal multimedia database model for storing, indexing, and querying transportation surveillance videos. It is worth mentioning that the proposed work focuses on designing a conceptual model, which serves as a basis for defining entities, attributes, and relationships. As this is our initial work, details such as GUI interface, physical layers, access strategy or operations will be addressed separately in our future work.

In our previous work, we proposed a vehicle extraction, vehicle tracking, and vehicle classification framework [4] [8] for indexing transportation surveillance videos. With this framework, each distinct vehicle can be automatically identified at diffident locations in a video frame. The object-level information such as the bounding box and the centroid can be recorded and stored in the database for future queries. However, it would be unnecessary to record such information in each frame as it will introduce a lot of redundancy in the database. Therefore, an intuitive method is to segment a video into meaningful segments and only record the "key frames". Transportation surveillance video segmentation is not as easy a task as that for regular videos such as movies, since it is hard to detect video shots or events in the continuous transportation surveillance video sequences. In L. Chen's CAI model [2], a video segmentation concept -- Common Appearance Interval (CAI) is proposed, which has some flavor of a video shot in a movie. In this concept, each video segment is endowed with some "semantic" meaning in terms of temporality and spatial relations. This concept is adopted in the proposed model in this paper. In L. Chen's CAI model, the spatial relations of pairs of objects are recorded. This makes it convenient to query the spatial relation of two objects. However, if there are n objects appearing in the frame, there will be C_n^2 pairs of records in the database. Since the spatial relations of vehicles are not the frequent query targets in the transportation video database, this will introduce a lot of redundancy. In this paper, we follow the basic idea of S.-C. Chen's MATN (Multimedia Augmented Transition Network) model [3] in solving this problem.

MATN model is good at modeling the replay of multimedia presentations. It also provides an efficient mechanism in modeling the spatial relations of semantic objects in the video. One disadvantage of MATN is that it cannot be efficiently applied to modeling the trajectories of moving objects. However, trajectories of vehicles are often queried in a transportation video database. Therefore, in our model, MATN is adjusted to suit our specific needs.

The MATN model proposed by S.-C. Chen [3] and L. Chen's CAI model [2] are two general-purpose models. Our proposed model combines the strength of these two models. We base our structure on MATN and adopt the concept of CAI to segment traffic videos. Motivated by our specific application, our model has some features neither of the two general-purpose models has. The proposed model models each traffic phase in a MATN-like main network and models the corresponding video segment using CAIs. While the main network models the spatio-temporal relations of vehicle objects at a coarser level, CAIs can capture more details of such relations. In addition, since MATN uses the concept of Multimedia Input Strings (MISs) as the input of an MATN model, we further extend its definition to model CAIs in MATN's main network. CAIs are further divided into sub-intervals and modeled by the sub-network structures in MATNs. The direction information of a moving vehicle rather than the spatial relation between two vehicle objects is recorded in our model. This is due to the fact that transportation video database queries are more often concerned with a moving vehicle's driving direction than its spatial relation with another object. We argue that in this type of applications, there is no need to store a huge amount of redundant information that is not often queried.

Some background information on transportation surveillance video processing is introduced in Section 2. The proposed model is formally defined in Section 3. Section 4 shows query methods and some sample queries. Section 5 analyzes the advantages of the proposed model. Section 6 concludes the paper.

2. Transportation surveillance video processing

In this section, the processing of transportation surveillance videos is introduced to provide some background information on building an intelligent transportation system. In our previous work [4], an unsupervised segmentation method called the Simultaneous Partition and Class Parameter Estimation (SPCPE) algorithm, coupled with a background learning and subtraction method, is used to identify the vehicle objects in a traffic video sequence [8]. The technique of background learning and subtraction is used to enhance the basic SPCPE algorithm in order to better identify vehicle objects in traffic surveillance videos. Figure 1 shows an example of the segmentation result of a vehicle with background extracted. The rectangular area is the Minimal Bounding Rectangle (MBR) of the vehicle that is represented by (x_{low}, y_{low}) and (x_{high}, y_{high}) -- the coordinates of the bottom right point and the upper left point of the MBR. $(x_{centroid}, y_{centroid})$ are the coordinates of that vehicle segment's centroid. It is used for tracking the positions of vehicles the across video frames.

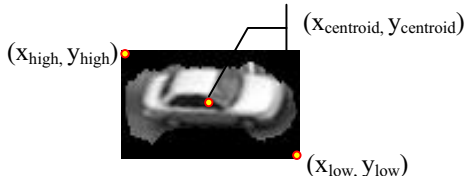


Figure 1 An example vehicle segment.

The framework in [4] has the ability to track moving vehicle objects (segments) within successive video frames. By distinguishing the static objects from mobile objects in the frame, tracking information can be used to determine the trails of vehicle objects. The last phase of the framework is to classify vehicle objects into different classes such as SUVs, pick-up trucks, and cars, etc. The classification algorithm is based on Principal Component Analysis.

With this framework, lots of useful data and information is generated. This provides a basis for an intelligent transportation system. The whole process is automatic. In this paper, a spatio-temporal database model is proposed to further organize, index and query these information.

3. Model definition

While the proposed model is a combination of S.-C. Chen et al.'s MATN model [3] and L. Chen's CAI model [2], its general structure is based on MATN. Therefore, in this section, we will start from introducing the basic idea of MATN model. Following that, the formal definition of our model is presented.

3.1 MATN model

Multimedia Augmented Transition Network (MATN) model originates from the Augmented Transition Network (ATN) [1] which is used for natural language processing. The inputs of an ATN are sentences that are composed of words. Similarly, in MATN, the inputs are Multimedia Input Strings that can be denoted by regular expressions. An MATN model simulates a finite state automaton in that it is constructed by nodes (states), arcs, inputs and transition functions. However, unlike finite state automaton, MATN allows recursions and has the condition/action tables. Recursions allow the user to play the desired video segments repeatedly. Condition/action tables are especially useful in an online environment. With the limitation of bandwidth, the user can choose to play some parts of the video when condition permits or get compressed version of the video. That is, an MATN can control the synchronization and Quality of Service (QoS) of multimedia streams. Another important feature of an MATN is its support for sub-networks. These features of MATN make it more powerful and effective than finite state automaton. It is worth mentioning that MATN not only can be used to model the spatio-temporal relations of multimedia streams in multimedia presentations but also can support multimedia database searching when spatio-temporal relations of multimedia objects are concerned. In this paper, we focus on the spatio-temporal modeling and multimedia database searching capability of MATN models.

3.2 The proposed model

MATN is a general purpose spatio-temporal model. To fit the specific needs of transportation video modeling, the original MATN model needs to be adjusted. This is for the ease of searching and querying the transportation video database. For example, velocity and driving direction are two very important properties of a moving vehicle but they are not explicitly modeled in a MATN model. In L. Chen's CAI model, a concept called Common Appearance Interval is defined to model an interval where a certain set of objects appear in the frame together. We incorporate this concept into our model. CAIs can be automatically generated from the tracking and segmentation phase.In the proposed model, moving vehicles are explicitly modeled, which correspond to the moving objects in CAI model. A Common Appearance Interval is further broken down into sub-intervals in which the relative positions (as defined in MATN models) of vehicles remain unchanged.

In MATN models, the spatial relations of moving vehicles are recorded based on 27 three dimensional relative positions. That is, the 3-D space is evenly divided into 27 positions with one of them being the reference position. The coordinates of moving vehicles are then compared with this reference position to decide their relative positions. The details of these 27 positions are shown in Figure 2, where position #1 is the reference position. The first and the third columns indicate the relative position numbers, while the second and the fourth columns are the relative coordinates. (x_t, y_t, z_t) and (x_s, y_s, z_s) represent the X-, Y-, and Z-coordinates of the reference position and the position of a vehicle object, respectively. The '≈' symbol means the difference between two coordinates is within a threshold value. For example, the relative position number 25 indicates an object's X- and Y-coordinates (x_s and y_s) are greater than that (x_t, y_t) of the reference position, while their Z-coordinates are approximately the same. We adopt this approach in modeling the spatial relations of moving vehicle objects. In the proposed model, the center of a video frame is chosen to be the reference position and each vehicle object is mapped to a point object represented by its centroid as illustrated in Figure 1. The centroid point of each vehicle object is then used to derive the relative position of that object to the reference position. However, it is worth mentioning that we only use the 2-D relative positions in MATN models as the z values (or the depth information) of all vehicle objects in a 2-D video sequence are zeros. Therefore, there are only 9 relative positions in our model, which are used to record the relative positions of vehicles in a video frame at a coarse granularity. More or fewer numbers may be used to divide an image or a video frame into sub-regions to allow more fuzzy or more precise queries as necessary.

For this specific application, the driving direction of a vehicle is also recorded. However, there is no need to record this information for all video frames as this will introduce a large amount of inter-frame redundancy. Only the changes of directions between intervals are computed and recorded in the proposed model.

Before the formal definition of the proposed model is presented, we first define the terms and concepts that will be used in the rest of the paper.

Definition 1. A *Vechile Object* (*VO*) is a 3-tuple *(OID, MBR, VSFs)*. *ID* is the unique identifier of a distinct vehicle in the database. *MBR* is the *Minimal Bounding Rectangle* of the Vehicle Object. (x_{low}, y_{lo}, z_{low}) and (x_{high}, y_{high}, z_{high}) are the 3-D coordinates of the bottom right point and the upper left point of the rectangle. The coordinates of the centroid of the *MBR* is used to determine the object's relative position in 27 positions. However, since only 2-D information is available, we will use the 9 positions as shown in Figure 2 only. *VSFs* stands for *Vehicle Segment Features*, which can be used for vehicle classification and tracking purpose. In our previous work [4], a set of vehicle features based on Principle Component Analysis was proposed actually used in our vehicle classification framework to decide the vehicle type (cars, pick-up trucks, SUVs, etc.)

Definition 2. *Traffic Light Phase (TLP)* is a segment of the traffic video during which there is no traffic light change. It is denoted by a 4-tuple *(PID*, *PSF*, *PEF*, *META). PID* is the ID of the video segment. *PSF* is the starting frame of the traffic light phase; *PEF* is the ending frame of the phase; *META* is the meta-data of this phase. It includes the allowed driving directions, the duration of the phase, etc.

Definition 3. *Traffic Video Clip* (*TVC*) is a contiguous trunk of traffic video. It consists of *TLPs* and can be denoted as a 2-tuple (*CID*, *META*). *CID* is its clip ID. *META* is the meta-data of this clip which can include such information as the time the clip is shot, the road/intersection location, camera settings, etc.

TVCs, *TLPs* and *VOs* are the constituting units of a transportation surveillance video database with *VO* being the smallest unit. Next we will give the formal definition of the proposed model and explain in detail how it can be used for transportation video database modeling.

Number	Relative Coordinates	Number	Relative Coordinates	Number	Relative Coordinates
1	$x_s \approx x_t$, $y_s \approx y_t$, $z_s \approx z_t$	10	$x_s < x_t$, $y_s \approx y_t$, $z_s \approx z_t$	19	$x_s > x_t$, $y_s \approx y_t$, $z_s \approx z_t$
2	$x_s \approx x_t$, $y_s \approx y_t$, $z_s < z_t$	11	$x_s < x_t$, $y_s \approx y_t$, $z_s < z_t$	20	$x_s > x_t$, $y_s \approx y_t$, $z_s < z_t$
3	$x_s \approx x_t$, $y_s \approx y_t$, $z_s > z_t$	2	$x_s < x_t$, $y_s \approx y_t$, $z_s > z_t$	21	$x_s > x_t$, $y_s \approx y_t$, $z_s > z_t$
4	$x_s \approx x_t$, $y_s < y_t$, $z_s \approx z_t$	13	$x_s < x_t$, $y_s < y_t$, $z_s \approx z_t$	22	$x_s > x_t$, $y_s < y_t$, $z_s \approx z_t$
5	$x_s \approx x_t$, $y_s < y_t$, $z_s < z_t$	14	$x_s < x_t$, $y_s < y_t$, $z_s < z_t$	23	$x_s > x_t$, $y_s < y_t$, $z_s < z_t$
6	$x_s \approx x_t$, $y_s < y_t$, $z_s > z_t$	15	$x_s < x_t$, $y_s < y_t$, $z_s > z_t$	24	$x_s > x_t$, $y_s < y_t$, $z_s > z_t$
7	$x_s \approx x_t$, $y_s > y_t$, $z_s \approx z_t$	16	$x_s < x_t$, $y_s > y_t$, $z_s \approx z_t$	25	$x_s > x_t$, $y_s > y_t$, $z_s \approx z_t$
8	$x_s \approx x_t$, $y_s > y_t$, $z_s < z_t$	17	$x_s < x_t$, $y_s > y_t$, $z_s < z_t$	26	$x_s > x_t$, $y_s > y_t$, $z_s < z_t$
9	$x_s \approx x_t$, $y_s > y_t$, $z_s > z_t$	18	$x_s < x_t$, $y_s > y_t$, $z_s > z_t$	27	$x_s > x_t$, $y_s > y_t$, $z_s > z_t$

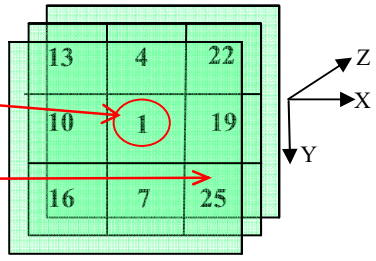


Figure 2 Three dimensional relative positions for vehicle objects.

3.3 Formal Definition of the proposed model

For convenience, we call the proposed model TVDM (Transportation Video Database Model). TVDM can be formally defined as follows.

Definition 4. A TVDM is an 8-tuple (Σ_m, Σ_s, Ω, δ, Q_m, Q_s, F, S). Σ_m,Σ_s are the alphabets of the traffic video streams that can be expressed by regular expressions. The meaning of the regular expression symbols used in this model is illustrated in Table 1.

Table 1. Meaning of Regular Expression Symbols

Symbol	Meaning
Unquoted Characters	Non-terminal symbols
'…'	Terminal symbols
=	Is defined as
[…]	Optional symbols
{…}+	One or more repetitions
{…}	Zero or more repetitions
&	Concurrent
…|…	Or
;	Rule terminator
,	Concatenation
[c_1 – c_2]	ASCII Characters

1. Σ_m is the Transportation Video Stream alphabet in the main network. A Transportation Video Stream in the main network is a *Common Appearance Interval (CAI)* as proposed in L. Chen's CAI model [2]. A *CAI* is an interval in which vehicle objects VO_1, VO_2, …, VO_m appear all together. A new *CAI* starts when there is a new vehicle appears in the video or an old one disappears in the video or both. Therefore, a *CAI* is a media stream that can be expressed in regular expressions,

CAI = CID, PID, CAID, {OID, ['&']};
CID = 'C', {'0'-'9'};
PID = 'P', {'0'-'9'};
CAID = 'CA', {'0'-'9'};
OID = 'O', {'0'-'9'};

where '&' means concurrent appearance of multiple vehicle objects, CID, PID, CAID, and OID stand for the ID's of *Clip*, *Phase*, *CAI* and *Object(s)*.

2. Σ_s is the Transportation Video Stream alphabet in the sub-network. A Transportation Video Stream in the sub-network corresponds to a sub-interval of a *CAI*. In the proposed model, sub-network models a *CAI* in a way that a *CAI* is further divided into sub-intervals. In each sub-interval, the relative positions of all objects in the *CAI* remain unchanged. We call such an interval CAI_{sub}. Therefore, we have

CAI_{sub} = CID, PID, CAID, CAISUBID, {OID, ('1' | '2' | …| '27'), ('N' | 'NW' | 'NE' | 'S' | 'SW' | 'SE' | 'E' | 'W'), ['&']};
CAISUBID = 'CAS', {'0'-'9'};

From the above regular expression, we can see that in each CAI_{sub}, a vehicle object is denoted by an object ID followed by a number and a symbol. The number is one

of the 27 3-D relative positions as illustrated in Figure 1. ('N' | 'NW' | 'NE' | 'S' | 'SW' | 'SE' | 'E' | 'W') denotes the moving direction of that vehicle object, where N stands for north, NW stands for northwest and so forth. The driving direction can be induced from the change of relative positions between sub-intervals. This direction information is kept here for easily capturing the trajectory of a vehicle which is frequently queried in transportation video database.

3. Ω is the special input symbol alphabet. Ω = {&&, ||, ~, *, α, β}. The meaning of these special symbols is shown in Table 2. These symbols are used in querying the transportation video database.

Table 2. Special Input Symbols

Symbol	Meaning
&&	Logical And
||	Logical Or
~	Logical Not
*	Wildcard
α	Arithmetic operators such as '+', '-'…
β	Condition operators such as '<', '>', '= =', '!='…

4. Q_m is the set of nodes (states) in the main network. Each node in Q_m is defined as a 4-tuple *(NID, FID, OID_{in}, OID_{out})*. *NID* is the node ID. *FID* is the frame ID. This frame is the starting frame of the next *CAI* that is on the outgoing arc of this node (state). OID_{in} is the list of IDs of the vehicle objects that newly appear in the next *CAI*. OID_{out} is the list of IDs of the vehicle objects that disappear in the next *CAI*. However, these two lists cannot be both empty. If both of them are empty, there is no new *CAI* generated.

5. Q_s corresponds to a sub-network. It is defined as a 2-tuple *(NID_{sub}, FID)*. NID_{sub} is the ID of a node in the subnetwork. Each node is associated with a *FID* which is a frame ID. This frame is the starting frame of the next CAI_{sub} that is on the outgoing arc of this node (state).

6. δ is a set of the transition functions from one node (state) to another. In the proposed model, two nodes are connected by an arc that is denoted by a transportation video stream. $\delta : Q_m \times CAI \rightarrow Q_m$ or $Q_s \times CAI_{sub} \rightarrow Q_s$.

7. F is the set of final states, where $F \subset Q_m \bigcup Q_s$.

8. S is the set of starting states, where $S \subset Q_m \bigcup Q_s$.

3.4 Modeling spatio-temporal relations in transportation video database with TVDM

In this section, the above-defined TVDM is applied to model a transportation video database. Details will be explained in an example shown in a TVDM diagram (Figure 3).

In the diagram, circles are nodes/states of the network. For simplicity, only the node ID is shown in the figure. However, the identification of a node is also dependent on the traffic light phase and the video segment it is in. The network flow can be easily traced by following the arcs (arrows) in the diagram. The solid arcs are the flow in the main network while the dotted ones are in the sub-network. Each arc can be distinguished by the unique traffic media stream on it. Again for simplicity, the full name of each vehicle object in the video streams (CAI/CAI_{sub}) is not given in the figure. Instead, characters such as 'A' and 'B' are used as the symbols to represent the vehicle objects.

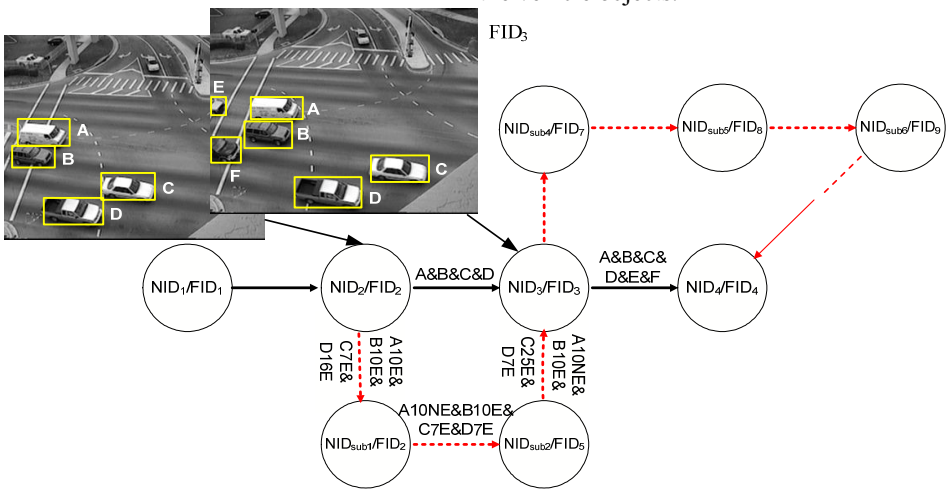


Figure 3. TVDM diagram

In this example, a traffic video clip shot at a major intersection is modeled. A video frame is divided into nine 2-D sub-regions. The entire traffic video clip is divided into phases according to the change of traffic lights. Each phase can be modeled with a network as shown in Figure 3. By connecting each such network with an arc, the network of the entire traffic video clip can be set up. There are two key frames shown in this figure. FID_2, FID_3 are the IDs of the two frames which are the turning points of two consecutive *CAIs*. Starting from the node NID_3, two new vehicle objects (E and F) move into the scene, signifying the end of the previous *CAI* and the beginning of the next *CAI*. The video stream between

NID_2 and NID_3 is denoted as 'A&B&C&D', meaning the concurrent appearance of vehicle objects A, B, C and D.

In the sub-network originating from NID_2 and ending at NID_3, the corresponding traffic video stream (A&B&C&D) is further modeled by the vehicle objects' relative positions and driving directions. Each such sub-stream is called a CAI_{sub} in which the relative positions of all vehicle objects remain unchanged. For example, the first arc of this subnetwork carries the sub-stream (A10E&B10E&C7E&D16E) in which vehicle A is in position 10 heading east; vehicle B is in position 10 heading east; vehicle C is in position 7 heading east; vehicle D is in position 16 also heading east. The arc originating from NID_{sub2} is the sub-stream in which vehicle A is still in position 10 but heading northeast; vehicle B is still in position 10 heading east; vehicle C has moved to position 25 and is still heading east; vehicle D has moved to position 7 heading east. In this traffic light phase, we can tell that vehicles are either driving west/east or northeast i.e. left turn. By storing this information in the meta-data of each traffic light phase, the vehicles driving toward illegal directions can be easily detected.

In the following sections, we will discuss the transportation video database queries with the proposed model and explore some typical scenarios that may be of users' interest in querying the transportation video database.

4. Transportatino video database queries

4.1 Multimedia Input String

As introduced in Section 3.1, the inputs of MATN models are Multimedia Input Strings (MIS) together with condition/action tables. An MIS is to simulate the input of an ATN, which is a natural language sentence. In MATN, there are two types of basic inputs. One is "Control_Command". It represents a control message that may occur during the multimedia presentation. A "Control_Command" is composed of conditions and actions. For example, a "Control_Command" can be "if the bandwidth < θ, display the compressed version of the specified media stream". "Bandwidth < θ" is the condition and "display the compressed version of the specified media stream" is the action. However, since multimedia presentation is not the focus of this paper, we concentrate on the other type of multimedia input strings which is "Media_Stream". In this paper, we designed our own input strings by following the basic idea of MATN's "Media_Stream". It has been used in Section 3 to define the traffic video stream in a CAI or CAI_{sub}.

Query strings are constructed based on Media_Streams. Query strings are used as the query input of the proposed model for querying the transportation video database. They include some special input symbols which have special meanings. The list of such symbols is in Table 2. For example, the symbol '||' means one of the conditions on the two sides has to be satisfied. "CAI.(A&B)||CAI.(A&C)" is the query string used to find CAIs containing "A&B" or "A&C". "CAI_{sub}.(A*N&B*S)" means moving vehicles A and B are in the same CAI_{sub} with A driving northward and B driving southward. "*" is the wild card symbol meaning that A and B can be in any relative positions. It is obvious that the first query string can easily find its matches (or no match) in the main network. The answer to the second query string can only be found in sub-networks of the model. However, the search shall start from the main network to find all video streams containing "A&B" and delve into the sub-network thereafter.

4.2 Object oriented transportation video database management and sample queries

In [7], a spatio-temporal model is proposed and integrated into ODBMS. Since our model also targets on modeling traffic video objects, we argue that ODBMS is suitable for transportation video databases.

In the proposed model, there are 6 classes of objects. The class names and properties are illustrated in Table 3, which is a generalization of the definitions introduced in Section 3. The first three classes are defined in Definitions 1, 2, and 3. The NODE class is defined in the formal definition of TVDM. For CAI/CAI_{sub} class, there are four important properties i.e. the starting node (NID_s) and the ending node (NID_e) in the main network, the starting nodes (NID/NID_{sub}) in the main/sub network, and the traffic video streams in main/sub network as defined in Section 3.3. The FRAME class contains the actual video frames in the definition of nodes of the network. Its properties include FID (frame ID), the Video Clip it is in i.e. CID and the actual frame in the form of an image file. A VO object corresponds to a distinct vehicle object. For example, if there is a vehicle object A, A.OID is its ID. Note that the MBR property in VO class is actually a list $<mbr_{f1}, mbr_{f2}, ..., mbr_{fn}>$ denoting the MBR property of a VO across the frames $f_1, f_2, \ldots, f_n$.

Table 3. Objects and Properties

Class	Properties
VO	OID， MBR， VSF
TLP	PID, PSF, PEF, META
TVC	CID, META
NODE	NID, OID_{in}, OID_{out}, FID
CAI/CAI_{sub}	NID_s, NID_e, NID/NID_{sub}, Media_Stream
FRAME	FID, CID, Frame

4.3 Sample queries

In order to test the effectiveness of the proposed spatio-temporal model, some typical transportation video database queries are studied in this section. In transportation video database, the user is often more interested in retrieving video data through queries on vehicle objects' properties and spatio-temporal relations among them.

Query Example 1. Given one vehicle, find all the vehicles north to it.

This is a simple query in which only the information of spatial relations between vehicles is needed. Suppose this

vehicle A is in position 1 and the target vehicle is B. The user can issue this query using a high-level language which will then be translated into a Media_Stream together with some arithmetic and logic operations:

$$CAI_{sub}.(A1*\&B4*)\|$$
$$(CAI_{sub}.(A1*\&B1*)\&\&$$
$$(B.mbr_{(CAIsub.NIDsub.FID)}.y < A.mbr_{(CAIsub.NIDsub.FID)}.y))$$

The meaning of the above expression is that in some CAI_{sub}, vehicle object A is in position 1 and B is in position 4 OR both A and B are in position 1 with the y-coordinate of B's centroid less than that of A's. It does not matter in what directions the two vehicles are moving. In either of these two situations, B is considered north to A. In the CAI model [2], the spatial positions between each pair of vehicle objects are recorded explicitly in the database. Therefore, for Query Example 1, there would be no need to compute the relation between B.mbr.y and A.mbr.y. Our model did not adopt this approach. In both the CAI model and our model, each vehicle object's coordinates (bounding box and centroid) are already stored in MBR. Therefore, it is easy to compute the relative spatial relation between two vehicle objects. However, our model chooses to store the spatial relation between pairs of vehicle objects at a coarser granularity. That is, for each vehicle object, only its relative position to the reference position is stored. If two vehicle objects happen to fall into the same position relative to the reference position, their spatial relation is examined on the run to avoid storing too much information in the database. This can reduce the redundancy in transportation surveillance video database, since the queries on spatial relations of vehicles are not as often as on vehicles' moving trajectories.

Query Example 2. Two vehicles are meeting each other with one from northwest and another from south east.

$$CAI_{sub}.(A*NW\&B*SE)\ \&\&$$
$$(Dist(A.mbr_{(CAIsub.NIDsub.FID)}.high, B.mbr_{(CAIsub.NIDsub.FID)}.low) <$$
$$Dist(A.mbr_{(CAIsub.NIDsub.FID)}.low, B.mbr_{(CAIsub.NIDsub.FID)}.high))$$

Dist(C_1, C_2) is the subroutine to calculate the distance between two points whose coordinates are C_1 and C_2. A.mbr.high is the coordinates of the upper left corner of A's MBR and A.mbr.low is the coordinates of the bottom right corner. The meanings of B.mbr.low, A.mbr.low and B.mbr.high are similar.

Query Example 3. Find vehicles that are speeding.

$$(A.OID \in Node1.OID_{in})\&\&(Node2.OID_{out} \in A.OID)\ \&\&$$
$$(dist(A.mbr_{(Node1.FID)}.centroid, A.mbr_{(Node2.FID)}.centroid) /$$
$$((Node2.FID - Node1.FID) / TVC.META.FR) > \theta)$$

This query can be conducted on the main network. Node1 and Node2 are two nodes in the main network where one of the incoming vehicles in Node1 is an outgoing vehicle in Node2. From the frame IDs of these two nodes, the number of frames in between can be calculated, which is divided by the frame rate (TVC.META.FR). In this way, the average velocity of the vehicle when passing the intersection can be calculated. If a vehicle's velocity is larger than θ, the allowed maximum speed, this vehicle is considered speeding. "dist($A.mbr_{(Node1.FID)}$.centroid, $A.mbr_{(Node2.FID)}$.centroid)" is the subroutine that computes the travel distance of the vehicle in passing this intersection. "FR" in TVC.META.FR" means the frame rate (frs/sec) which is stored in the meta-data of that video clip.

Query Example 4. Find vehicles that take a U-turn.

$$((CAI_{sub1}.(A*S)\&\&CAI_{sub2}.(A*N))\|$$
$$(CAI_{sub1}.(A*N)\&\&CAI_{sub2}.(A*S))\|$$
$$(CAI_{sub1}.(A*E)\&\&CAI_{sub2}.(A*W))\|$$
$$(CAI_{sub1}.(A*W)\&\&\ CAI_{sub2}.(A*E))\|$$
$$(CAI_{sub1}.(A*NW)\&\&\ CAI_{sub2}.(A*SE))\|$$
$$(CAI_{sub1}.(A*SE)\&\&\ CAI_{sub2}.(A*NW))\|$$
$$(CAI_{sub1}.(A*NE)\&\&\ CAI_{sub2}.(A*SW))\|$$
$$(CAI_{sub1}.(A*NE)\&\&\ CAI_{sub2}.(A*SW)))\ \&\&$$
$$((CAI_{sub2.NIDsub.FID} - CAI_{sub1.NIDsub.FID})/\ TVC.META.FR < \delta)$$

In this query, the sub-networks are searched for any vehicle that drives toward opposite directions within reasonable time duration δ when passing the intersection.

Query Example 5. Find vehicles that drive toward illegal direction in the traffic light phase when only north-bound and south-bound vehicles are allowed.

$$\sim(CAI_{sub}.(A*S)\|CAI_{sub}.(A*N))$$

The south-bound and the north-bound allowed directions are information extracted from the meta-data of that traffic light phase. According to this information, we can detect vehicles driving at wrong directions by using similar queries as the above.

Query Example 6. Find vehicles that stop at some time.

$$CAI.(A*)\&\&$$
$$(dist(A.mbr_{(CAI.NIDs.FID)}.centroid, A.mbr_{(CAI.NIDe.FID)}.centroid)$$
$$/$$
$$((CAI.NID_s.FID - CAI.NID_e.FID) / TVC.META.FR) == 0)$$

A vehicle's velocity when passing the intersection cannot be zero. Otherwise, it is considered to have stopped. This can be used in identifying accidents or traffic jams. If one or more vehicles stay still for a long consecutive sequence of CAIs, there might be an accident or jam occurring in the intersection.

Query Example 7. Given a vehicle A that is driving eastward, find a vehicle B that overtakes A.

$$(CAI_{sub1}.(A*E\&B*E)\&\&$$
$$(B.mbr_{(CAIsub1.NIDsub.FID)}.x < A.mbr_{(CAIsub1.NIDsub.FID)}.x)\&\&$$
$$(B.mbr_{(CAIsub1.NIDsub.FID)}.y \approx A.mbr_{(CAIsub1.NIDsub.FID)}.y))\&\&$$
$$(CAI_{sub2}.(A*E\&B*E)\&\&$$
$$(B.mbr_{(CAIsub2.NIDsub.FID)}.x > A.mbr_{(CAIsub2.NIDsub.FID)}.x)\&\&$$
$$(B.mbr_{(CAIsub2.NIDsub.FID)}.y \approx A.mbr_{(CAIsub2.NIDsub.FID)}.y))$$

The condition of an overtaking is that both vehicles are driving in the same direction with one behind another. The one behind overtakes the other by using an empty lane next to the lane both vehicles are on. Therefore, initially the x-coordinate of A (the leading vehicle) is larger than that of B's. After overtaking, A's x-coordinate

is smaller than that of B's. As the whole process may not finish within one CAI_{sub}, CAI_{sub1}, and CAI_{sub2} do not have to be two consecutive intervals (the same applies to all the previous queries.) However, we do need to find out the closest pair of CAI_{sub2} and CAI_{sub1}, since any CAI_{sub} after/before CAI_{sub2}/CAI_{sub1} with A and B in it may also satisfy the above conditions.

5. Advantages

The goal of this paper is not to design a master-of-all spatiotemporal database model, which is often not feasible. Instead, the proposed model in this paper is domain-specific. That is, it focuses on modeling the transportation surveillance video database. Targeting at the specific characteristics of transportation video, the proposed model can extract, index, and store the key information in the video. With these information stored in the database, transportation video data can be efficiently accessed and queried. This is one of the major advantages of our model.

The proposed model combines the strength of two general purpose spatio-temporal database models – MATN and CAI. We follow MATN's basic structure as well as its way of modeling spatial relations among objects. The key concept in L. Chen's model – CAI is also adopted in our model. This provides a way in partitioning transportation videos into "meaningful" segments and extracting the spatio-temporal information out of that. By combining the advantages of the two general-purpose models, our proposed model can better meet the needs of a transportation surveillance video database.

More specifically, motivated by this specific application, the proposed model only stores information that is frequently queried. Therefore, unlike CAI model, the proposed model does not record spatial relations between each pair of moving objects as this is not the frequent query type in the transportation video database. Furthermore, this will introduce redundancy into the database. The proposed model only records the relative spatial-relation of moving objects at a coarse granularity based on MATN model. The direction information of a moving vehicle is also recorded since this is a big concern of the user's queries. For example, the illegal driving direction and U-turn can be easily detected in our model, while it is not that easy in either the MATN or the CAI model. In order to further extract useful information from the video, CAIs are further divided into sub-intervals in which all moving vehicles' relative positions remain unchanged. We call this sub-interval CAI_{sub}. This sub-division enables us to model the video streams at a finer granularity instead of simply using CAIs.

6. Conclusion

This paper proposed a spatio-temporal multimedia database model for transportation surveillance video. The formal definition of the model provides the basis for constructing a transportation surveillance video database, which is a key in building an Intelligent Transportation System. The proposed model combines the strength of two general-purpose spatio-temporal models –MATN and CAI. The proposed model is also adjusted to suit the specific needs of this specific application. To avoid redundancy, only the frequently queried information is stored in the database. Thus the extraction of key information from the transportation video is performed in a way to facilitate queries in this specific domain. With this model design, data indexing and database queries can be performed more efficiently.

8. References

[1] Woods, W. Transition network grammars for natural language analysis. *Comm. ACM,* vol. 13, pp.591-602, Oct. 1970.

[2] Chen, L., and Özsu, M. T. Modeling of video objects in a video database. In *Proc. of IEEE International Conference on Multimedia,* Lausanne, Switzerland, August 2002, pp.217-221.

[3] Chen, S.-C., and Kashyap, R. L. A spatio-temporal semantic model for multimedia database systems and multimedia information systems. *IEEE Transactions on Knowledge and Data Engineering,* vol. 13, no. 4, pp. 607-622, July/August 2001.

[4] Zhang, C., Chen, X., and Chen, W.-B. A PCA-based vehicle classification framework. *IEEE International Workshop on Multimedia Databases and Data Management,* in conjunction with *IEEE International Conference on Data Engineering (ICDE 2006),* April 8, 2006, Atlanta, Georgia, USA.

[5] Day, Y.F., Dagtas, S., Iino, M., Khokhar, A., and Ghafoor, A. Object-oriented conceptual modeling of video data. In *Proc. of Eleventh International Conference on Data Engineering (ICDE),* March 1995, pp. 401-408.

[6] Wasfi, A.K., and Arif, G. An approach for video meta-data modeling and querying processing. In *Proc. of ACM Multimedia,* 1999, pp. 215-224.

[7] Li, J.Z., Özsu, M. T., and Szafron, D. Modeling of moving objects in a video database. In *Proc. of IEEE International Conference on Multimedia Computing and Systems,* pp. 336-343, 1997.

[8] Chen, S.-C., Shyu, M.-L. , Peeta, S., and Zhang, C. Learning-Based spatio-temporal vehicle tracking and indexing for transportation multimedia database systems", *IEEE Transactions on Intelligent Transportation Systems*, vol. 4, no. 3, pp. 154-167, September 2003.

[9] Mouza, C., and Rigaux P. Mobility patterns. In *Proc. of 2nd Workshop on Spatio-temporal Database Management (STDBM'04),* Toronto, Canada, August 30th, 2004.

[10] Babu, S., and Widom, J. Continuous queries over data streams. In *Proc. of SIGMOD Record,* September 2001.

[11] Mokbel, M. F., Xiong, X., Hammad, M. A., and Aref, W. G. Continuous query processing of spatio-temporal data streams in PLACE. In *Proc. of Second Workshop on Spatio-temporal Database Management (STDBM'04),* Toronto, Canada, August 30th, 2004.

Predicted Range Aggregate Processing in Spatio-temporal Databases

Wei Liao, Guifen Tang, Ning Jing, Zhinong Zhong

School of Electronic Science and Engineering, National University of Defense Technology
Changsha, China
liaoweinudt@yahoo.com.cn

Abstract

Predicted range aggregate (PRA) query is an important researching issue in spatio-temporal databases. Recent studies have developed two major classes of PRA query methods: (1) accurate approaches, which search the common moving objects indexes to obtain an accurate result; and (2) estimate methods, which utilize approximate techniques to estimate the result with an acceptable error.

In this paper, we present a novel accurate prediction index technique, named PRA-tree, for range aggregation of moving objects. PRA-tree takes into account both the velocity and space distribution of moving objects. First, the velocity domain is partitioned into different velocity buckets, and moving objects are classified into different velocity buckets by their velocities, thus objects in one bucket have similar velocities. Then we use aTPR-tree, which is based on the basic TPR-tree structure and added with aggregate information in intermediate nodes, to index objects in each bucket. PRA-tree is supplemented by a hash index on IDs of moving objects, and exploits bottom-up deletion algorithm, thus having a good dynamic performance and concurrency. Also new PRA query methods with a more precise branch-and-bound searching strategy are developed for PRA-tree. Extensive experiments confirm that the proposed methods are efficient and practical.

1. Introduction

Traditional research in spatio-temporal databases often aims at predicted query, which retrieves the moving objects lying inside a multidimensional hyper-rectangle in future. In many scenarios (e.g., statistical analysis, traffic monitoring, etc.), however, users are interested only in summarized information about such objects, instead of their individual properties. Consider, for example, a spatio-temporal database managing the vehicles in a city, results of predicted queries (e.g., finding all vehicles in the center) are meaningless (due to continuous object movements), while the aggregate information (the number of vehicles) is usually stable and measurable (e.g., finding the number of vehicles across the center during the next 5 minutes) [1], [2].

Specifically, given a set S of moving points in the d-dimensional space, a predicted range aggregate (PRA) query returns a single value that summarizes the set $R \subseteq S$ of points in a d-dimensional hyper-rectangle q_R and a future time interval (or a future timestamp) q_T according to some aggregation function (e.g., *count*, *max*, *min*, *sum*, *average*). In this paper, we consider predicted range *distinct count* queries on multidimensional moving points, where the result is the size of R (e.g., the number of vehicles in an area q_R and a future time interval (or a future timestamp) q_T), but the solutions can apply to any other aggregation mentioned above with straightforward adaptation.

1.1 Motivation

A PRA query can be trivially processed as an ordinary query, i.e., by first retrieving the qualifying objects and then aggregating their properties. This approach assumes that the server manages the detailed information of moving objects using a (typically disk-based) spatio-temporal access method [3], [4], [5], however, incurs significant overhead since, the objective is to retrieve only a single value (as opposed to every qualifying object). The most popular aggregate indexes [6], [7] aim at estimating the PRA results. These approaches are motivated by the fact that approximate aggregates with small error are often as useful as the exact ones in practice, but can be obtained much more efficiently. In particular, such estimation methods require only a fraction of the dataset or a small amount of statistics and are significantly faster than exact retrieval. In this paper, we focus on accurate PRA queries processing with a novel aggregate index.

1.2 Contributions

In this paper, we propose a new indexing method, called predicted range aggregate tree (PRA-tree), for efficient

---

Proceedings of the 3rd Workshop on STDBM
Seoul, Korea, September 11, 2006

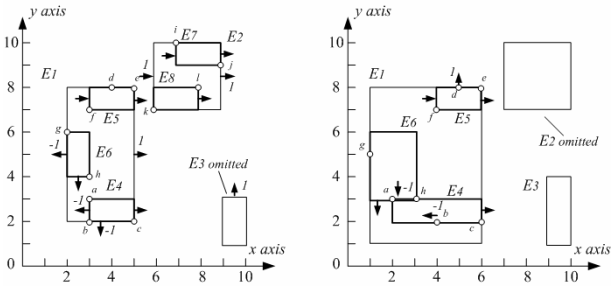
PRA queries processing. The PRA-tree takes into account the distribution of moving objects both in velocity and space domain. First, the velocity domain is partitioned into velocity buckets, and moving objects are classified into different velocity buckets by their velocities, thus objects in one bucket have similar velocities. Then we use aTPR-tree, which is based on the basic TPR-tree structure and added with aggregate information in intermediate nodes, to index the objects in each bucket. We supplement the PRA-tree by a hash index constructed on the IDs of moving objects, and develop a bottom-up deletion algorithm to obtain good dynamic performance and concurrency. Finally, we present novel algorithms that use a more precisely branch-and-bound searching strategy for PRA queries based on PRA-tree.

The rest of the paper is organized as follows: Section 2 reviews previous work related to ours. Section 3 provides the problem definition and an overview of the proposed methods. Section 4 presents the PRA-tree index structure and its construction, insertion, deletion and update techniques. Section 5 elaborates algorithms for PRA queries. Section 6 evaluates the proposed methods through extensive experimental evaluation. Finally, Section 7 concludes the paper.

2. Related Work

Section 2.1 discusses the most popular TPR-tree index, while Section 2.2 overviews the aggregate R-tree (aR-tree) which motivates our solution.

2.1 The TPR-tree



a) MBRs at time 0 b) MBRs of at time 1

Figure 1 MBRs of TPR-tree

The TPR-tree is an extension of the R-tree that can answer predicted queries on dynamic objects. The index structure is very similar to R-tree, and the difference is that the index stores velocities of elements along with their MBRs in nodes. The leaf node entry contains not only the position of moving objects but also their velocities. Similarly, an intermediate node entry also stores MBRs and velocity vector VBRs of its child nodes. As in traditional R-tree, the extents of MBR are such that tightly encloses all entries in the node at construction time. The velocity vector of the intermediate node MBR is determined as follows: (i) the velocity of the upper edge is the maximum of all velocities on this dimension in the sub-tree; (ii) the velocity of the lower edge is the minimum of all velocities on this dimension. This ensures that the MBR always encloses the underlying objects, but it is not necessarily tight all the time. The TPR-tree inherits the problems related to the R-tree, such as overlap and dead space. Since the index structure is dynamic, its query performance degrades quickly with time. The algorithms for insertion and deletion are also similar to R-tree, while the TPR-tree uses time-parameterized metrics for parameters such as the area, perimeter, and distance from the centroid [4]. For example, Figure 1a) shows the MBRs and their velocities of TPR-tree at construction time 0, and Figure 1b) shows MBRs of the same TPR-tree at future time 1.

2.2 The Aggregate R-tree (aR-tree)

The aR-tree [8] improves the conventional R-tree by adding aggregate information into intermediate nodes. Figure 2a) shows an example, where for each intermediate entry, in addition to the minimum bounding rectangle (MBR), the tree stores the number of objects in its subtree (i.e., count aggregate function). To answer a RA query *q* (the shaded rectangle), the root R is first retrieved and its entries are compared with *q*. For every entry, there are three cases: 1) The MBR of the entry (e.g., e1) does not intersect *q*, and thus its subtree is not explored further. 2) The entry partially intersects *q* (e.g., e2) and its child nodes are fetched to continue the search. 3) The entry is contained in *q* (e.g., e3), then we simply add the aggregate number of the entry (i.e., 3 for e3) without accessing its subtree. As a result, only two node accesses (R and R2) are necessary, while a conventional R-tree (i.e., without the aggregate numbers) would also visit R3. The cost savings increase with the window of the query, which is an important fact because in practice RA queries often involve large rectangles.

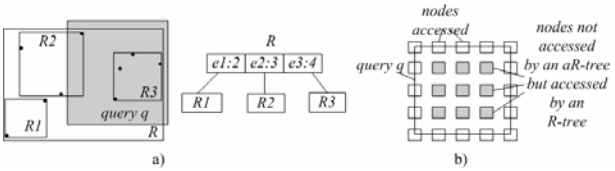


Figure 2 The aR-tree

3 Preliminary and Overview

We start with a concrete definition of aggregate query in spatio-temporal databases before presenting the PRA-tree index. By definition, a spatiotemporal object is a unified object with spatial and temporal extent [2]. A pure spatial object can be a point, a line, or a region in two or three-dimensional space. The position and/or shape either changes continuously (e.g., the motion of vehicles) or discretely (e.g., the shrink of forest) with time. In this paper we concentrate on the points (moving objects), which have continuously moving positions, in two-dimensional space. In spatio-temporal databases, a

moving object is represented as <*Loc*, *Vec*, A_1, A_2…A_n>, where *Loc*, *Vec* denote the position and velocity vector respectively, and A_1, A_2…A_n denote the non-spatial properties of this object.

An aggregate function takes a set of tuples and returns a single value that summarizes the information contained in the set of tuples. Spatio-temporal aggregate query first retrieves the qualifying objects that satisfy a spatio-temporal query predicate and returns aggregation information (e.g., count, average) on the non-spatial properties of objects.

Given a range query $q(q_R,q_T)$, which obtains the objects lying in the query window q_T and space area q_R, and a moving objects dataset *P*, a spatio-temporal query predicate *Sel(P)* can be expressed as $Sel(P)=\{p_i | \exists t \in q_T \ such\ that\ p_i(t) \in q_R\}$. Especially, if $q_T=t_f$, where t_f denotes a future timestamp, then the query *q* is a predicted timestamp range query. Generally, the query space area is a static rectangle. Therefore, we can define predicted range aggregate query as the following.

Definition 1 (predicted range aggregate query). Given a dataset *P*, in which each tuple is represented as <*Loc*, *Vec*, A_1, A_2,…,A_n>, where the domain of A_i is D_i, and a aggregate function $f : D_1 \times D_2 \times \ldots \times D_n \rightarrow D_{agg}$, where D_{agg} denote the range of *f*. then an predicted range aggregate query can be defined as follows:

$f(q,P)=f(Sel(P))=f(\{p_i | \exists t \in q_T \ such\ that\ p_i(t) \in q_R\})$.

Similar to the aR-tree, we enhance the conventional TPR-tree by keeping aggregate information in intermediate nodes. To answer a PRA query $q(q_R,q_T)$, the root node is first retrieved and its entries are compared with *q*. For every entry, there are three cases: 1) the MBR of this entry does not intersect query area q_R all through the query time q_T, then this subtree is not visited further; 2) the entry is totally contained in the query area q_R during q_T, and we simply add the aggregate information without accessing its subtree; 3) the entry partially intersects the area q_R during q_T, then its child nodes need to be fetched and continue searching until the leaf. Using this approach, the cost of processing PRA queries can be significantly reduced.

However, the TPR-tee is constructed merely in the space domain, and slightly considering the velocity distribution of moving objects. At construction time, TPR-tree index clusters objects mainly according to their spatial proximity into different nodes; however, the velocities of objects in the same page are always discrepant greatly. The minority of objects with higher velocity make the velocity-bounding rectangle (VBR) relatively larger. In addition, the MBR will increase extremely with time, causing deterioration of the query performance and dynamic maintenance. Actually, in most applications PRA queries often visit unnecessary intermediate TPR-tree nodes due to the extremely large MBRs and massive dead space.

Figure 3 shows the moving objects in two dimensional space with velocity vector (10, 0) and (-10, 0) for example. Figure 3 a) illustrates the MBRs of TPR-tree constructed merely in space domain. Obviously, the MBRs of intermediate TPR-tree nodes extend extremely along the horizontal direction, thus incurring the degradation of query performance. Therefore, an effective aggregate query indexing technique needs to consider the distribution of moving objects in both velocity and space domain to avoid the deterioration of query and dynamic performance.

Motivated by this, we propose a predictive range aggregate tree (PRA-tree) index structure. First, the velocity domain is partitioned into buckets with about the same objects number. Then moving objects are classified into different velocity buckets by their velocities, and objects with similar velocities are clustered into one bucket. Finally, an aTPR-tree structure is presented for indexing moving objects in each bucket.

Figure 3 b) and c) shows the MBRs of PRA-tree that considers the distribution of velocity domain. The moving objects are classified into two velocity buckets. The bucket1 (as shown in figure 3 b)) contains the moving objects with velocity vector (10, 0), while bucket2 (as shown in figure 3 c)) contains the moving objects with velocity vector (-10, 0). As seen, the MBR of each bucket moves with time, but the shapes of MBR keep unchanged. So PRA-tree can hold a good query performance during all the future time.

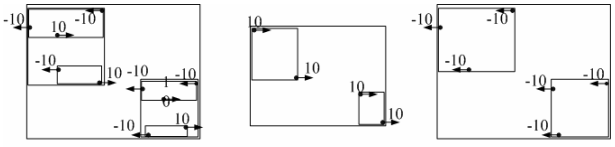


Figure 3 MBRs of PRA-tree

In addition, existing TPR-tree update algorithms work in a top-down manner. For each update, one index traversal to locate the item for deletion and another traversal to insert a new item are needed. The deletion operation affects the disk I/O cost greatly, for the MBRs of TPR-tree become looser with time, thus incurring lots of area overlaps between MBRs, and the searching operation for deletion needs to visit all the TPR-tree nodes at worst case. The top-down approach for visiting the hierarchical structure is very simple, and easy for dynamic maintenance. Nevertheless, for TPR-tree like index structures, the area between intermediate node MBRs inevitably overlaps each other (which is not occurred in other traditional indexes such B-tree), so that the top-down searching strategy is inefficient in nature. This manner results in the deterioration of TPR-tree, and is not suitable in frequent update applications. Motivated by this, we exploit the bottom-up delete strategy to improve the dynamic performance of PRA-tree.

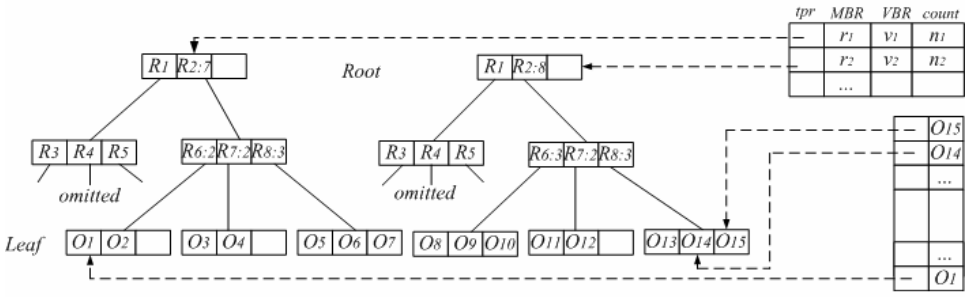


Figure 4The structure of PRA-tree

4. The PRA-tree

Section 4.1 discusses the structure of PRA-tree, while Section 4.2 provides the construction method, Section 4.3 gives the insertion, deletion and update techniques, Section 4.4 evaluates the effect factors of PRA-tree performance.

4.1 The PRA-tree Structure

Specifically, in PRA-tree structure we keep the basic TPR-tree index structure and add aggregate summary into its intermediate nodes to make a new index, the aggregate TPR-tree (aTPR-tree). The entries in aTPR-tree are organized as vector <*MBR, VBR, Agg, ptr*>, where *MBR, VBR, Agg, ptr* denote the space area, velocity range, aggregate information of this node and pointer to its subtree respectively. In addition, we introduce a main memory linear queue for management of velocity buckets, the items of which are formed as <*MBR, VBR, Agg, tpr*>, where *MBR, VBR, Agg, tpr* denote the space area, velocity range, aggregate information of this bucket and pointer to its corresponding aTPR-tree respectively. To improve the dynamic performance of PRA-tree, we supplement a disk-based hash index structure to access the leaf node of PRA-tree directly. Further, we modified the original TPR-tree node item as vector <*entry, …, entry, parentptr*>, where *entry* denotes the child nodes contained in this node, and *parentptr* denotes the physical address of its parent node. Compared with node page size, the space consumption by pointer *parentptr* is trivial, and its effect on the fanout of PRA-tree is ignorable.

Figure 4 illustrates the PRA-tree structure. The top right corner is a velocity bucket queue, in which each item describes the MBR, VBR and aggregate information (i.e., count) of its corresponding aTPR-tree. The bottom right corner is a hash index constructed on IDs of moving objects, the item is defined as vector <*oid, ptr*>, where *oid* denotes the identifier of moving objects, and *ptr* denotes physical offset of the object entry in leaf node; the left of Figure 4 is the aTPR-tree structures pointed to by the velocity bucket queue. In this figure, the pointer to parent node is not clearly depicted for concision.

4.2 Construction

We use spatio-temporal histograms as mentioned in [9] on two velocity dimensions to compute the velocity buckets number and their velocity range. The main idea is to divide the moving objects set into different partitions with about the same objects number. Then the algorithm scans the set of moving objects in sequence, and inserts objects into the aTPR-tree pointed to by corresponding bucket according to their velocities. To avoid redundant disk I/Os caused by frequent insertion one by one, construction algorithm exploits the bulk loading technique [9] to construct the aTPR-tree. However, unlike the traditional bulk loading method, the construction algorithm must compute and store the aggregate information in the intermediate nodes.

The foremost problem for PRA-tree is to choose the number of velocity buckets. With too few velocity buckets the PRA-tree can not gain the optimal query and dynamic maintenance performance, while too many velocity buckets may cause shifts between velocity buckets when update, thus resulting in the deterioration of index structure. So choosing a proper number of velocity buckets can make the query and dynamic performance of PRA-tree optimal. In the experimental section, we evaluate the effect of velocity buckets number on the query and dynamic maintenance performance.

4.3 Insertion , Deletion and Update

The insertion algorithm is straightforward. When inserting a new moving object into the PRA-tree index, the algorithm first scans the main-memory velocity bucket queue to find the bucket that contains this object, and the aTPR-tree related to this bucket can be obtained by the pointer in the queue item. Then a new object entry is produced and inserted into a suitable leaf node using the standard TPR*-tree insertion algorithm. Finally, a new item is produced and inserted into the hash index. If overflow occurs while insertion, the aTPR-tree node must be split with the method mentioned in [4]. In addition, the algorithm must modify the aggregate information of this bucket and aTPR-tree nodes along the searching path.

To delete an object entry from the PRA-tree, a simple straight approach is to first find the velocity bucket that

contains this object and delete the object entry from the aTPR-tree with standard deletion algorithm. However, usually the searching path for deletion may be several, so the algorithm must keep the searching path to perform another operation to complete the modification of aggregate information along the path, thus causing redundant disk I/Os. Motivated by this, we exploit the bottom-up deletion strategy like [10] to improve the deletion performance. The deletion algorithm first locates the leaf node that holds the object entry with hash index, and then deletes the entry from this leaf node directly. Then the algorithm ascends the branches of PRA-tree by the pointer *parentptr* until the root node, and modifies the MBR, VBR and aggregate information of intermediate nodes along the path meanwhile. Finally, the corresponding object item in the hash index must be deleted to reflect the change and also the bucket item related to this aTPR-tree must be modified.

The update algorithm uses the standard deletion-insertion mechanism in TPR-tree. The algorithm first deletes the old entry from the PRA-tree and then inserts a new entry into the PRA-tree.

4.4 Effects on Index Performance

PRA-tree partitions moving objects into velocity buckets that do not overlap each other in velocity domain. So the index can perform a good concurrency when a large amount of concurrent dynamic operations occur. Obviously, the concurrency improves with the number of velocity buckets. However, with too large a bucket number the index structure is prone to instability. That is to say, because the velocity range of each bucket is too small, when an object is updated, the likely object's shift from one bucket to another bucket may cause excessive disk I/Os. In addition, limited by the system memory, and considering the space and velocity distribution uncertainty of moving objects, too many velocity buckets do not reduce node accesses for processing PRA query. Therefore, PRA-tree with a proper number of velocity buckets can obtain the optimal query and update performance.

The cost of maintain the velocity bucket queue is inexpensive. The queue is constructed along with PRA-tree, when dynamic operations such as insertion and deletion occurred, the summary in which must also be updated. The queue is pinned in main memory, thus decrease the visiting cost greatly. And the space consumed by velocity queue is rather small and ignorable.

5 The Predicted Range Aggregate Query Algorithm

We now illustrate the algorithm for answering a predicted range aggregate query with PRA-tree. There are two types of queries: predicted time-slice range aggregate query and predicted window range aggregate query, which have been thoroughly surveyed in [1]. For example, we consider the following queries: i) "how many cars are expected to appear at the center of city 10 minutes from now?" and ii) "how many cars are expected to cross the center of city during the next 5 minutes?" Section 5.1 presents the algorithm for queries as case 1, and section 5.2 details the algorithm for queries as case2.

5.1 Predicted Time-slice Aggregate Range Query (PTRA Query)

PTRA query returns the aggregate information of objects that will fall in a rectangle at a future timestamp based on the current motion of moving objects. Obviously, according to the current MBR and VBR of each node in PRA-tree, we can get the MBR of this node at future timestamp, and then an aggregate range search is implemented under the PRA-tree at this timestamp.

Specifically, given a PTRA query $q(q_R,q_T)$, the algorithm first scans the velocity bucket queue, and for each bucket we can get whether any object in this bucket lies in the query area q_R at timestamp q_T according to the bucket's MBR and VBR. If q_R does not intersect the space area covered by a velocity bucket at future timestamp q_T, the aTPR-tree pointed by this bucket need not to be visited. Or else if q_R contains the space area covered by a velocity bucket at future timestamp q_T, the aggregate information is returned from the bucket item directly and added to the query result. Otherwise, if q_R intersects the space area covered by a velocity bucket at q_T, the corresponding aTPR-tree is obtained and then from the root point, the algorithm recursively computes the topological relation between the MBR of this node at future timestamp q_T and query area q_R. If the MBR is contained in q_R, then the algorithm adds the aggregate information in this node to the query result, or else if the MBR does not intersect q_R each other, then the node is skipped; otherwise this subtree is explored further until the leaf level.

5.2 Predicted Window Aggregate Range Query (PWRA Query)

PWRA query returns the aggregate information of objects that will fall in a query rectangle at a future time window based on current motion of moving objects. To answer a PWRA query, a straightforward method is to judge whether the MBR of current node is totally contained in the query area at some future time. If true, then the algorithm only adds the aggregate information without visiting this subtree; or else if the MBR of this node does not intersect the query area at any future time, then this node is skipped. Otherwise, this subtree is explored further. This approach will access nodes whose MBR partially intersect the query area while the moving objects enclosed in totally lie in the query area at different timestamp (as shown in figure 5),thus causing excessive disk I/Os.

Motivated by this, we present an enhanced predictive range aggregate query (EPRA) algorithm with a more precise branch and bound criterion. Before introducing the EPRA algorithm, we give the following lemmas.

Lemma 1 Given a moving rectangle *R* with fixed edge velocities and a static query rectangle *q*, let $q \cap R \neq \Phi$. If the four vertices of *R* lie in the query area *q* at different future timestamps, then the moving points enclosed in *R* will lie in the query area *q* during the future time.

Proof: As an illustration of Lemma 1, consider Figure 5 where the relationship between *R(a, b, c, d)* and query *q* is shown as case a) and case b), since $q \cap R \neq \Phi$. Supposing the VBR of *R* is$< v_x^-, v_x^+, v_y^-, v_y^+ >$, then the direction of VBR can only be depicted as in Figure 5 a) and b), for vertices *a, b, c* and *d* will lie in *q* at future time.

In case a), *c* has the velocity $< v_x^-, v_y^+ >$, for $\forall p \in R$ point *p* has the velocity $< v_x, v_y >$, where $v_x > v_x^-$ and $v_y < v_y^+$. If *c* lies in *q* at future timestamp t_c, then *c* must cross the line l_2 and not cross the line l_3 at t_c. So that for point *p*, it must have crossed line l_2 before t_c and still not cross line l_3, the *p* lies in *q* at some timestamp before t_c.

In case b): for *a, b* and *c* will lie in *q* at future times, obviously at least either *b* or *c* must lies in *q* at some future timestamp t_s with *d*, then at t_s the topological relationship between *R* and *q* can be illustrated as Figure 5 a). According to the discussion above, we can conclude that every point enclosed in *R* will lie in *q* at some future timestamp.

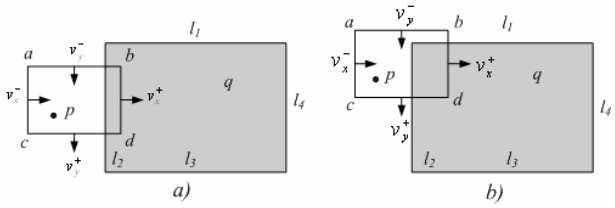


Figure 5 Topological relations between *R* and *q*

Lemma 2 Given a moving rectangle *R* with fixed edge velocities and a static query rectangle *q*, let $q \cap R = \Phi$. If the four vertices of *R* lie in the query area *q* at different future timestamps, then the moving points enclosed in *R* will lie in the query area *q* during the future time.

Proof: Supposing vertex *d* to be the first point that will lie in the query *q*, then at this timestamp the topological relationship between *R* and *q* can be shown as in Figure 5. So according to Lemma 1, we can deduce Lemma 2.

Heuristic 1: Given an intermediate entry *E* and a PRA query $q(q_R,q_T)$, the subtree of *E* totally satisfy query *q* if the vertices of MBR in entry E lie in query area q_R during future time q_T. If true, the query algorithm only returns the aggregate information of *E* without accessing its subtree.

Heuristic 1 reduces the searching cost considerably, while incurring rather small computational overhead. Specifically, the algorithm first scans the velocity bucket queue, according to the current MBR and VBR of this bucket, the future timestamps when each vertex lies in the query area q_R can computed. If all the vertices can lie in q_R during the query window q_T, then the algorithm only need to return the aggregate information of this bucket; or else if none of the vertices will lie in q_R during the query window q_T, then the algorithm skips this bucket; otherwise the bucket is explored further. Similarly, when searching the aTPR-tree, from the root point the timestamps when four vertices of MBR in each node lie in q_R is computed. If all the vertices can lie in q_R during the query window q_T, then the algorithm only need to return the aggregate information of this node; or else if none of the vertices will lie in q_R during the query window q_T, then the algorithm skips this node; otherwise this subtree is explored further.

The Algorithm 1 describes the pseudo-code for processing PWRA query as follows:

Algorithm 1
Input: a PWRA query $q(q_R,q_T)$, output: *Agg(q)*
1. Initialize *Agg(q)*, set $Agg(q) \leftarrow 0$
2. For each item *E* in the velocity bucket queue
3. Compute the timestamps when each vertex lies in q_R;
4. If all the timestamps lie between q_T
5. Then $Agg(q) \leftarrow Agg(q) + E.Agg$;
6. Else if none of the timestamps lies between q_T
7. Then skip *E*;
8. Else get the aTPR-tree pointed by this bucket item *E*;
9. from the root point, for each entry E in this node
10. Compute the timestamps when each vertex lies in q_R;
11. If all the timestamps lie between q_T
12. Then $Agg(q) \leftarrow Agg(q) + E.Agg$
13. Else if none of the timestamps lies between q_T
14. then skip *E*
15. Else explore the subtree recursively until leaf level
16. Compute the aggregate information in leaf node E
17. Set $Agg(q) \leftarrow Agg(q) + E.Agg$
18. End if
19. End for
20. End if
21. End for
End algorithm 1

6 Experimental Results and Performance Analysis

6.1 Experimental Setting and Details

In this section, we evaluate the query and update performance of PRA-tree with aTPR-tree and TPR*-tree. We use the Network-based Generator of Moving Objects [11] to generate 100k moving objects. The input to the generator is the road map of Oldenburg (a city in Germany). An object appears on a network node, and randomly chooses a destination. When the object reaches its destination, an update is reported by randomly selecting the next destination. When normalize the data space to 10000×10000, the default velocity of objects is

equal to 20 per timestamp. At each timestamp about 0.8 % moving objects update their velocities. The PRA queries are generated as follows: i) the query spatial extent *qRlen* is set as 100×100,400×400,800×800,1200×1200,1600×1600 respectively, and the starting point of its extent randomly distributes in (10000-*qRlen*)×(10000-*qRlen*).ii) the query time window is [*Tisu*，*Tisu*+*qTlen*] (*Tisu* is the time when the query is presented), where *qTlen*=20,40,60,80,100 respectively. The query performance is measured as the average number of node accesses in processing ten PRA queries with the same parameters. The update performance is measured as the average node accesses in executing 100 moving objects updates.

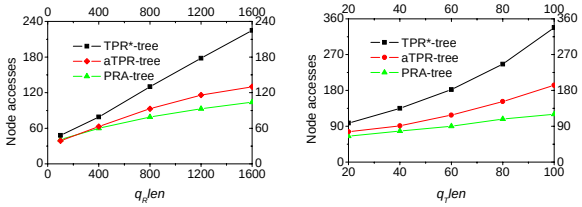
Table 1 summarizes the parameters of PRA-tree exploited for the workload. For all simulations, we use a Celeron 2.4GHz CPU with 256MByte memory.

Table 1 PRA-tree parameters

Parameter	Value	Description
Number of buckets	25/50/100/150/200/250	The velocity domain is split into 5×5, 5×10, 10×10, 10×15, and 10×25 respectively.
Page size	1k	The page size of PRA-tree.
Fanout	21	The average fanout of PRA-tree in intermediate node.
Average height	3	The average height of PRA-tree.

6.2 Performance Analysis

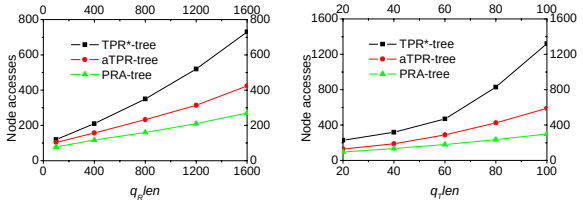
We compare the query and update performance of PRA-tree, TPR*-tree and aTPR-tree by node accesses. In order to study the deterioration of the indexes with time, we measure the performance of PRA-tree, aTPR-tree and TPR*-tree, using the same query workload, after every 5k updates.



a) *qTlen*=50 b) *qRlen*=1000

Figure 6 comparison of PTRA query performance

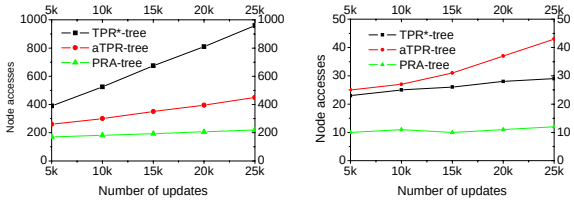
Figure 6 a) and b) shows the PTRA query cost as a function of *qRlen* and *qTlen* respectively. In Figure 6 a) we fix the parameter *qTlen*=50 and in Figure 6 b) we fix the parameter *qRlen*=1000. As seen, the query performance of PRA-tree works best and TPR*-tree exhibits a worst performance. This is because PRA-tree is constructed both on the space and velocity domain, the area of MBRs overlap relatively less than those of aTPR-tree and TPR*-tree, thus having a good query performance. While TPR*-tree and aTPR-tree may visit many unnecessary nodes for the massive overlaps between area of MBRs with time, causing a worse query performance. In addition, the larger the query range area, the better the PTRA query performance of PRA tree, for the intermediate nodes in PRA-tree store the aggregate information of this subtree, thus reducing the node accesses needed by TPR*-tree. The aTPR-tree holds a relative worse performance than PRA-tree for its larger MBRs with time.



a) *qTlen*=50 b) *qRlen*=1000

Figure 7 comparison of PWRA query performance

Figure 7 a) and b) shows the PWRA query cost as a function of *qRlen* and *qTlen* respectively. In Figure 7 a) we fix the parameter *qTlen*=50 and in Figure 7 b) we fix the parameter *qRlen*=1000. As seen, the query performance of PRA-tree works best and TPR*-tree exhibits a worst performance. This is because the MBRs PRA-tree overlap relatively less than those of aTPR-tree and TPR*-tree, thus having a good query performance. While TPR*-tree and aTPR-tree may visit many unnecessary nodes for the massive overlaps between area of MBRs with time. Furthermore, PRA-tree exploits more precise branch-and-bound strategy, thus having a best performance.



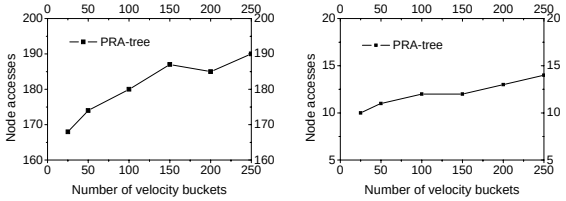
a) update cost b) query cost

Figure 8 comparison of update& query performance

Figure 8 compares the average PRWA query and update cost as a function of the number of updates. As shown in figure 8 a), the node accesses needed in PRA-tree update execution are far less than TPR*-tree and aTPR-tree. And PRA-tree and TPR*-tree have nearly constant update cost This is because PRA-tree exploits the bottom-up deletion strategy, avoiding the excessive node accesses for deletion search, while TPR*-tree and aTPR-tree process update in top-down manner, needing more node accesses.

Figure 8 b) shows the node accesses in processing one PWRA query as function of an interval of 5k updates when fixing the parameters *qTlen*=50 and *qRlen*=1000. It is clear that the query cost increases with the number of updates. The PRA-tree has a slow increasing query cost, while the cost of TPR*-tree and aTPR-tree increase

significantly. This is because the MBRs of PRA-tree extend less than those of TPR*-tree and aTPR-tree, so the degradation is not much expensive.



a) effect on update cost b) effect on query cost

Figure 9 effect of velocity bucket number

Figure 9 shows the effect of velocity buckets number on PWRA query and update performance of PRA-tree. We fix the parameters *qTlen*=50 and *qRlen*=1000. As shown in Figure9 a), the update cost of PRA-tree increase slightly with the number of velocity buckets, because the small velocity bucket window causes the shift of moving objects between buckets, thus incurs excessive node accesses. From Figure 9 b) we can see, the PWRA query performance will degrade with too many buckets, this is because of, as mentioned earlier, the uncertainty of moving objects distribution in velocity and space, the probability of overlaps between the area covered by velocity buckets and query region don't decrease linearly as expected. However, it may bring more TPR-tree node accesses. In this experiment, we set the number of velocity bucket equal to 25.

7. Conclusion

This paper investigates the problem of PRA queries. Our contribution is a novel indexing method, referred to as predicted range aggregate R-tree (PRA-tree). PRA-tree considers the distribution of moving objects both in velocity and space domain. First, we partition the velocity domain into velocity buckets, and then we use aTPR-tree to index the moving objects in each bucket. To support frequent updates a supplemented hash index on leaf nodes is added to PRA-tree. Also an extended bottom-up deletion algorithm is developed for PRA-tree.

An open problem is to extend our work to support large scale of concurrent PRA queries efficiently. Further, the PRA query results need to be revaluated for the frequent object updates, so developing novel incremental algorithms for PRA queries is also an interesting work.

References

[1] Yufei Tao and Dimitris Papadias, Range Aggregate Processing in Spatial Databases. *IEEE TKDE*, NO.12, pp.1555-1570, 2004.

[2] Ine´s Fernando Vega Lo´pez, Richard T. Snodgrass, and Bongki Moon. Spatiotemporal Aggregate Computation: A Survey. *IEEE TKDE*, NO.2, pp.271-286, 2005.

[3] Simonas Saltenis, Christian S. Jensen, et al.. Indexing the Positions of Continuously Moving Objects. *Proc. SIGMOD Conf.*, pp.331-342, 2000.

[4] Tao Y., Papadias D., and Sun J.. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. *Proc. VLDB Conf.*, pp.790-801, 2003.

[5] Jignesh M. Patel, Yun Chen, V. Prasad Chaka. STRIPES: An Efficient Index for Predicted Trajectories. *Proc. SIGMOD Conf.*, pp.635-646, 2004.

[6] Yufei Tao, Dimitris Papadias, Jian Zhai, and Qing Li. Venn Sampling: A Novel Prediction Technique for Moving Objects. *Proc. Int'l Conf. Data Eng.*, pp.680-691, 2005.

[7] Yufei Tao, Jimeng Sun, Dimitris Papadias. Selectivity Estimation for Predictive Spatio-Temporal Queries. *Proc. Int'l Conf. Data Eng.*, pp.417-428, 2003.

[8] M. Jurgens and H. Lenz. The Ra*-Tree: An Improved R-Tree with Materialized Data for Supporting Range Queries on OLAP-Data. *Proc. DEXA Workshop*, 1998.

[9] Bin Lin and Jianwen Su. On bulk loading TPR-tree. *Proc. IEEE Conf. Mobile Data Management (MDM)*, pp.114-124, 2004.

[10] M. Lee, W. Hsu, C. Jensen, B. Cui, and K. Teo. Supporting Frequent Updates in R-Trees: A Bottom-Up Approach. *Proc. VLDB Conf.*, pp.608-619, 2003.

[11] Thomas Brinkhoff. A Framework for Generating Network-Based Moving Objects. *GeoInformatica*, Vol.6 (2), pp.153-180, Kluwer, 2002.

Additively Weighted Voronoi Diagrams for Optimal Sequenced Route Queries*

Mehdi Sharifzadeh and Cyrus Shahabi

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
[sharifza, shahabi]@usc.edu

Abstract

The *Optimal Sequenced Route (OSR)* query strives to find a route of minimum length starting from a given source location and passing through a number of *typed* locations in a specific sequence imposed on the types of the locations. In this paper, we propose a pre-computation approach to OSR query in vector spaces. We exploit the geometric properties of the solution space and theoretically prove its relation to *Additively Weighted Voronoi diagrams*. Our approach recursively accesses these diagrams to incrementally build the optimal sequenced route. Our experimental results verify that our pre-computation approach outperforms the previous index-based approaches in terms of query response time.

1 Introduction

Suppose that we are planning a Saturday trip in town as following: first we intend to visit a shopping center in the afternoon to check the season's new arrivals, then we plan to dine in an Italian restaurant in early evening, and finally, we would like to watch a specific movie at late night. Naturally, we intend to drive the minimum overall distance to these destinations. In other words, we need to find the locations of the shopping center s_i, the Italian restaurant r_j, and the theater t_k that shows the specific movie, which driving toward them considering the sequence of the plan shortens our trip (in terms of distance or time). Note that in this example, a time constraint enforces the order in which these destinations are to be visited; we usually do not have dinner in the afternoon, or do not go for shopping at late night. This type of query is also essential in other application domains such as *crisis management*, *air traffic flow management*, *supply chain management*, and *video surveillance*.

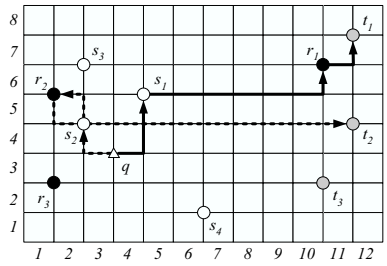


Figure 1: 3 different types of point sets

We call this type of query, where the order of the sequence in which some points must be visited is *enforced* and cannot be changed, the *Optimal Sequenced Route* or *OSR* query. Similar types of the planning queries have received recent attention by the database community [5, 9, 3, 2] to solve problems in the domains of air traffic flow and supply chain management, which shows the importance of these query types. We use the first example described above as our running example throughout the paper. Figure 1 shows three different types of *point sets* shown by white, black and gray circles, which represent shopping centers, Italian restaurants, and theaters, respectively, and a starting point q (shown by $\triangle$). The distance between two points is their Manhattan distance. Here, the route (q, s_1, r_1, t_1) (shown with solid lines in the figure) with the length of 12 units is the optimum answer to our query. One can show that OSR query may not be optimally answered by simply performing a series of independent nearest neighbor queries at different locations (e.g., 15-unit length route (q, s_2, r_2, t_2)).

We, for the first time, introduced the OSR query in [8] and proposed two online approaches for vector and metric spaces (e.g., road networks). The R-LORD algorithm of [8] subsequently performs range queries utilizing an R-tree index structure to filter out the points that cannot possibly be on the optimal route, and then builds the optimal route in reverse sequence (i.e., from

*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.

Proceedings of the third Workshop on STDBM Seoul, Korea, September 11, 2006

ending to the starting point). In this paper, we exploit the geometry of the OSR problem space to propose a novel *pre-computation* algorithm which utilizes Voronoi diagrams for processing OSR queries in vector spaces. In our running examples, we use Manhattan distance in $\mathbb{R}^2$ which is a representative of the network distance if the road network consists of a set of north-south and east-west roads [1].

1.1 Contributions

Assume that we are only interested in the optimal sequenced routes that follow a fixed sequence M. For example, assume that this fixed sequence for Figure 1 is $(white, black, grey)$. Suppose we can partition the space of points of interest (e.g., $\mathbb{R}^2$) into regions each including all the points with a common OSR. For instance, in Figure 1, there are many points similar to q for which the optimal sequenced route to a *white*, then a *black*, and finally a *grey* point is the route (s_1, r_1, t_1). Suppose that our partitioning identifies the region including these points. Furthermore, assume that the partitioning also identifies the region corresponding to each and every possible route (s_i, r_j, t_k) to a *white*, a *black*, and a *grey* point. Assume that we associate and store all regions and their corresponding OSRs. Therefore, for a given starting point q we can address the OSR query with sequence M by simply locating the *unique* region that includes q and reporting its corresponding optimal sequenced route. In Section 3, we prove that such partitioning exists as an *Additively Weighted Voronoi diagram*, a specific type of general Voronoi diagrams. We theoretically exploit interesting properties of these diagrams which makes them appropriate data structures for efficient OSR query processing.

We enhance the above query processing approach by exploiting an interesting property of OSRs proved in Lemma 1 (see Section 3). The lemma states that if (s_1, r_1, t_1) is q's optimal route to a *white*, a *black*, and a *grey* point, then (r_1, t_1) is s_1's optimal route to a *black* and then a *grey* point. *Recursively*, (t_1) is r_1's optimal route to a *grey* point. Lemma 1 enables us to avoid storing the complete OSR corresponding to each region of the diagram for a given sequence M; we store only the first point of the OSR for each region of the Voronoi diagram (e.g., s_1 for the region corresponding to (s_1, r_1, t_1)). Instead, we require to compute and store the Voronoi diagrams corresponding to all suffixes of M.

For instance, to answer queries with sequence $(white, black, grey)$ in Figure 1, our approach requires the diagrams corresponding to three sequences $(white, black, grey)$, $(black, grey)$, and $(grey)$. We refer to these diagrams as the *OSR-Voronoi family* of sequence $(white, black, grey)$. Now, we iteratively process a given OSR query. For the starting point q, we first find s_1, the white point associated with the region including q in the diagram of $(white, black, grey)$. Then, we find r_1, the black point stored with the region including s_1 in the diagram of $(black, grey)$. Finally, we seek for t_1, the grey point corresponding to the region including r_1 in the diagram of $(grey)$. Now, the final OSR of q is the route (s_1, r_1, t_1), which includes s_1, r_1, and t_1 in the order of their retrieval.

Our OSR query processing using Voronoi diagrams best suits the applications where the sequences of user's interest are known in advance. This allows pre-computation of the corresponding diagrams. Notice that using the OSR-Voronoi family of a sequence M, we can address OSR queries of *all suffix sequences* of M. Through extensive experiments with a real-world dataset, we show that our approach significantly outperforms the R-LORD approach proposed in [8] in terms of response time. We also show that the off-line process used to build the OSR-Voronoi family of a single sequence is reasonably fast. With a given sequence M, this pre-computation phase takes $O(\sum_{i=1}^{|M|} |U_{M_i}| \log |U_{M_i}|)$ computation steps.

2 Preliminaries

2.1 Formal Problem Definition

In this section, we describe the terms and notations used in [8] to formally define the OSR query. We use the same notations throughout the paper .

Let $U_1, \ldots, U_n$ be n sets, each containing points in a d-dimensional space $\mathbb{R}^d$, and $D(.,.)$ be a distance metric defined in $\mathbb{R}^d$ where $D(.,.)$ obeys the triangular inequality. To illustrate, in the example of Figure 1, U_1, U_2, and U_3 are the sets of black, white, and gray points, representing restaurants, shopping centers and theaters, respectively. We first define the following four terms.

Definition 1: Given n, the number of point sets U_i, we say $M = (M_1, M_2, \ldots, M_m)$ is a *sequence* if and only if $1 \leq M_i \leq n$ for $1 \leq i \leq m$. That is, given the point sets U_i, a user's OSR query is valid only if she asks for existing location types. For the example of Figure 1 where $n = 3$, $(2, 1, 2)$ is a sequence (specifying a shopping center, a restaurant, and a shopping center), while $(3, 4, 1)$ is not a sequence because 4 is not an existing point set. We use $sfx(M, i) = (M_{i+1}, \ldots, M_m)$ to refer to the suffix of M with size $m - i$.

Definition 2: We say $R = (p_1, p_2, \ldots, p_r)$ is a *route* if and only if $p_i \in \mathbb{R}^d$ for each $1 \leq i \leq r$. We use $p \oplus R = (p, p_1, \ldots, p_r)$ to denote a new route that starts from starting point p and goes sequentially through p_1 to p_r. The route $p \oplus R$ is the result of adding p to the head of route R. We use $sfx(R, i) = (p_{i+1}, \ldots, p_r)$ to refer to the suffix of R with size $r - i$.

Definition 3: We define the *length* of a route $R = (p_1, p_2, \ldots, p_r)$ as

$$L(R) = \sum_{i=1}^{r-1} D(p_i, p_{i+1}) \quad (1)$$

Note that $L(R) = 0$ for $r = 1$. For example, the length of the route (s_2, r_2, s_3) in Figure 1 is 4 units where D is the L_1 norm (i.e., Manhattan distance).

Definition 4: Let $M = (M_1, M_2, \ldots, M_m)$ be a sequence. We refer to the route $R = (p_1, p_2, \ldots, p_m)$ as a

sequenced route that follows sequence M if and only if $p_i \in U_{M_i}$ where $1 \leq i \leq m$. In Figure 1, (s_2, r_2, s_3) is a sequenced route that follows $(2, 1, 2)$ which means that the route passes only through a white, then a black and finally a white point.

Now, we formally define the OSR query.

Definition 5: For a given *starting* point q in $\mathbb{R}^d$ and the sequence $M = (M_1, M_2, \ldots, M_m)$, the **Optimal Sequenced Route** (OSR) Query, $Q(q, M)$, is defined as finding a sequenced route $R = (p_1, \ldots, p_m)$ that follows M where the value of the following function L is minimum over all the sequenced routes that follow M:

$$L(q, R) = D(q, p_1) + L(R) \quad (2)$$

Note that $L(q, R)$ is in fact the length of route $R_q = q \oplus R$. Throughout the paper, we use $Q(q, M) = (p_1, p_2, \ldots, p_m)$ to denote the answer to the OSR query Q. Without loss of generality, we assume that this optimal route is unique for given q and M. For the example in Section 1 where $(U_1, U_2, U_3) = (black, white, gray)$, $M = (2, 1, 3)$, and $D(.,.)$ is the shortest path distance, the answer to the OSR query is $Q(q, M) = (s_1, r_1, t_1)$.

One special variation of OSR is when the user asks for an optimal sequenced route that ends in a *given* destination q'. A special case of this query is where she intends to return to her starting location q. This variation of OSR can simply be addressed by defining a new point set $U_{n+1} = \{q'\}$ and a new sequence $M' = (M_1, \ldots, M_m, n+1)$. Consequently, $Q(q, M')$ will be the optimal route following M which ends in q'.

Table 1 summarizes the notations we use throughout the paper.

2.2 Voronoi Diagrams

The general Voronoi diagram of a given set S including points in $\mathbb{R}^d$ partitions the space into regions each including all points with a common closest point in the given set according to a distance metric $\mathbf{d}(.,.)$. The region corresponding to the point p contains all the points $q \in \mathbb{R}^d$ for which we have

$$\forall p' \in S, p' \neq p \Rightarrow \mathbf{d}(q, p) \leq \mathbf{d}(q, p') \quad (3)$$

The equality holds for the points on the borders of p's and p''s regions. Incorporating arbitrary distance metrics $\mathbf{d}(.,.)$ in the above equation results in different variations of Voronoi diagrams. Figure 2 shows the *standard* Voronoi diagram of nine points in $\mathbb{R}^2$ where the distance metric is Euclidean. We refer to the region $V(p)$ containing the point p as its Voronoi *cell*. We also refer to point p as the *generator* of Voronoi cell $V(p)$.

Now assume that S is a set of points $p \in \mathbb{R}^d$ each assigned a weight $w(p) \in \mathbb{R}$. We define the distance metric $\mathbf{d}(x, p)$ as $D(x, p) + w(p)$ where $D(.,.)$ denotes a distance metric. Without loss of generality, we assume that $D(.,.)$ is Euclidean distance. The Additively Weighted (AW) Voronoi diagram of the points in S with respect to distance function $D(.,.)$ is defined using Equation 3. Here, the cell corresponding to point p

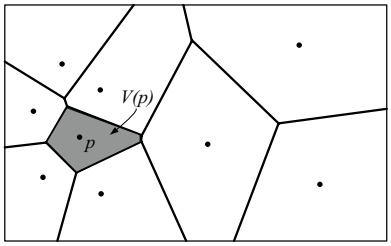


Figure 2: The standard Voronoi diagram of 9 points, the point p and its Voronoi cell $V(p)$

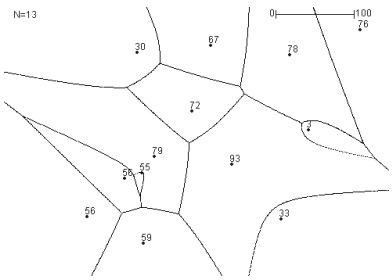


Figure 3: The AW-Voronoi diagram of 13 points each labeled with its positive weight

contains all the points $q \in \mathbb{R}^d$ for which we have

$$\forall p' \in S, p' \neq p \Rightarrow D(q, p) + w(p) \leq D(q, p') + w(p') \quad (4)$$

Figure 3 illustrates the AW-Voronoi diagram of a set of 13 points labeled with their positive weights.

The Computational Geometry literature generally uses *negative* weights in Equation 4 to define the AW-Voronoi diagrams. When negative weights are assigned to points p, their AW-Voronoi diagram can be seen as the standard Voronoi diagram of a set of d-dimensional balls each centered at a point p with a radius equal to the absolute value of $w(p)$. Here, the set S includes balls such as b centered at a point p. The distance $\mathbf{d}(q, b)$ between a ball b and the point q outside b is the smallest Euclidean distance between q and any point inside b. The literature uses this view to define the AW-Voronoi diagram of the centers of a set of balls (e.g., disks in $\mathbb{R}^2$) [6]. Each region of the space is assigned to a unique ball which is the common closest ball of all points inside that region. Figure 4 illustrates defining the AW-Voronoi diagram of five points with negative weights $w(p) < 0$ using five corresponding disks. Here, $\mathbf{d}(q, c)$, the distance of a point q to a disk c centered at p with radius $r = -w(p)$ is defined as $D(q, p) - r$. As the figure shows, point q which is inside the cell of disk c (corresponding to point p) is closer to disk c than to disk c'. It also shows that some points such as p'' have empty cells. That is, no point in the space is closer to the disk c'' than to any other disk.

The AW-Voronoi diagram demonstrates the following properties:

Property AWV-1 With Euclidean distance (i.e. L_2 norm) as metric $D(.,.)$ in Equation 4, the cell edges are either straight lines or hyperbolic curves [6].

Symbol	*Meaning*	*Symbol*	*Meaning*
U_i	a *point set* in $\mathbb{R}^d$	$\|U_i\|$	cardinality of the set U_i
n	number of point sets U_i	$D(.,.)$	distance function in $\mathbb{R}^d$
M	a sequence, $= (M_1, \ldots, M_m)$	$\|M\|$	m, size of sequence M = number of items in M
M_i	i-th item of M	R	route $(p_1, p_2, \ldots, p_r)$, where p_i is a point
$\|R\|$	r, number of points in R	p_i	i-th point in R
$L(R)$	length of R	$q \oplus R$	route $R_p = (q, p_1, \ldots, p_r)$ where $R = (p_1, \ldots, p_r)$
$sfx(R,i)$	route $(p_{i+1}, \ldots, p_r)$, a suffix of R	$sfx(M,i)$	sequence $(M_{i+1}, \ldots, M_m)$, a suffix of M
$L(q,R)$	$L(q \oplus R)$, length of the route $q \oplus R$		

Table 1: Summary of notations

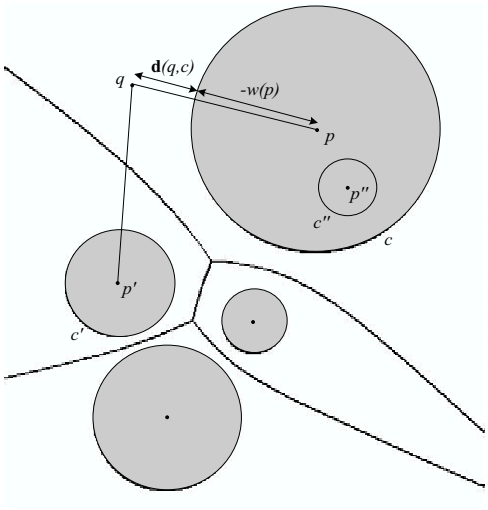


Figure 4: The AW-Voronoi diagram of five disks (i.e., five points with negative weights). Here, the distance function $D(.,.)$ is Euclidean.

Hence, these curved cells are not convex polygons as the cells of general Voronoi diagrams. Using Manhattan distance (i.e. L_1 norm), all edges become straight lines.

Property AWV-2 The arbitrary numeric values of weights can result in empty cells [6]. We refer to the generator point of an empty cell as a *trivial* point.

Property AWV-3 The AW-Voronoi diagram of n points consists of at most $O(n)$ connected edges [7].

3 OSR and AW-Voronoi Diagrams

Assume that we are only interested in the optimal sequenced routes that follow a fixed sequence M (i.e., $Q(q, M)$ for points such as q). Suppose we can partition the space $\mathbb{R}^d$ into regions each including all the points q with a common OSR. That is, for any two points x and y inside the same region, we have $Q(x, M) = Q(y, M)$. Assume that for each region, we pre-compute this common OSR. Furthermore, we store all regions and their corresponding OSRs for the given sequence M. Therefore, for a given starting point q we can address the query $Q(q, M)$ by simply locating the region including q and reporting its corresponding optimal route.

In this section, we show that the above partitioning which facilitates processing OSR queries exists as an AW-Voronoi diagram. We prove that a unique property of optimal sequenced routes enables us to employ this partitioning. First, we prove Lemma 1 which states that the optimal route from p_i that follows a suffix of the original sequence M is the same as the corresponding suffix of the optimal route from q that follows M.

Lemma 1. *Given the sequence $M = (M_1, \ldots, M_m)$ and the starting point q, if $Q(q, M) = R = (p_1, \ldots, p_m)$, then for any $1 \leq i < m$, we have $Q(p_i, M') = sfx(R, i)$ where $M' = sfx(M, i)$.*

Proof. The proof is by contradiction. Assume that $Q(p_i, M') = R' = (p'_1, \ldots, p'_{m-i})$. Obviously $sfx(R, i) = (p_{i+1}, \ldots, p_m)$ follows sequence M', therefore we have $L(p_i, R') < L(p_i, (p_{i+1}, \ldots, p_m))$. We add $L(q, (p_1, \ldots, p_i))$ to the both sides of this inequality to get:

$$L(q, (p_1, \ldots, p_i, p'_1, \ldots p'_{m-i})) < L(q, (p_1, \ldots, p_m))$$

The above inequality shows that the answer to $Q(q, M)$ must be $(p_1, \ldots, p_i, p'_1, \ldots, p'_{m-i})$ which clearly follows sequence M. This contradicts our assumption that $Q(q, M) = (p_1, \ldots, p_m)$. □

This property of the OSRs implies that *only* the last point of the optimal sequenced route $Q(q, M)$ (i.e., p_m) must necessarily be the closest point of U_M to its previous point in the route (i.e., p_{m-1}).

Here, we utilize the special case of Lemma 1 for the first point of the route (i.e., p_1 for $i = 1$). For instance, if the optimal route from point q to a white, a black, and a grey point is (s_1, r_1, t_1), then the optimal route from the point s_1 to a black then a grey point is (r_1, t_1) (see Figure 1). Therefore, if we identify the white point s_1 with which the optimal route from q to a white, a black, then a grey point *starts* (i.e., the first point of the route) and we know that the optimal route from s_1 to a black and then a grey point is (r_1, t_1), we have found the optimal route from q (we just append s_1 to the head of (r_1, t_1)). The following lemma identifies this first point.

Lemma 2. *Given the sequence M and the starting point q, if there exists a point $p_1 \in U_{M_1}$ so that $\forall p'_1 \in U_{M_1}, p'_1 \neq p_1$*

$$D(q, p_1) + L(p_1, Q(p_1, M')) < D(q, p'_1) + L(p'_1, Q(p'_1, M')) \quad (5)$$

where $M' = sfx(M, 1)$, then the optimal route $Q(q, M)$ starts with p_1 which is followed by the points in $Q(p_1, M')$ (i.e., $Q(q, M) = p_1 \oplus Q(p_1, M')$).

Proof. The proof is by contradiction. Assume that $Q(q, M) = (p'_1, \ldots, p'_m)$ starts with $p'_1 \neq p_1$. By definition, the length of $q \oplus Q(q, M)$ is minimum over all the routes which follow sequence M. It is clear that the route $p_1 \oplus Q(p_1, M')$ follows M, so we have

$$L(q, (p_1 \oplus Q(p_1, M'))) \geq L(q, Q(q, M)) \quad (6)$$

Lemma 1 states that we have $Q(p'_1, M') = (p'_2, \ldots, p'_m)$ where $M' = sfx(M, 1)$. Hence, we get

$$Q(q, M) = p'_1 \oplus Q(p'_1, M') \quad (7)$$

Therefore, we replace $Q(q, M)$ in Equation 6 to get

$$L(q, p_1 \oplus Q(p_1, M')) \geq L(q, p'_1 \oplus Q(p'_1, M')) \quad (8)$$

Finally, we use the definition of function $L()$ to change Equation 8 as follows for the point $p'_1 \neq p_1$:

$$D(q, p_1) + L(p_1, Q(p_1, M')) \geq D(q, p'_1) + L(p'_1, Q(p'_1, M')) \quad (9)$$

The above inequality contradicts the one given in the assumption of the lemma.

□

In the example of Figure 1, U_1, U_2, and U_3 are the sets of black, white, and gray points, respectively. For the sequence $M = (2, 1, 3)$, we have $M' = (1, 3)$ and

$$Q(s_1, M') = (r_1, t_1), Q(s_2, M') = (r_3, t_3),$$

$$Q(s_3, M') = (r_1, t_1), Q(s_4, M') = (r_1, t_1)$$

$$D(q, s_1) + L(s_1, Q(s_1, M')) = 3 + (7 + 2) = 12$$

$$D(q, s_2) + L(s_2, Q(s_2, M')) = 2 + (3 + 9) = 14$$

$$D(q, s_3) + L(s_3, Q(s_3, M')) = 4 + (8 + 2) = 14$$

$$D(q, s_4) + L(s_4, Q(s_4, M')) = 5 + (9 + 2) = 16$$

According to Lemma 2, the OSR of q to a white, a black and a grey point starts with s_1 followed by $Q(s_1, M') = (r_1, t_1)$.

We now strive to find the locus of all points in space whose OSR that follows a given sequence starts with a point p_1 and hence all share a common OSR for that sequence. In Figure 1, for any white point s_i, there are points such as q whose optimal routes given the sequence $(white, black, grey)$ start with s_i. Notice that the rest of their optimal route is the same (e.g., (r_1, t_1) for s_1) and depends only on the location of point s_i and the given sequence (e.g., $(black, grey)$).

Theorem 1. *Let $M = (M_1, M_2, \ldots, M_m)$ be a given sequence and $M' = sfx(M, 1)$. The locus of all points q with a common optimal sequenced route $Q(q, M)$ which starts with p_1 is inside p_1's cell in the AW-Voronoi diagram of the set of points in U_{M_1} where the weight of each point p is $w(p) = L(p, Q(p, M'))$. Their common optimal route is $Q(q, M) = p_1 \oplus Q(p_1, M')$.*

Proof. Let the set $S = U_{M_1}$ and $w(p) = L(p, Q(p, M'))$. According to the definition of the AW-Voronoi cell of p_1 given by Equation 4, for the point q inside this cell we get $\forall p'_1 \in U_{M_1}, p'_1 \neq p_1$

$$D(q, p_1) + L(p_1, Q(p_1, M')) < D(q, p'_1) + L(p'_1, Q(p'_1, M'))$$

Therefore, according to Lemma 2 the optimal route $Q(q, M)$ is $p_1 \oplus Q(p_1, M')$ which clearly starts with p_1, the generator of the Voronoi cell including q. □

Given a sequence M, we refer to the AW-Voronoi diagram of the points p in U_{M_1} using the weights $w(p) = L(p, Q(p, M'))$ where $M' = sfx(M, 1)$, as the **OSR-Voronoi diagram** of the sequence M. Figure 5a illustrates the OSR-Voronoi diagram of sequence $(white, black, grey)$ (i.e., $M = (2, 1, 3)$). The distance function $D(.,.)$ is Manhattan distance and each white point s_i is labeled with its weight (i.e., $L(s_i, Q(s_i, (1, 3)))$). The OSR-Voronoi diagram has some interesting properties whose proofs are omitted due to lack of space:

Property OSRV-1 All points in the OSR-Voronoi diagram of the sequence M (any $p \in U_{M_1}$) have positive weights.

Property OSRV-2 Given the sequence M, if for the points p and $p' \in U_{M_1}$ we have $Q(p, M') = Q(p', M') = (p_2, \ldots, p_m)$ and $D(p, p_2) = D(p, p') + D(p', p_2)$ where $M' = sfx(M, 1)$, then p is a trivial point in OSR-Voronoi diagram of sequence M. That is, the Voronoi cell of p is empty and no OSR that follows M starts with p.

4 OSR Query Processing using OSR-Voronoi Diagrams

In this section, we describe our OSR query processing approach that utilizes OSR-Voronoi diagrams defined in Section 3. Assume that we have already built the OSR-Voronoi diagram of a given sequence M. The user asks for the OSR $Q(q, M)$ given a starting point q. Subsequently, we locate the Voronoi cell which contains q in the OSR-Voronoi diagram of M. According to Theorem 1, the OSR of q starts with the generator of this cell (e.g., s_1 in Figure 5a). This point is then followed by the result of another OSR query (i.e., $Q(s_1, M')$). If we already knew the answer to this second query, we immediately report user's desired route as $s_1 \oplus Q(s_1, M')$. Now the problem is that the we need the OSR-Voronoi diagram of sequence M' to answer the second OSR query. Repeating the same reasoning for M' verifies that we will require the OSR-Voronoi diagrams of all suffixes of the original sequence M. That is, to find the OSR which follows $M = (M_1, \ldots, M_m)$ we need the OSR-Voronoi diagrams of all sequences $(M_i, \ldots, M_m)$ for all $1 \leq i \leq m$. We refer to these diagrams as the *OSR-Voronoi family* of M. For instance, to answer queries with sequence $(white, black, grey)$ in

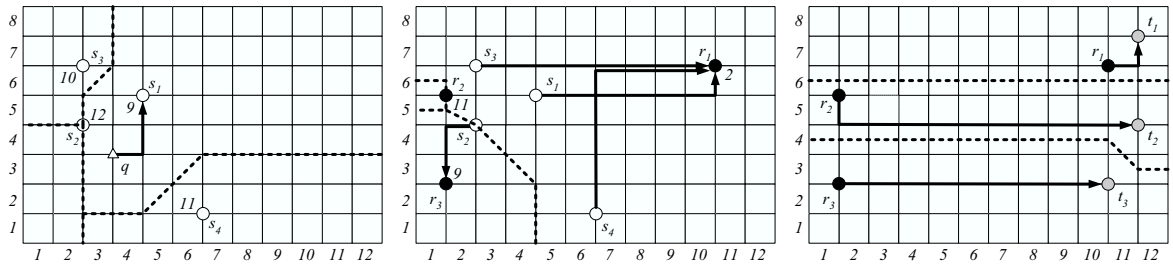


Figure 5: The OSR-Voronoi diagrams of a) the sequence $(white, black, grey)$, b) the sequence $(black, grey)$, and c) the sequence $(grey)$ using the white points in Figure 1

```
Function ComputeOSR(point q, sequence M)
    Returns route
1. V(p) = the cell containing q in the OSR-Voronoi diagram of M
2. p = the generator of V(p)
3. if |M| = 1 then
4.     return (p)
5. else
6.     return p ⊕ ComputeOSR(p, sfx(M, 1))
```

Figure 6: Computing OSR for query $Q(q, M)$

Figure 1, we require the OSR-Voronoi diagrams of sequences $(white, black, grey)$, $(black, grey)$, and $(grey)$. Notice that the OSR-Voronoi family of a sequence can also be used to answer OSR queries of any suffix of that sequence.

4.1 Query Processing

Here, we assume that we have already built the OSR-Voronoi family of a given sequence M. Subsequently, we employ this set of diagrams to answer the OSR query $Q(q, M)$. Section 4.2 shows how we recursively build all these diagrams.

We describe our query processing approach using the example of Figure 1. Notice that here the sequence is $(white, black, grey)$ and the distance function $D(.,.)$ is Manhattan distance. We have already built and loaded the OSR-Voronoi diagrams of sequences $(white, black, grey)$, $(black, grey)$, and $(grey)$ illustrated in Figures 5a, 5b, and 5c, respectively. Notice that the last diagram is the standard Voronoi diagram on grey points as the OSR of a point q which follows the sequence $(grey)$ is simply the closest grey point to q (see Lemma 1 for $i = m - 1$). First, we locate the cell containing q in the OSR-Voronoi diagram of $(white, black, grey)$. Following Theorem 1, our OSR starts with s_1, the generator of this cell (see Figure 5a). The rest of the OSR is the same as the OSR of s_1 which follows $(black, grey)$. Hence, we next find r_1, the generator of the cell containing s_1 in the OSR-Voronoi diagram of $(black, grey)$ shown in Figure 5b. Finally, we find the closest grey point to r_1 as the generator of the cell containing r_1 in the OSR-Voronoi diagram of $(grey)$ (see Figure 5c). The OSR of point q which follows sequence $(white, black, grey)$ is (s_1, r_1, t_1) built using the three subsequently retrieved points. Figure 6 shows the pseudo-code of our OSR query processing approach as a *recursive* function *ComputeOSR(point, sequence)*.

4.2 Data Structures

The OSR-Voronoi diagrams are instances of AW-Voronoi diagrams whose weights are constrained by geometry. Therefore, we build them using any algorithm for building AW-Voronoi diagrams. The algorithm proposed by Karavelas and Yvinec [4] computes the diagram of n points in $O(n \log n)$ steps. Our approach utilizes the OSR-Voronoi diagrams of all suffixes of a sequence M to find the OSRs that follow M. Therefore, to answer queries with sequence $(white, black, grey)$ in Figure 1, we require the OSR-Voronoi diagrams of this sequence together with those of sequences $(black, grey)$ and $(grey)$. In general, for a given sequence M, we require $|M|$ OSR-Voronoi diagrams of all suffixes of M to answer $Q(q, M)$. Hence, the pre-computation phase takes $O(\sum_{i=1}^{|M|} |U_{M_i}| \log |U_{M_i}|)$ computation steps.

We incrementally build the OSR-Voronoi family of $(white, black, grey)$ as follows. First, we build the OSR-Voronoi diagram of $(grey)$ shown in Figure 5c which is simply the standard Voronoi diagram of the set of grey points $T = \{t_1, t_2, t_3\}$ with respect to Manhattan distance [6]. Second, for each black point r_i we find the generator grey point of the cell containing r_i in this OSR-Voronoi diagram. In Figure 5c, each black point is connected to its corresponding grey generator by a solid line. We assign the distance between the point r_i and its corresponding grey point to the weight of r_i. For instance, we assign $w(r_1) = 2$. Third, we build the OSR-Voronoi diagram of $(black, grey)$ using the set of black points $R = \{r_1, r_2, r_3\}$ and their weights (see Figure 5b). Similar to the previous step, for each white point s_i we locate r_x, the generator of the cell which contains s_i in the diagram built in this step and assign $D(s_i, r_x) + w(r_x)$ to the weight of s_i. For example, the weight of s_1 is $7 + 2 = 9$. Finally, we build the OSR-Voronoi diagram of $(white, black, grey)$ using the set of white points $S = \{s_1, s_2, s_3, s_4\}$ with respect to their corresponding weights (see Figure 5a). Figure 7 shows the pseudo-code of the above algorithm.

Once we computed the OSR-Voronoi family of a sequence, we require to store the boundaries of the Voronoi cells corresponding to each OSR-Voronoi diagram of the family. However, with Euclidean distance, these boundaries consist of curved edges and hence cannot be stored as simple polygons. This problem can be

```
Function BuildOSRVoronois(sequence M = (M_1, ..., M_m))
    Returns OSR-Voronoi diagrams
1.  OSRV((M_m)) = the standard Voronoi diagram of U_{M_m}
2.  for each p in U_{M_m}
3.      w(p) = 0
4.  i = m − 1
5.  while i > 0 do {
6.      M' = sfx(M, i)
7.      for each p in U_{M_i} {
8.          V(p') = the cell containing p in the OSRV(M')
9.          p' = the generator of V(p')
10.         w(p) = D(p, p') + w(p')
        }
11.     OSRV(sfx(M, i − 1)) = the AW-Voronoi diagram of U_{M_i}
12.     i = i − 1
    }
13. return OSRV's
```

Figure 7: Incrementally building the data structures required to answer the OSR query $Q(q, M)$

solved in the two following ways. Each solution guarantees that given a starting point q, we can easily determine the cell containing q.

1. The cells are approximated by convex polygons and consequently these polygons are stored. The polygons are also indexed by a spatial index structure such as R-tree. Therefore, the cell containing the starting point q is easily retrieved by a spatial `contains()` query. This solution introduces an error in OSR query processing proportional to the approximation error.

2. Instead of storing the cells, the neighborhood graph of the cell generators is stored (similar to Delaunay graph for the general Voronoi diagrams [6]). Vertices of the graph are the generators of the OSR-Voronoi diagram of M. There is a graph edge connecting p and p' iff $V(p)$ and $V(p')$ have common boundaries. Now, to determine the inclusion of a starting point q in a cell, we start traversing the graph from any vertex p. We iteratively visit the vertex p', the neighbor of the current vertex p which minimizes $D(q, p') + w(p')$ among p's neighbors and p itself. When no such a vertex is found, q is inside the Voronoi cell of the current vertex p.

5 Performance Evaluation

We conducted several experiments to evaluate the performance of our proposed approach. We compared the query response time of our Voronoi-based approach (denoted by OSRV) with that of our R-LORD algorithm proposed in [8]. We also evaluated our new OSRV approach with respect to its overall time required to build the OSR-Voronoi family of a given query. We evaluated OSRV by investigating the effect of the following two parameters on its performance: 1) size of sequence M in $Q(q, M)$ (i.e., number of points in the optimal route), and 2) cardinality of the datasets (i.e., $\sum_{i=1}^{n} |U_i|$).

We used a real-world dataset which is obtained from the U.S. Geological Survey (USGS) and consists of $950,000$ locations of different businesses (e.g., schools) in the entire country. Table 2 shows the characteristics of this dataset. In our experiments, we randomly selected sets of $40K$, $70K$, $250K$ and $500K$, and $950K$ points from this dataset. We used CGAL's implementation of [4] to implement the OSR-Voronoi diagrams [1]. We ran 1000 OSR queries from random starting points on a DELL Precision 470 with Xeon 3.2 GHz processor and 3GB of RAM.

Points	*Size*	*Points*	*Size*
Hospital	5,314	Building	15,127
Summit	69,498	Cemetery	109,557
Church	127,949	School	139,523
Populated place	167,203	Institution	319,751

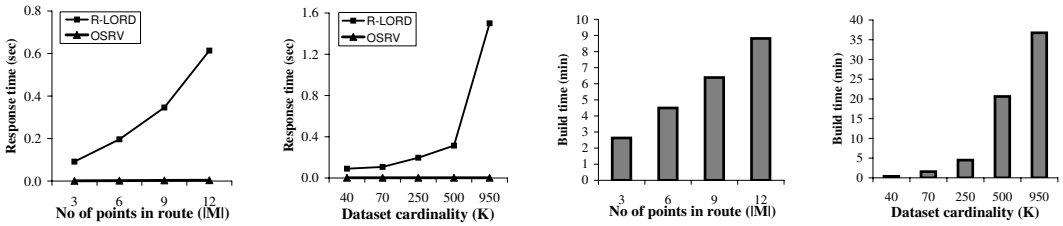
Table 2: USGS dataset

In the first set of experiments, we study the performance of OSRV in terms of query time. Figure 8a shows the query response time for our OSRV and R-LORD approaches when the number of points in optimal route (i.e., $|M|$) varies from 3 to 12. All the required diagrams fit in main memory, so the reported time represents CPU time. While the figure depicts the experimental results on 250K USGS dataset, the trend is the same with those of our other datasets with different cardinalities. Figure shows that OSRV always outperforms R-LORD. As expected, R-LORD performs slower with the bigger sequences as it requires more range queries on the R-tree. In contrast, OSR query with OSRV has almost no cost in time. OSRV's response time is unnoticeably increasing as OSRV consists of a sequence of only $|M|$ point locations on our efficient AW-Voronoi diagrams.

Figure 8b illustrates the result of our second set of experiments on the impact of the cardinality of the dataset on the efficiency of OSRV. We varied the cardinality of our real datasets from $40K$ to $950K$ and ran OSR queries of sequence size $|M| = 6$. Figure 8b shows that OSRV's performance is not much affected by the dataset size and verifies the scalability of OSRV. This is while the processing time of R-LORD moderately increases for larger datasets. For example, R-LORD's query response time is 0.09 seconds for $40,000$ points, and it increases by a factor of 16 when the number of points increases to $950,000$ by a factor of 24. Consequently, OSRV is the superior approach although R-LORD also exhibits a reasonable performance for large datasets.

The last experiment aims to measure the time required to build the diagrams in the OSR-Voronoi family of a given sequence. Note that this is a batch process which is performed off-line. Figure 8c illustrates the average build time for different sizes of the sequence M for $250K$ dataset. Moreover, Figure 8d depicts the result of the same experiment for different dataset cardinalities when the sequence size is 6. Figure 8c shows that it takes only 9 minutes to build 12 OSR-Voronoi diagrams needed for the OSR query of a sequence of size 12. However, Figure 8d shows that despite its excellent performance for small datasets, OSRV's procedure for building the OSR-Voronoi family of a sequence of size 6 takes about 36 minutes for the $950K$ dataset. This is still a reasonable performance for an off-line process.

[1] http://www.cgal.org/



a) Query cost vs. sequence size (250K) b) Query cost vs. cardinality ($|M| = 6$) c) Build cost vs. sequence size (250K) d) Build cost vs. cardinality ($|M| = 6$)

Figure 8: Experimental results on the performance of OSRV

We believe that upgrading main-memory AW-Voronoi diagrams to their secondary-memory versions will significantly improve OSRV's off-line performance for large datasets.

6 Related Work

In an independent research, Li et al. [5] studied Trip Planning Queries (TPQ), a class of queries similar to our OSR query. With a TPQ, the user specifies a subset (not a sequence) of location types R and asks for the optimal route from her starting location to a specified destination which passes through at least one database point of each type in R. In particular, TPQ eliminates the sequence order of OSR to define a new query. Consequently, finding accurate solutions to TPQ becomes NP-hard as the size of candidate space significantly grows. The paper proposes several approximation methods to provide near-optimal answers to TPQs. In theory, OSR algorithms are able to address address TPQ queries. This can be done by running any OSR algorithm $|R|!$ times, each time using a permutation of point types in R as the sequence M and returning the optimal route among all the resulted routes. This exponentially complex solution is inefficient in practice. The approximation algorithms of [5] are designed to handle the exponential growth of the problem's search space.

Terrovitis et al. [9] addressed k-stops shortest path problem for spatial databases. The problem seeks for the optimal path from a starting location to a given destination which passes through exactly k intermediate points of the location database. The k-stops problem is a specialized case of OSR queries. To be specific, OSR reduces a k-stops problem to query $Q(q, M)$ where q is the starting location, $M = (1, \ldots, 1)$, $|M| = k$, and U_1 is the single given point set representing the location database. The destination is considered by the variation of the above OSR query described in Section 2.1. A major drawback is that [9] only considers Euclidean space.

7 Conclusions and Future work

We studied the problem space of the OSR query in vector spaces. We exploited the geometric properties of the solution space and identified the AW-Voronoi diagrams as its corresponding geometric representation. In particular, we proved that given a sequence M the locus of all points with a common optimal sequenced route R is the cell of the first point of R in a AW-Voronoi diagram whose weights depend on the sequence M. Based on our theoretical findings, we proposed a novel OSR query processing approach using a family of pre-computed AW-Voronoi diagrams. Through experiments with a real-world dataset, we showed that our approach significantly outperforms the previous solutions to OSR query in terms of response time.

While in this paper we adapted AW-Voronoi diagrams for OSR queries, we believe that these diagrams can also be used to answer a variation of OSR query where the sequence is not important (TPQ problem [5]). An orthogonal problem is studying efficient algorithms to store the AW-Voronoi diagrams in secondary storage.

References

[1] Y. Du, D. Zhang, and T. Xia. The Optimal-Location Query. In *Proceedings of SSTD'05*, pages 163–180, 2005.

[2] C. du Mouza, P. Rigaux, and M. Scholl. Efficient Evaluation of parameterized pattern queries. In *Proceedings of CIKM'05*, pages 728–735. ACM, 2005.

[3] M. Hadjieleftheriou, G. Kollios, P. Bakalov, and V. J. Tsotras. Complex Spatio-Temporal Pattern Queries. In *Proceedings of VLDB'05*, pages 877–888, 2005.

[4] M. I. Karavelas and M. Yvinec. Dynamic Additively Weighted Voronoi Diagrams in 2d. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA'02)*, pages 586–598, London, UK, 2002. Springer-Verlag.

[5] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng. On Trip Planning Queries in Spatial Databases. In *Proceedings of SSTD'05*, pages 273–290. Springer, August 2005.

[6] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations, Concepts and Applications of Voronoi Diagrams*. John Wiley and Sons Ltd., 2nd edition, 2000.

[7] Y. Ostrovksy-Berman. Computing Transportation Voronoi Diagrams in Optimal Time. In *Proceedings of the 21st European Workshop on Computational Geometry (EWCG'05)*, pages 159–162, 2005.

[8] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The Optimal Sequenced Route. *The VLDB Journal*, 2006. To appear.

[9] M. Terrovitis, S. Bakiras, D. Papadias, and K. Mouratidis. Constrained Shortest Path Computation. In *Proceedings of SSTD'05*, pages 181–199. Springer, August 2005.

On Construction of Holistic Synopses under the Duplicate Semantics of Streaming Queries

David Toman

D. R. Cheriton School of Computer Science
University of Waterloo, Canada
david@uwaterloo.ca

Abstract

Transaction-time temporal databases and query languages provide a solid framework for analyzing the properties of queries over data streams. In this paper we focus on issues connected with the construction of space-bounded synopses that enable answering of continuous queries over unbounded data streams. We link the problem to the problem of query-driven data expiration in append-only temporal databases and study space bounds on synopses that are sufficient and necessary for query answering under duplicate semantics.

1 Introduction

A considerable effort to understand many aspects of query processing over streaming data—data that is arriving in fragments over time—has been the topic of research in the last several years, see e.g., [2, 3, 4, 8, 9, 10, 12, 13], and many others. These efforts have mainly focused on efficient processing of *continuous queries*—queries evaluated continuously as more data is arriving on the stream—over unbounded data streams. A *key* component of such solutions is the construction of *synopses*—data summaries that allow execution of continuous queries without the need to buffer or otherwise store the whole history of the data stream.

The goal of this paper is to show that certain requirements, often rather desirable in such systems—such as the use of SQL-style duplicate semantics while maintaining reasonable bounds on the size of the summary data—are not possible to achieve. The paper then shows how acceptable results can be achieved by carefully limiting the expressive power of streaming query languages in which continuous queries are formulated.

As we desire to derive bounds as strong as possible, we adopt a *holistic* approach to the construction of synopses for continuous queries. Unlike other approaches that construct synopses on a per-physical-operator basis (e.g., for the so called symmetric joins, etc., [2]), we develop techniques that tailor the synopsis to the complete continuous query—hence the use of the term *holistic synopsis*.

Many of the techniques presented in this paper can be traced to approaches designed to allow efficient data expiration in transaction-time temporal databases [15]. The novel contributions of this paper are mainly concerned with the use of *duplicate semantics* for queries and can be summarized as follows:

1. We show that adopting an SQL-style duplicate semantics for SQL-like streaming queries makes construction of bounded synopses impossible; in the general case, the synopses may have to grow at least linearly with the stream length.

2. We show that for certain limited languages, this growth can be tamed to a logarithmic factor; that factor, however, cannot be avoided.

Note that the negative results presented in this paper are based on information-theoretic properties of queries and thus cannot be improved upon by more sophisticated algorithms without resorting to approximations. We also contrast these results with similar results obtained for the set semantics of the same languages where constant bounds in the length of the stream can be obtained. In addition to the technical results, the paper provides a strong parallel between

Proceedings of the third Workshop on STDBM
Seoul, Korea, September 11, 2006

techniques developed for transaction-time temporal databases [7], data expiration [15], and approaches to efficient evaluation of streaming queries.

The rest of the paper is organized as follows: Section 2 provides the basic definitions and shows the links between streaming queries and temporal databases. Section 3 shows that in general, duplicate semantics leads at least to a logarithmic lower bound on the size of the synopses, measured in the length of the data stream; it also identifies cases in which allowing duplicates in the data model leads to linear lower bounds on the size of the synopses needed to answer a continuous query. Section 4 shows fragments of query languages for which a logarithmic bound can be achieved. We conclude with identifying open issues in the area in Section 5.

2 Background

We first review the relevant definitions in the area of transaction-time temporal databases and link them to querying data streams. The presentation in this section is based on a chapter on data expiration [15] with terminology suitably modified to data streams.

2.1 Temporal Queries for Data Streams

We first formalize the notion of data stream as follows.

Definition 2.1 (Data Stream/History) Let σ be a relational schema. A *data stream* is a sequence

$$S = \langle S_0, S_1, \ldots, S_t, \ldots \rangle$$

where each S_t is a multiset of tuples conforming to σ that have arrived in S at time t. We call the multisets S_t *states of* S *at* t. We assume discrete integer-like time with time instants drawn from a linearly ordered set and allow multiple tuples to arrive at the same time instant. The data values forming the tuples belong to the domain of *uninterpreted constants*; the data domain is equipped with equality only. At any particular finite time t, we have access only to a *finite prefix* of S of the form

$$S(t) = \langle S_0, S_1, \ldots, S_t \rangle$$

We use $\mathbf{T}(t)$ and $\mathbf{D}(t)$ to denote the *active temporal* and *data* domains of a stream (prefix), respectively. Note that the active domains change with time as new data arrives on the stream. The active temporal domain $\mathbf{T}(t)$ is, in our setting, just the set $\{0, \ldots, t\}$; the definition, however, allows to use timestamps from any linearly ordered set.

Without loss of generality, we present our results for a single data stream. However, the results immediately extend to multiple streams, e.g., by coding multiple streams by values of a distinguished attribute. This formalization shows that data stream is just a variant name for an *append-only* temporal database (often called a *transaction-time temporal database*). This observation allows us to use off-the shelf temporal query languages to query data streams. In this paper we use *two-sorted first-order logic* (2-FOL) to query such streams:

Definition 2.2 (Streaming/Temporal Queries) Let S be a data stream. The syntax of *first-order streaming queries* over S is given by the following grammar.

$$\begin{array}{rcl} Q & ::= & S(t, x_1, \ldots, x_k) \\ & | & x_i = x_j \mid t_i < t_j \mid t_i = t_j \mid t_i > t_j \\ & | & Q \wedge Q \mid \exists x_i.Q \mid \exists t_i.Q \mid \varepsilon\, Q \\ & | & Q \vee Q \mid Q \wedge \neg Q \end{array}$$

The $\varepsilon\, Q$ subformula stands for *duplicate elimination*. In addition, we assume that the queries are *range restricted*; this is enforced by requiring the usual restrictions on the occurrences of equalities and inequalities and the variable-compatibility conditions for disjunctions ($\vee$) and negations ($\wedge\neg$). The semantics of the queries is defined using the usual Tarskian-style satisfaction relation extended to account for duplication; we write

$$S(t), \theta, n \models Q$$

to stand for "*the substitution/tuple* θ *is an answer to* Q *with* n *duplicates, when evaluated over* $S(t)$*, a prefix of* S". The full semantics of the above language is given in Figure 1. In particular, the semantics specifies how duplicates are handled in queries[1]. An *answer* to a query at time t is the multiset

$$S_t^Q := \{\underbrace{\theta, \ldots, \theta}_{n} \mid S(t), \theta, n \models Q\}.$$

Note that the value n must be unique for a given tuple θ, i.e., $\theta(x)$ functionally determines n. To simplify the definition, we assume that substitutions not present in an answer have 0 duplicates.

Again, the query language in Definition 2.2 is just a temporal query language when the stream is regarded as finite prefix of a database history. Note also, that

[1] We utilize a SQL-style definition of duplicates, i.e., a product for conjunctions, sum for disjunction (union) and existential quantification (projection), and difference for range-restricted negation (set difference).

$$
\begin{aligned}
S(t), \theta, n &\models S(t, x_1, \ldots, x_k) && \text{if } \langle \theta(x_1), \ldots, \theta(x_k) \rangle \in S_{\theta(t)} \text{ duplicated } n \text{ times} \\
S(t), \theta, 1 &\models x_i = x_j && \text{if } \theta(x_i) = \theta(x_j) \\
S(t), \theta, 1 &\models t_i < t_j && \text{if } \theta(t_i) < \theta(t_j) \\
S(t), \theta_1 \circ \theta_2, m \cdot n &\models Q_1 \wedge Q_2 && \text{if } S(t), \theta_1, m \models Q_1 \text{ and } S(t), \theta_2, n \models Q_2 \\
S(t), \theta, \textstyle\sum_{v \in \mathbf{D}(t)} n_v &\models \exists x.Q && \text{if } S(t), \theta[v/x], n_v \models Q \\
S(t), \theta, \textstyle\sum_{s \in \mathbf{T}(t)} n_s &\models \exists t.Q && \text{if } S(t), \theta[s/t], n_s \models Q \\
S(t), \theta, 1 &\models \varepsilon\, Q && \text{if } S(t), \theta, n \models Q \\
S(t), \theta, n + m &\models Q_1 \vee Q_2 && \text{if } S(t), \theta, m \models Q_1 \text{ and } S(t), \theta, n \models Q_2 \\
S(t), \theta, \max(0, m - n) &\models Q_1 \wedge \neg Q_2 && \text{if } S(t), \theta, m \models Q_1 \text{ and } S(t), \theta, n \models Q_2
\end{aligned}
$$

Figure 1: SQL-style Duplicate Semantics for Streaming Queries.

the semantics is defined with respect to the finite portion of the data stream; answering queries over *all potential extensions* of a stream has been shown undecidable for any reasonably powerful query language [6], for detailed discussion of this phenomenon see [7].

Definition 2.3 (Continuous Query Answer) Let Q be a query without free temporal variables. An *answer to a continuous query* specified by Q is defined as a stream

$$S^Q = \langle S_0^Q, S_1^Q, \ldots, S_i^Q, \ldots \rangle$$

where S_t^Q is the answer to Q over the stream prefix $S(t)$ as defined by the semantics in Figure 1.

The restriction to queries without free temporal variables is needed to ensure that results of a query form a proper data stream. This arrangement allows for compositionally, possibility of view definitions, etc. Note also that the representation of duplicates in *binary* (i.e., using counts to represent the numbers of duplicates) is a more compact representation than explicitly duplicating the tuples. Thus all lower bounds derived in this paper also hold for the SQL-style representation in unary (i.e., by explicit replication of the tuples in question).

2.2 Holistic Synopses and Data Expiration

For continuous queries, it is often not feasible to store the whole data stream in computer storage. Therefore, streaming systems use summaries called *synopses* to remember the parts of the data stream that are necessary to generate subsequent answers to continuous queries.

Definition 2.4 (Holistic Synopsis) Let Q be a query over S. A *holistic synopsis* for Q is a triple $(\emptyset, \Delta, \Gamma)$ that satisfies the following property:

$$Q\langle S_0, \ldots, S_t \rangle = \Gamma(\Delta(S_t, \Delta(S_{t-1}, \Delta(\ldots \Delta(S_0, \emptyset)))))$$

for any prefix $\langle S_0, \ldots, S_t \rangle$ of S. In addition, we require that the triple $(\emptyset, \Delta, \Gamma)$ can be effectively constructed from Q.

The first two components define the actual *holistic synopsis* for Q as a self-maintainable materialized view of S: the $\emptyset$ component tells us what the contents of this view is in the beginning and the Δ component tells us how to update the view when more data arrives in S. The last component, Γ now generates the answers to Q only accessing the information in the view. Note that the definition does not specify what data model the view uses nor what query languages are used for the three components of the synopsis. Note that this definition is essentially the same as the definition of a *data expiration operator* for transaction-time temporal databases [15].

How do we compare Holistic Synopses?

Intuitively, we have replaced the complete prefix of S with a materialized view defined by the $\emptyset$ and Δ queries. Thus our aim is to minimize the size of the materialized view in terms of:

1. the length of the data stream S, $|\mathbf{T}(t)|$,
2. the number of distinct values in S, $|\mathbf{D}(t)|$, and
3. the size of Q.

The dependency on the length of the data stream is the most critical factor. Thus we call a synopsis *bounded* if it is bounded by a constant function in the length of the stream. We call a synopsis log-*bounded* if it is bounded by a function logarithmic in $|\mathbf{T}(t)|$.

For streaming query languages that use set semantics, results obtained for temporal databases can be applied:

Proposition 2.5 ([14]) *A bounded holistic synopsis exists for any two sorted first order streaming (2-FOL) query* under set semantics.

The above theorem shows that for queries under set semantics, bounded synopses exist for rather powerful query languages, in particular for the language introduced in Definition 2.2. The rest of the paper argues that bounded synopses do not exist when duplicate semantics is used *for the same language* and that log-bounded synopses are the best we can hope for in various fragments of the first-order language under duplicate semantics.

3 Lower Bounds

Now we are ready to provide simple lower bounds that show why the use of unrestricted duplicate semantics for streaming queries may be expensive and, in certain cases, not feasible at all. An $\Omega(\log|\mathbf{T}(t)|)$ lower bound has been observed for queries with the counting aggregate [14]. Similar query, e.g.,

$$(\exists t.S(t,a)) \wedge \neg(\exists t.S(t,b)),$$

that expresses the fact that there were more a's than b's in the stream S, for a and b distinct constants, yields such bound for queries with duplicates. It is easy to see using the pigeon-hole principle that a synopsis for this query needs $\Theta(\log|\mathbf{T}(t)|)$ bits.

However, it turns out that log-bounded synopses are not sufficient for first-order queries with duplicate semantics.

Theorem 3.1 *There is a first-order query for which any synopsis is bounded from below by* $\Omega(|\mathbf{T}(t)|)$.

Proof (sketch): Consider the query

$$\varepsilon\exists t_1,t_2.t_1<t_2 \wedge \neg((\exists x.S(t_1,x)) \wedge \neg(\exists x.S(t_2,x))) \\ \wedge\neg((\exists x.S(t_2,x)) \wedge \neg(\exists x.S(t_1,x)))$$

The query expresses the condition *two states of S contain the same number of tuples*. Now consider a prefix of a data stream $S(t)$ such that S_i contains the value a duplicated m_i times, $m_i \neq m_j$ for $0 \leq i \neq j \leq t$. To be able to answer the above query when the stream is extended by the state S_{n+1} we need at least a set of values $\{m_0,\ldots,m_t\}$. To represent this set we need at least

$$\sum_{i=0}^{t}\log(m_i) \geq t\cdot\log(\min\{m_0,\ldots,m_t\}) \in \Omega(|\mathbf{T}(t)|)$$

bits.

The unfortunate consequence of this lower bound is that, in general, the best synopsis for first-order queries under duplicate semantics are essentially as big as the original data stream (and thus we may be better off just storing the stream itself).

4 Upper Bounds

We now investigate cases in which logarithmic bounds on the size of holistic synopses can be obtained. To this end, we need to restrict the streaming query languages. We consider two different fragments of first-order logic:

1. positive first-order queries, and
2. temporal logic queries.

In both cases our aim is to make the proof of Theorem 3.1 inapplicable. In the first case by disallowing negation and in the second by disallowing multiple temporal contexts to exist at the same time.

4.1 First-order Temporal Logic Queries

We start with the case of the past fragment of the *first-order temporal logic* (FOTL), a logic based on implicit access to the temporal attribute of tuples using *modal operators*. This way the number of coexisting temporal contexts is limited while still commanding a sufficient expressive power[2]. The syntax of the language is defined as follows:

Definition 4.1 (First-order Temporal Logic) Let S be a data stream. The syntax of *FOTL queries* over S is given by the following grammar.

$$\begin{array}{rcl} Q & ::= & S(x_1,\ldots,x_k) \mid x_i = x_j \\ & \mid & Q \wedge Q \mid \exists x_i.Q \mid \varepsilon\, Q \\ & \mid & Q \vee Q \mid Q \wedge \neg Q \\ & \mid & Q \textbf{ since } Q \mid \bullet Q \end{array}$$

Note that formulas of FOTL do not use variables ranging over the temporal domain; handling of this aspect of the queries is *encapsulated* in the *temporal operators* **since** and $\bullet$ (previous time). Thus the semantics is now defined *with respect to an evaluation point* using a satisfaction relation

$$S(t),\theta,s,n \models Q$$

which, similarly to Definition 2.2, states that *the tuple θ is an answer to Q at time s with n duplicates in $S(t)$*. Note that the time point s may be different from t; this allows referring to past states of the data stream S in queries. The semantics of FOTL mimics that of 2-FOL introduced in Figure 1: all the standard first-order connectives and quantifiers are evaluated per state of S (using the so-called snapshot semantics [11]). The only addition are the rules for handling the temporal

[2]It has been shown, however, that FOTL is strictly less expressive fragment of 2-FOL, the language introduced in Definition 2.2 [1, 16].

operators; here we need to extend the standard definitions to handle *duplicates*. The semantics of the **since** operators (there are two variants to account for two ways to determine the number of duplicates) is defined as follows:

$S(t), \theta, s, \max_{s_2 \in \mathbf{T}} \sum_{s_1 \in \mathbf{T}(t)} m_{s_1} \models Q_1$ **since**$_1$ Q_2 if
$S(t), \theta, n_{s_2}, s_2 \models Q_2$ for some $s_2 < s, n_{s_2} > 0$ and
$S(t), \theta, m_{s_1}, s_1 \models Q_1$ for all $s_1 < s_2 \leq s, m_{s_1} > 0$

$S(t), \theta, s, \sum_{s_2 \in \mathbf{T}(t)} n_{s_2} \models Q_1$ **since**$_2$ Q_2 if
$S(t), \theta, n_{s_2}, s_2 \models Q_2$ for some $s_2 < s, n_{s_2} > 0$ and
$S(t), \theta, m_{s_1}, s_1 \models Q_1$ for all $s_1 < s_2 \leq s, m_{s_1} > 0$

For the previous time operator, $\bullet Q$, the duplicate semantics is defined as follows:

$S(t), \theta, s, n \models \bullet Q$ if $S(t), \theta, s-1, n \models Q$ and $s > 0$

An answer to a continuous query Q at time t is defined as the multiset

$$\{\underbrace{\theta, \ldots, \theta}_{n} \mid S(t), \theta, t, n \models Q\}.$$

The two variants, **since**$_1$ and **since**$_2$, differ in the way they handle duplicates: **since**$_1$ counts the maximal number of the answers θ satisfying Q_1 since Q_2 was true; **since**$_2$ counts the number of times the tuple θ satisfies Q_2 such that Q_1 was true since then[3].

Additional temporal connectives can be derived from the **since** and $\bullet$ operators; again, the handling of duplicates is the only concern here.

Example 4.2 The *sometime in the past* ($\blacklozenge$) connective can be defined as follows:

$$\blacklozenge_1 Q = \text{true } \textbf{since}_1 \ Q$$
$$\blacklozenge_2 Q = \text{true } \textbf{since}_2 \ Q$$

The two variants differ again in how duplication of results is defined: in the first case it indicates how far in the past was the earliest Q has been and the second case how many times Q has been true in the past[4]. Again additional connectives can be defined by combining the above two. For example, should we wish to know how far in the past the latest Q was we can write $(\neg Q)$ **since**$_1$ Q.

[3]Similar to the duplicate semantics for SQL, this is just a particular a way of choosing how many duplicates are in an answer to a particular query. The definitions above work correctly with the rest of the technical development in this paper. A comprehensive study of various possibilities to define duplication of tuples for temporal queries is beyond the scope of this paper.

[4]Assuming the constant *true* is not duplicated.

Note that one might be tempted to define the duplicate semantics or the *sometime in the past* ($\blacklozenge$) operator, e.g., as follows:

$S(t), \theta, t, n \models \blacklozenge Q$ if $S(t), \theta, s, n \models \blacklozenge Q$ for some $s < t$.

This definition would seemingly allow formulating the query "*were there two time instants with the same number of elements in the stream?*". However, it is important to see that such a definition is *incompatible* with the definition of duplicate semantics: it allows assigning two different duplicities to the same tuple—this is illegal under the duplicate semantics.

Synopses for FOTL

To define a synopsis for a given FOTL formula we use the following identities:

Lemma 4.3 Let S be a data stream and Q_1 and Q_2 FOTL queries. Then

$S(t), \theta, s, n+m \models Q_1$ **since**$_1$ Q_2 if
- $S(t), \theta, s-1, n \models Q_1$ **since**$_1$ Q_2 and
- $S(t), \theta, s, m \models Q_1$.

$S(t), \theta, s, m \models Q_1$ **since**$_1$ Q_2
- $S(t), \theta, s-1, n \models Q_2$,
- $S(t), \theta, s-1, l \not\models Q_1$ **since**$_1$ Q_2, and
- $S(t), \theta, s, m \models Q_1$.

$S(t), \theta, s, n+m \models Q_1$ **since**$_2$ Q_2 if
- $S(t), \theta, s-1, n \models Q_1$ **since**$_2$ Q_2,
- $S(t), \theta, s-1, m \models Q_2$, and
- $S(t), \theta, s, l \models Q_1$.

$S(t), \theta, s, m \models Q_1$ **since**$_2$ Q_2 if
- $S(t), \theta, s-1, n \not\models Q_1$ **since**$_2$ Q_2,
- $S(t), \theta, s-1, m \models Q_2$, and
- $S(t), \theta, s, l \models Q_1$.

$S(t), \theta, s, n \models Q_1$ **since**$_2$ Q_2 if
- $S(t), \theta, s-1, n \models Q_1$ **since**$_2$ Q_2,
- $S(t), \theta, s-1, m \not\models Q_2$, and
- $S(t), \theta, s, l \models Q_1$.

Proof (sketch): Immediate from the definitions.

With the help of these identities we can modify the approach to bounded checking of past FOTL constraints [5] as follows:

Definition 4.4 Let Q be a FOTL query and $\{\alpha_1, \ldots, \alpha_k\}$ all of its temporal subformulas (i.e., formulas rooted by a temporal operator). We define auxiliary views R^{α_i}, one for each of the subformulas α_i. The attributes of R^{α_i} correspond to the free variables in α_i with an additional distinguished *integer-valued* attribute n that accounts for the duplication of tuples.

The auxiliary views are initialized to an empty set each (this formally corresponds to the $\emptyset$ component of the synopsis) and then, whenever a new state of S arrives, the views are rematerialized using the rules in Lemma 4.3 (the Δ component). To evaluate the temporal subformulas in the preconditions of the rules we use the current and new instances of the auxiliary views (this imposes an ordering on the execution of the re-materializations). An answer to Q (the Γ component) is derived by executing Q after all its temporal subformulas have been replaced by the auxiliary views.

Theorem 4.5 *Let Q be a FOTL query. Then the instances of views constructed using Definition 4.4 form a $\log(|\mathbf{T}(t)|)$-bounded holistic synopsis for Q.*

Proof (sketch): Every time a new data state arrives on S, it is sufficient to access only the t-th and $(t-1)$st states of the auxiliary materialized views and the last state of S. The size of the auxiliary views depends on the size of the query (number of temporal subqueries), the size of the active data domain $|\mathbf{D}(t)|$ at time t, and $\log(|\mathbf{T}(t)|)$ due to the necessity to keep the counters counting the numbers of duplicates.

The above upper bound matches the lower bound from Section 3 as the query "*have there been more a values than b values in the stream S*" can be formulated in FOTL using the formula

$$(\blacklozenge_2 S(a)) \wedge \neg(\blacklozenge_2 S(b)).$$

Thus the above technique is (worst-case) optimal up to a constant factor (w.r.t. $|\mathbf{T}(t)|$).

Example 4.6 The auxiliary views for the query above are $R^{\blacklozenge_2 S(a)}(n)$ and $R^{\blacklozenge_2 S(b)}(n)$; note that both the views have single integer attribute as the original subqueries are closed formulas.

The Δ operator is defined as

$$R_t^{\blacklozenge_2 S(a)} := \{n + m \mid S(t), m, t \models S(a), \\ R(t), n, t-1 \models R^{\blacklozenge_2 S(a)}\}$$

where the stream R is the stream generated for the auxiliary relations. Similar definition is used for $R^{\blacklozenge_2 S(b)}(n)$. Note that as streams, the auxiliary relations in this example are 0-ary—conceptually they contain the *true* tuple duplicated an appropriate number of times; the actual binary representations therefore contain just one integer value each.

The Γ part is then defined as $R^{\blacklozenge_2 S(a)} \wedge \neg R^{\blacklozenge_2 S(b)}$.

4.2 Conjunctive and Positive Queries

Queries in FOTL provide a powerful way of querying data streams. They also generalize the so called *windowed* queries in a natural way. However, there is a mismatch between streaming query languages that use SQL-like syntax (with explicit access to the time instant attributes [2]) and FOTL. Indeed, [1, 16] show that the later are strictly less expressive. Also, the lower bound for 2-FOL in Section 3 compared to the upper bound for FOTL in Section 4 show, that this discrepancy cannot be repaired by adopting duplicate semantics.

In this section we therefore look on common sublanguages of 2-FOL, namely on conjunctive queries (CQ) and unions of conjunctive queries (UCQ). We show that, in these cases the technique developed for FOTL can be adopted to CQ and UCQ.

Definition 4.7 (Conjunctive Query) A *conjunctive query* is an expression of the form

$$\exists t_1, \ldots, t_k, \bar{y}.S(t_1, \bar{x_1}), \wedge \ldots \wedge S(t_k, \bar{x_k}) \wedge \phi \wedge \psi$$

where $\bar{x_i}$ are vectors of data variables, φ is a conjunction of equalities over the data variables and ψ a ordering condition over the temporal variables. The query can be potentially prefixed by duplicate elimination.

Note that we require the answers to CQ not to contain any free temporal variables in order for them to serve as continuous queries over data streams.

The construction of synopses for CQ proceeds in two steps:

1. First a given CQ is rewritten to an equivalent union of CQs, such that the condition ψ in each of the constructed queries imposes a linear order among the variables $t_1, \ldots, t_k$.
2. Second, each of the above queries is translated to FOTL and then the synopsis construction for FOTL is used.

This approach is supported by the following two lemmas:

Lemma 4.8 Let Q be a CQ of the form

$$\exists t_1, \ldots, t_k, \bar{y}.S(t_1, \bar{x}_1), \wedge \ldots \wedge S(t_k, \bar{x}_k) \wedge \phi \wedge \psi.$$

Then there is a finite set of CQ such that

- the (duplicate preserving) disjunction of these CQ is equivalent to the original CQ and
- the subformulas ψ in these queries impose a linear order on valuations of the variables $t_1, \ldots, t_k$.

Proof (sketch): Let Ψ be the set of all formulas that express linear orders over $t_1, \ldots, t_k$ consistent with ψ. This set is finite and due to the law of excluded middle, no two elements of Ψ can be made true by the same valuation. Thus the set of CQ

$$\{\exists t_1, \ldots, t_k, \bar{y}.S(t_1, \bar{x}_1), \wedge \ldots \wedge S(t_k, \bar{x}_k) \wedge \phi \wedge \varphi \mid \varphi \in \Psi\}$$

fulfills the requirements of the Lemma.

Thus, for each two variables, we have $t_i < t_j$, $t_i = t_j$, or $t_i > t_j$. This allows us to construct a FOTL formula by using the $\blacklozenge_2$ connective to simulate the inequalities. Hence the following Lemma:

Lemma 4.9 Let Q be a CQ of the form

$$\exists t_1, \ldots, t_k, \bar{y}.S(t_1, \bar{x}_1), \wedge \ldots \wedge S(t_k, \bar{x}_k) \wedge \phi \wedge \varphi.$$

in which φ imposes a linear order on the temporal variables $t_1, \ldots, t_k$. Then there is an equivalent formula in FOTL.

Proof (sketch): We replace the CQ by a FOTL query of the form

$$\begin{aligned}\exists \bar{y}.\blacklozenge(&S(\bar{x}_{i_1}) \wedge \ldots \wedge S(\bar{x}_{i_l}) \wedge \\ &\blacklozenge(S(\bar{x}_{j_1}) \wedge \ldots \wedge S(\bar{x}_{j_{l'}}) \wedge \ldots \\ &\quad \blacklozenge(S(\bar{x}_{k_1}) \wedge \ldots \wedge S(\bar{x}_{k_{l''}}) \wedge) \ldots))\end{aligned}$$

where the subscripts $i_1, \ldots, i_l$ refer to those conjuncts in the original query that are first in the linear ordering of temporal variables, $j_1, \ldots, j_{l'}$ to the second, and $k_1, \ldots, k_{l''}$ to the last, i.e., the linear order of the temporal variables was

$$t_{i_1} = \ldots = t_{i_l} > t_{j_1} = \ldots = t_{j_{l'}} > \ldots > t_{k_1} = \ldots = t_{k_{l''}}$$

in φ.

We finish the construction by applying the approach to construction of bounded synopses introduces in Section 4.1. The use of this approach for UCQ is immediate.

5 Conclusion

In this paper we have shown that duplicate semantics causes a severe difficulties in constructing bounded synopses for processing continuous queries. Indeed, even simple queries may require a synopsis (linearly) proportional to the length of the original stream—which defeats the usefulness of such synopsis. We have also developed restricted fragments of streaming queries that allow logarithmically-bounded synopses to be used and shown how such synopses can be constructed.

5.1 Future Directions of Research

There are many directions of research to pursue in this direction. Among these are:

Alternatives to standard duplicate semantics. In this paper we mainly considered the standard *SQL-style* approach of defining duplicate semantics of queries. However, beyond compatibility concerns, there is no principal reason why other numerical functions, such as the "min" and "max" functions, couldn't be used to define the duplicate semantic for conjunctions and disjunctions, respectively. The main open question is whether a plausible duplicate semantics for first-order queries to which Theorem 3.1 doesn't apply exists.

2-FOL queries with log-bounded synopses. We have shown two sublanguages of 2-FOL queries for which log-bounded synopses can be constructed. However, it is not clear whether it is possible to syntactically characterize those 2-FOL queries (possibly up to query equivalence) for which log-bounded synopses exist.

Aggregates. Another question relates to the possibility of introducing *aggregate functions* into the query language—again, the best lower bounds we know today are logarithmic in the length of the stream. However, the techniques proposed in this paper cannot cope with aggregates mainly as aggregate functions introduce new *domain elements*.

Possible and certain answers. Yet another direction of research is to study fragments of query languages for which the possible result semantics is viable.

Also, the focus of this paper was on developing techniques that allow *precise* answers to continuous queries to be computed. Another direction of research is to consider appropriate ways to *approximate* the answers and trade-offs between quality of the approximations and space needed for storing synopses. While there has been a large amount of work in this area, surveying the issues connected with approximate query answers is beyond the scope of this paper.

References

[1] Serge Abiteboul, Laurent Herr, and Jan Van den Bussche. Temporal Versus First-Order Logic to Query Temporal Databases. In *ACM Symposium on Principles of Database Systems*, pages 49–57, 1996.

[2] Arwind Arasu, Shivnath Babu, and Jennifer Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution, 2003.

[3] Ahmed Ayada and Jeffrey F. Naughton. Static Optimization of Conjunctive Queries with Sliding Windows Over Infinite Streams. In *ACM SIGMOD International Conference on Management of Data*, pages 419–430, 2004.

[4] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In *ACM Symposium on Principles of Database Systems*, pages 1–16, 2002.

[5] J. Chomicki. Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *ACM Transactions on Database Systems*, 20(2):149–186, June 1995.

[6] J. Chomicki and D. Niwinski. On the Feasibility of Checking Temporal Integrity Constraints. *Journal of Computer and System Sciences*, 51(3):523–535, December 1995.

[7] J. Chomicki and D. Toman. Temporal Databases. In M. Fischer, D. Gabbay, and L. Villa, editors, *Handbook of Temporal Reasoning in Artificial Intelligence*, pages 429–467. Elsevier *Foundations of Artificial Intelligence*, 2005.

[8] Lukasz Golab and M. Tamer Özsu. Processing sliding window multi-joins in continuous queries over data streams. In *International Conference on Very Large Data Bases (VLDB)*, pages 500–511, 2003.

[9] Jaewoo Kang, Jeffrey F. Naughton, and Stratis Viglas. Evaluating Window Joins over Unbounded Streams. In *International Conference on Data Engineering (ICDE)*, pages 341–352, 2003.

[10] Flip Korn, S. Muthukrishnan, and Yunyue Zhu. Checks and Balances: Monitoring Data Quality Problems in Network Traffic Databases. In *International Conference on Very Large Data Bases (VLDB)*, pages 536–547, 2003.

[11] Richard T. Snodgrass, I. Ahn, G. Ariav, D. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Kafer, N. Kline, K. Kulkarni, T. Y. C. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. A. Sripada. TSQL2 Language Specification. *SIGMOD Record*, 23(1):65–86, March 1994.

[12] Utkarsh Srivastava and Jennifer Widom. Memory-Limited Execution of Windowed Stream Joins. In *International Conference on Very Large Data Bases (VLDB)*, pages 324–335, 2004.

[13] Nesime Tatbul, Ugur Cetintemel, Stanley B. Zdonik, Mitch Cherniack, and Michael Stonebraker. Load Shedding in a Data Stream Manager. In *International Conference on Very Large Data Bases (VLDB)*, pages 309–320, 2003.

[14] David Toman. Expiration of Historical Databases. In *International Symposium on Temporal Representation and Reasoning*, pages 128–135. IEEE Press, 2001.

[15] David Toman. Logical Data Expiration. In Jan Chomicki, Gunter Saake, and Ron van der Meyden, editors, *Logics for Emerging Applications of Databases*, chapter 7, pages 203–238. Springer, 2003.

[16] David Toman and Damian Niwinski. First-Order Queries over Temporal Databases Inexpressible in Temporal Logic. In *International Conference on Extending Database Technology*, Avignon, France, 1996.

Mining Long, Sharable Patterns in Trajectories of Moving Objects

Gyozo Gidófalvi / Torben Bach Pedersen

Geomatic aps / Aalborg University
gyg@geomatic.dk / tbp@cs.aau.dk

Abstract

The efficient analysis of spatio–temporal data, generated by moving objects, is an essential requirement for intelligent location–based services. Spatio-temporal rules can be found by constructing spatio–temporal baskets, from which traditional association rule mining methods can discover spatio–temporal rules. When the items in the baskets are spatio–temporal identifiers and are derived from trajectories of moving objects, the discovered rules represent frequently travelled routes. For some applications, e.g., an intelligent ridesharing application, these frequent routes are only interesting if they are long and sharable, i.e., can potentially be shared by several users. This paper presents a database projection based method for efficiently extracting such long, sharable frequent routes. The method prunes the search space by making use of the minimum length and sharable requirements and avoids the generation of the exponential number of sub–routes of long routes. A SQL–based implementation is described, and experiments on real life data show the effectiveness of the method.

1 Introduction

In recent years Global Positioning Systems (GPS) have become increasingly available and accurate in mobile devices. As a result large amounts of spatio–temporal data is being generated by users of such mobile devices, referred to as *moving objects* in the following. Trajectories of moving objects, or trajectories for short, contain regularities or patterns. For example, a person tends to drive almost every weekday to work approximately at the same time using the same route. The benefits of finding such regularities or patterns is many–fold. First, such patterns can help the efficient management of trajectories. Second, they can be used to facilitate various Location–Based Services (LBS). One LBS example is an intelligent rideshare application, which finds sharable routes for a set of commuters and suggests rideshare possibilities to them, is considered. Such a rideshare application can be one possible solution to the ever increasing congestion problems of urban transportation networks.

Patterns in trajectories for an intelligent rideshare application are only interesting if those patterns are sharable by multiple commuters, are reoccurring frequently, and are worthwhile pursuing, i.e., are long enough for the savings to compensate for the coordination efforts. The discovery of Long, Sharable Patterns (LSP) in trajectories is difficult for several reasons. Patterns do not usually exist along the whole trajectory. As a example, consider two commuters A and B living in the same area of town, leaving for work approximately the same time, and working in the same part of town. Given the underlying road network and traffic conditions, for a given support threshold the middle part of the trips of the two commuters may be frequent, the initial and final parts may not. In recent work [4] a general problem transformation method, called *pivoting*, was proposed for the analysis of spatio–temporal data. Pivoting is the process of grouping a set of records based on a set of attributes and assigning the values of likely another set of attributes to groups or baskets. Pivoting applied to spatio–temporal data allows the construction of spatio–temporal baskets, which can be mined with traditional association rule mining algorithms. When the items in the baskets are spatio–temporal identifiers and are derived from trajectories, the discovered rules represent frequently travelled routes. While there exist several efficient association rule mining methods [6], the straight–forward application of these algorithms to spatio–temporal baskets representing trajectories is infeasible for two reasons. First, all sub–patterns of frequent patterns are also frequent, but not interesting, as longer patterns are preferred. Second, the support criterion used in association rule mining algorithms is inadequate for a rideshare application, i.e., a fre-

Proceedings of the third Workshop on STDBM
Seoul, Korea, September 11, 2006

quent itemset representing a frequent trajectory pattern, may be supported by a single commuter on many occasions and hence presents no rideshare opportunity.

In this paper, to overcome the above difficulties of finding LSPs in trajectories, a novel method is given. According to a new support criterion, the proposed method first efficiently filters the trajectories to contain only sub-trajectories that are frequent. Next, it removes trajectories that do not meet the minimum length criterion. Then it alternates two steps until there are undiscovered LSPs. The first step entails the discovery of a LSP. The second step entails the filtering of trajectories by the previously discovered pattern. An advantage of the proposed method is the ease of implementation in commercial Relational Database Management Systems (RDBMSes). To demonstrate this, a SQL–based implementation is described. The effectiveness of the method is demonstrated on the publicly available INFATI data, which contains trajectories of cars driving on a road network.

The herein presented work is novel in several aspects. It is the first to consider the problem of mining LSPs in trajectories. It describes a novel transformation, and the relationship between the problem of mining LSPs in trajectories and mining frequent itemsets. Finally, it describes an effective method with a simple SQL–implementation to mine such LSPs in trajectories.

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the transformation, the use of the framework in frequent itemset mining, and formally defines the task of mining LSPs in trajectories. he proposed algorithm and a SQL–based implementation for mining such patterns is described in Section 4. Section 5 presents experimental results. Finally Section 6 concludes and points to future research.

2 Related work

Frequent pattern mining is a core field in data mining research. Since the first solution to the problem of frequent itemset mining [1, 2], various specialized in–memory data structures have been proposed to improve the mining efficiency, see [6] for an overview. It has been recognized that the set of all frequent itemsets is too large for analytical purposes and the information they contain is redundant. To remedy this, two modification to the task have been proposed: mining of Closed Frequent Itemsets (CFI) and mining of maximal frequent itemsets. A frequent itemset X is *closed* if no itemset Y exists with the same support as X such that $X \subset Y$. A frequent itemset X is *maximal* if no frequent itemset Y exists such that $X \subset Y$. Prominent methods that efficiently exploit these modifications to the problem are MAFIA, GenMax, CLOSET(+), and CHARM [6]. Later in the paper, a relationship between the problems of mining LSPs in trajectories and mining CFIs are described. While CFI mining methods can be modified to find the desired solution that meets the *sharable* criterion, they employ complex data structures and their implementation is quite involved; hence their augmentation is difficult. In particular, a projection-based CFI mining algorithm that employs an in–memory FP-tree to represent itemsets, would need to be modified at every node to maintain a set of distinct objects at that have transactions associated with them that support the itemset that is represented by the node. In comparison, the herein presented method –building on work presented in [11]– exploits the power of commercial RDBMSs, yielding a simple, but effective solution.

Since trajectories are temporally ordered sequences of locations, sequential pattern mining [3] naturally comes to mind. However, a straight forward interpretation of trips as transactions and application of a state–of–the–art closed frequent sequential pattern mining algorithm [15] does not yield the desired solution, since in this case sequences of frequent subtrajectories would be found. Furthermore, since the trajectories can contain hundreds of items, closedness checking of frequent itemsets even for prominent methods would be computationally expensive. Interpreting single elements of trajectories as transactions and applying closed sequential pattern mining could find frequent sub–trajectories. However a number of problems arise. First, to meet the sharable criterion, the in–memory data structures would need similar, nontrivial augmentation as described above. Second, since patterns in trajectories could be extremely long, even state–of–the–art sequential mining methods [12, 15] would have a difficulties handling patterns of such lengths. Third, patterns in trajectories repeat themselves, which cannot be handled by traditional sequential pattern mining algorithms. The extraction of spatio–temporal periodic patterns from trajectories is studied in [9], where a bottom–up, level–wise, and a faster top–down mining algorithm is presented. Although the technique is effective, the patterns found are within the trajectory of a single moving object. In comparison, the herein presented method effectively discovers long, sharable, periodic patterns.

Moving objects databases are particular cases of spatio–temporal databases that represent and manage changes related to the movement of objects. A necessary component to such databases are specialized spatio–temporal indices such as the Spatio–Temporal R–tree (STR–tree) and Trajectory–Bundle tree (TB–tree) [7]. An STR-tree organizes line segments of a trajectory according to both their spatial properties and the trajectories they belong to, while a TB–tree only preserves trajectories. If trajectories are projected to the time–of–day domain, STR–tree index values on the projected trajectories could be used as an alternative representation of trajectories. While this ap-

proach would reduce the size of the problem of mining LSPs in trajectories, it would not solve it. In comparison, the herein presented method solves the problem of mining LSPs in trajectories, which is orthogonal, but not unrelated to indexing of trajectories.

In [13] a way to effectively retrieve trajectories in the presence of noise is presented. Similarity functions, based on the longest sharable subsequence, are defined, facilitating an intuitive notion of similarity between trajectories. While such an efficient similarity search between the trajectories will discover similar trajectories, the usefulness of this similarity in terms of length and support would not be explicit. In comparison, there herein proposed method returns only patterns that meet the user–specified support and length constraints. Furthermore, the trajectory patterns returned by our method are explicit, as opposed to the only implicit patterns contained in similar trajectories.

3 Long, sharable patterns in trajectories

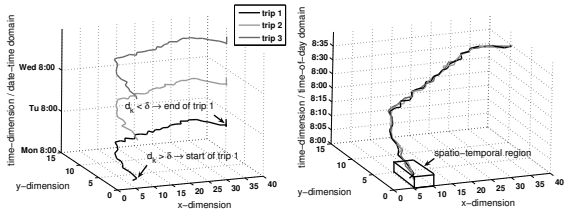
The following section describes a novel transformation of raw trajectories. This transformation allows (1) the formulation of the problem of mining LSPs in trajectories in a framework similar to that used in frequent itemset mining, (2) to establish a relationship between the two problems.

3.1 From trajectories to transactions

The proposed transformation of raw trajectories consists of three steps: identification of trips, projection of the temporal dimension, and spatio–temporal region substitution. It is assumed that locations of moving objects are sampled over a long history. That is, a raw trajectory is a long sequence of (x, y, t) measurements at regular time intervals.

Identification of trips

A trip is a temporally consecutive set or sequence of measurements such that for any measurement m_i in the sequence, the sum of spatial displacement during the k measurements immediately following m_i, denoted d_k, is larger than some user–defined displacement, δ. Trips can be identified in a straight–forward manner by linearly scanning through a trajectory, and calculating d_k using a look–ahead window of k measurements. That is, scanning through the total trajectory from the beginning, the first measurement for which $d_k > \delta$, signals the beginning of the first trip. Consecutive measurements are part of this trip until a measurement is reached for which $d_k \leq \delta$, which signals the end of the first trajectory. Trips following the first trip are detected in the same fashion from the remaining part of the total trajectory. Figure 1(a) shows three example trips that are derived from the total trajectory of one moving object.



(a) Identification of trips in raw trajectories

(b) Time–of–day projection and spatio–temporal region substitution

Figure 1: From trajectories to transactions

Projection of the temporal dimension

Since frequent patterns within a single object's trajectory are expected to repeat themselves daily, the temporal dimension of the so identified trips is projected down to the time–of–day domain. This projection is essential to discover the daily periodic nature of patterns in trajectories. Mining patterns with other periodicity can be facilitated by projections of the temporal domain to appropriate finer, or coarser levels of granularity. Finer levels of granularity can be used to detect patterns with shorter periodicity. For example, a delivery person might use a different route depending on the time–of–hour knowing that at the given time of the hour certain traffic conditions arise, which make an otherwise optimal delivery route sub–optimal. The detection of these patterns in delivery routes requires the projection of the temporal dimension to the time–of–hour domain. Conversely, coarser levels of granularity can be used to detect patterns with longer periodicity. For example, a person might visit his bank only at the end of pay periods. The detection of this pattern requires the projection of the temporal dimension to the day–of–month domain. Finally, to discover the pattern that the above mentioned person makes these visits to his bank Saturday mornings following the end of pay periods, requires the projection of the temporal domain to a combination of the day–of-month, the day–of-week, and the part–of-day domains. Performing different projections is part of the inherently iterative and only semi-automatic process of doing data mining when the exact format of the patterns searched for is not known beforehand. Figure 1(b) shows the projection of the temporal dimension to the time–of–day domain for the three trips identified in Figure 1(a). Since the projection of a single database record is a constant operation, the total processing time of this transformation step is optimal and linear in the number of database records.

Spatio–temporal region substitution

Trajectories are noisy. One source of this noise is due to imprecise GPS measurements. From the point of view of patterns in such trajectories, slight deviation of trajectories from the patterns can be viewed as noise. Examples of such deviations could be due to a few minute delay, or to the usage of different lanes on

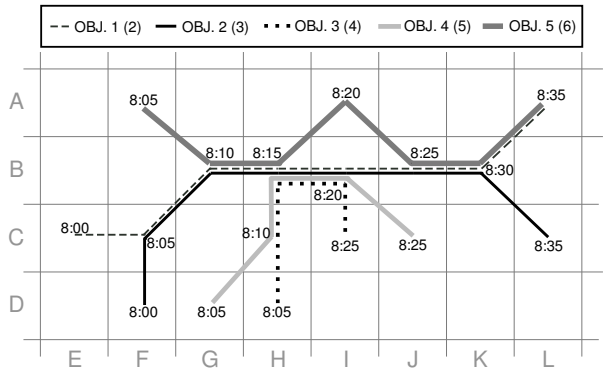


Figure 2: Illustration of the sample trajectory DB

the route. Hence, while a person might be driving from home to work at approximately the same time of day using approximately the same route, the chance of two identical trajectories is highly unlikely. Consequently, patterns in raw trajectories are few and certainly not long. Thus, patterns have to be mined in trajectories represented in a generalized way, yielding general patterns in trajectories. To achieve this generalization of trajectories, individual (x, y, t) measurements of a trajectory are discretized and mapped to the spatio–temporal regions they fall into. Thus, a generalized trajectory is constructed by substituting (x, y, t) measurements with the spatio–temporal regions they map to. If within a trajectory multiple (x, y, t) measurements map to the same spatio–temporal region, they are substituted with a single instance of the corresponding spatio–temporal region. The box in Figure 1(b) represents such a spatio–temporal region. Since spatio–temporal substitution of a single database record can be achieved using simple arithmetics from the spatial and temporal coordinates, the processing time of this transformation step is optimal and linear in the number of database records.

3.2 Example trajectory database

Figure 2 visualizes a sample trajectory database. It shows the trajectories of trips of 5 moving objects, which were derived using the three transformation steps described in Section 3.1. For clarity, the temporal dimension is projected down to the 2D–plane. Spatio–temporal regions are defined by the square cells and a five minute interval centered around time instances written inside the square. Each connected line represents specific trips of a particular object. The number of times that trip was performed by the object is represented in the width of the line, and is also written in parenthesis next to the object name in the legend. For example, the trip trajectory associated with object 3 was performed 4 times by the object. The object was in spatial regions HD, HC, HB, IB, and IC during time intervals $8{:}05 \pm 2.5$ minutes, $8{:}10 \pm 2.5$ minutes, $8{:}15 \pm 2.5$ minutes, $8{:}20 \pm 2.5$ minutes, and $8{:}25 \pm 2.5$ minutes, respectively. In the following a spatio-temporal region will be referred to by its concatenated values of the cell identifiers along the x– and y–axis, and the corresponding time instance denoting the center of the time interval of the spatio–temporal region. Hence, trips associated with object 3 will be denoted by the a sequence {HD8:05, HC8:10, HB8:15, IB8:20, IC8:25}. Furthermore, the trajectory database T is assumed to be in a relational format with schema $\langle oid, tid, item \rangle$, where *item* is a single item, that is part of the transaction *tid* associated with object *oid*. Hence, each of the four trips of object 3 is represented by 5 unique rows in T.

3.3 Problem statement

After performing the three above transformation steps, the data set can be represented in a database T containing tuples $\langle oid, tid, s \rangle$, where *oid* is an object identifier, *tid* is a trip identifier, and s is a sequence of spatio–temporal region identifiers. Since spatio–temporal region identifiers contain a temporal component, the sequence s can, without loss of information, be represented as a **set** of spatio–temporal region identifiers. Conforming to the naming convention used in the frequent itemset mining framework, a spatio–temporal region identifier will be equivalently referred to as an *item*, and a sequence of spatio–temporal region identifiers will be equivalently referred to as a *transaction*. Let X be a set of items, called an *itemset*. A transaction t *satisfies* an itemset X iff $X \subseteq t$. Let ST_X denote the set of transactions that satisfy X. The following definitions are emphasized to point out the differences between the frequent itemset mining framework and the one established here.

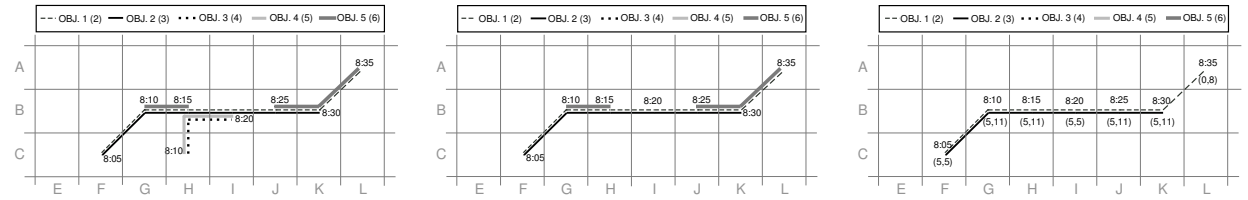
Definition 1 *The* **n–*support of an itemset*** X *in* T*, denoted as* $X.supp(n)$*, is defined as the number of transactions in* ST_X *if the number of distinct oids associated with the transactions in* ST_X *is greater than or equal to* n*, and* 0 *otherwise. The* **n–*support of an item*** i *in* T*, denoted as* $i.supp(n)$*, is equal to the* n*–support of the itemset that contains only* i*.*

Definition 2 *The* ***length of an itemset*** X*, denoted as* $|X|$*, is defined as the number of items in* X*.*

Definition 3 *An itemset* X *is* **n–*frequent*** *in* T *if* $X.supp(n) \geq MinSupp$*, and* X *is* ***long*** *if* $|X| \geq MinLength$*, where MinLength, MinSupp, and* n *are user–defined values.*

Definition 4 *An itemset* X *is* **n–*closed*** *if there exists no itemset* Y *such that* $X \subset Y$ *and* $X.supp(n) = Y.supp(n)$*.*

The task of mining LSPs in trajectories can be defined as finding all long, n–closed, n–frequent itemsets. Itemsets that meet these requirements are also referred to as LSPs, or just patterns.



(a) The sample DB after STEP 1 (b) The sample DB after STEP 2 (c) STEP 3: Item–conditional sample DB $TF_{|\mathrm{FC8:05}}$ and pattern discovery

Figure 3: Illustration of steps 1, 2, and 3

4 Projection–based LSP mining

Now let us turn to the description of the proposed method for mining LSPs in trajectories. This description is based on a number of observations, each of which is associated with a particular step in the method. In related technical report [5], these observations are also stated as lemmas, and their corresponding proofs show the correctness and completeness of the method. To demonstrate the simplicity of the implementation in a RDBMS, for each step a simple SQL–statement is given. The effect of each step is also illustrated on the previously introduced sample trajectory database assuming *MinLength* = 4, *MinSupp* = 2, and $n = 2$.

STEP 1: Filtering infrequent items

Items, i.e., spatio–temporal regions that are not frequent in T cannot be part of a LSP. Hence as first step of the method, T is filtered such that it contains items with n–support larger than or equal to *MinSupp*.

The first step can be formulated in two SQL statements:

```
INSERT INTO F (item, i_cnt)
SELECT item, count(*) i_cnt FROM T
GROUP BY item HAVING COUNT(DISTINCT oid) >= n
       AND COUNT(*) >= MinSupp

CREATE VIEW TFV AS
SELECT T.oid, T.tid, T.item
FROM T, F
WHERE T.item = F.item
```

The first statement finds items that meet the unique support criterion. The second statement constructs a filtered view of T, called *TFV*, in which transactions only contain the items found by the previous statement.

The effects of the first step are illustrated in Figure 3(a). Spatio–temporal regions, which are part of trajectories that belong to less than 2 distinct objects, are removed from trajectories. From the point of view of an intelligent rideshare application these spatio–temporal regions are uninteresting, since these parts of the trajectories cannot be shared by any objects, i.e., are not sharable.

STEP 2: Filtering of short transactions

Transactions, i.e., trip trajectories, having less than *MinLength* frequent items cannot satisfy a LSP. Hence, the second step of the method further filters *TFV* and constructs *TF* that only contain transactions that have at least *MinLength* number of items.

The second step can be formulated in one SQL statement:

```
INSERT INTO TF (tid, oid, item)
SELECT tid, oid, item FROM TFV
WHERE tid IN
   (SELECT tid FROM TFV GROUP BY tid
    HAVING COUNT(item) >= MinLength)
```

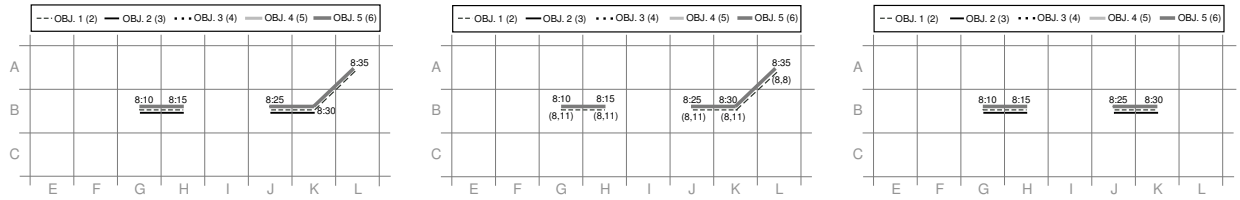
The sub–select is used to find trip identifiers that have at least *MinLength* number of items. The outer part of the statement selects all records belonging to these trip identifiers and inserts them into *TF*.

The effects of the second step are illustrated in Figure 3(b). In particular, the remaining sharable parts of trips belonging to objects 3 and 5 are deleted, because the length of them is not greater than or equal to *MinLength*, which is 4 in the example. Also, note that although in this case items HB8:15 and IB8:20 did not become infrequent in *TF*, they lost n–support.

Before stating further observations and continuing with the development of the proposed method it is important to note the following. The set of discoverable LSPs from T is equivalent to the set of discoverable LSPs from *TF*. This is ensured by first two observations. Since further steps of the proposed method will discover LSPs from *TF*, these two observation ensure the correctness of the method so far. However, it is also important to note that not all transactions in *TF* necessarily satisfy a LSP. This is due to the sequentiality of the first two steps. After the first step all the remaining items in transactions are frequent items. Then, in the second step, some of these transactions, which are not long, are deleted. Due to this deletion a frequent item in the remaining long transactions may become non–frequent, which in turn may cause some transactions to become short again. While there is no simple solution to break this circle, note that the correctness of the first and second steps are not violated since the deleted items and transactions could not have satisfied a LSP.

STEP 3: Item–conditional DB projection

For the following discussion, adopted from [10], let an item–conditional database of transactions, equivalently referred to as an an item–projected database, be



(a) The sample DB after STEP 5 (b) Item–conditional DB $TF_{|\text{LA8:35}}$ and PDD (c) Sample DB after the second PDD

Figure 4: Illustration of Pattern Discovery and Deletion phases

defined as:

Definition 5 *Let T be a database of transactions, and i an item in T. Then, the item–conditional database of transactions, is denoted as $T_{|i}$ and contains all the items from the transactions containing i.*

The construction of an item–conditional database of transactions can be formulated in a single SQL statement as:

```
INSERT INTO T_i (oid, tid, item)
SELECT t1.oid, t1.tid, t1.item
FROM TF t1, TF t2
WHERE t1.tid = t2.tid and t2.item = i
```

Given n frequent items in T, the problem of finding CFIs can be divided into n subproblems of finding the CFIs in each of the n item–projected databases [10]. Using the divide–and–conquer paradigm, each of these n subproblems can be solved by recursively mining the item-projected databases as necessary.

STEP 4: Discovery of the single most frequent closed itemset

Since i is in every transaction of the item–projected database $T_{|i}$, and hence has maximum n–support, the items in $T_{|i}$ can be grouped in two: items that have the same n–support as i, and items that have n–support less than that of i. The set of items that have the same n–support in the $T_{|i}$ as i is the Single Most Frequent Closed Itemset (SMFCI) in $T_{|i}$. The fourth step of the method discovers this SMFCI.

The fourth step can be formulated in two SQL statements:

```
INSERT INTO FT_i (item, i_cnt)
SELECT item, COUNT(*) i_cnt FROM T_i
GROUP BY item
HAVING COUNT(DISTINCT oid) >= n

SELECT item FROM FT_i
WHERE i_cnt = (SELECT MAX(i_cnt) FROM FT_i)
```

The first statement derives n–support of n–frequent items in the item–projected database, while the second statement selects those items from these n–frequent items that have maximum n–support.

Figure 3(c) shows the effects of projecting TF based on the item FC8:05. The numbers in parentheses show the n–support of the items in $TF_{|\text{FC8:05}}$ and TF respectively. The SMFCI that is immediately discovered from $TF_{|\text{FC8:05}}$ is {FC8:05, GB8:10, HB8:15, IB8:20, JB8:25, KB8:30}. LA8:35 is the only item that is in $TF_{|\text{FC8:05}}$, but is not in the discovered SMFCI. Since further projecting $TF_{|\text{FC8:05}}$ on LA8:35 yields a database of transactions where no item meets the minimum n–support criterion, the discovered SMFCI is the only CFI present in $TF_{|\text{FC8:05}}$. Since the discovered SMFCI meets both the minimum length and an minimum n–support criteria it is a pattern.

STEP 5: Deletion of unnecessary items

The subproblems that are recursively solved by the method presented so far are overlapping. That is to say, viewed from a top level, a CFI that has n items is at least once discovered in each of the n corresponding item–projected databases. To eliminate this redundancy, both in the mining process and the result set, observe that an item j can be deleted from TF if it has the same n–support in $TF_{|i}$ as in TF. The intuition behind the observation is the following. If j has the same n–support in $TF_{|i}$ as in TF, it implies that all the transactions in TF that satisfy j are also present in $TF_{|i}$. Thus, the set of patterns containing j, which can be discovered from TF, can also be discovered from $TF_{|i}$.

The fifth step can be formulated in two SQL statements:

```
INSERT INTO FT_i (item, i_cnt)
SELECT item, count(*) i_cnt FROM T_i
GROUP BY item
HAVING COUNT(DISTINCT oid) >= n

DELETE FROM TF WHERE TF.item IN
   (SELECT F.item FROM F, FT_i
    WHERE F.item = FT_i.item
    AND F.i_cnt = FT_i.i_cnt)
```

The first statement counts the n–support of items in $TF_{|i}$. The second statement deletes all items in TF that have the same n–support in TF as in $TF_{|i}$.

Figure 4(a) shows the effects of deleting the unnecessary items after the mining of $TF_{|\text{FC8:05}}$. Since items FC:8:05 and IB8:20 have the same n–support in $TF_{|\text{FC8:05}}$ as in TF, shown in Figure 3(c), they are deleted from TF. Items remaining in TF are shown in Figure 4(a).

Item-projection ordered by increasing n–Support

A LSP p in T, containing items $i_1 \dots i_k$, can be discovered from any one of the item–projected databases

$T_{|i_1}, \ldots, T_{|i_k}$. Steps 4 and 5 of the proposed method assure that p will be discovered from exactly one of these item–projected databases, but the method presented so far does not specify which one. While this point is irrelevant from the point of view of correctness, it is crucial from the point of view of effectiveness.

To illustrate this, assume that $i_1.supp(n) < i_2.supp(n) < \ldots < i_k.supp(n)$. If projections are performed in decreasing order of item n–support, then, first $T_{|i_k}$ is constructed, then $T_{|i_k|i_{k-1}}$ is constructed from it, and so on, all the way to $T_{|i_k|i_{k-1}|\ldots|i_1}$, from which finally p is discovered. If on the other hand, projections are performed in increasing order of item n–support, then p is discovered from the first item–projected database that is constructed, namely $T_{|i_1}$.

Assume that p and its qualifying $(k-l+1)$ (at least l–long) sub–patterns are the only LSPs in T. Then during the whole mining process, the total number of projections in the decreasing processing order is $P_{dec} = k$, whereas in the increasing processing order the total number of projections is only $P_{inc} = k - l + 1$. If k and l are comparable and large, then $P_{dec} \gg P_{inc}$. Similar statements can be made about the total size of the projected databases in both cases. Hence, item–projection should be performed in increasing order of item n-support.

Alternating pattern discovery and deletion

Alternating steps 3, 4 and 5, all patterns can be discovered in a recursive fashion. The sequential application of these steps is referred to as a *Pattern Discovery and Deletion phase* (PDD). Mining terminates when all items have been deleted from *TF*.

Figures 3(c) and 4(a) shows the effects of the first of these PDD phases. Figures 4(b) and 4(c) show the effect of the next pattern PDD phase. Since after the first PDD phase LA8:35 has the lowest n–support in *TF*, namely 8, it is chosen as the next item to base the database projection on. Figure 4(b) show $TF_{|\text{LA8:35}}$ with the corresponding n–support of the items in $TF_{|\text{FC8:05}}$ and *TF* respectively. Since all the items have the same n–support in $TF_{|\text{FC8:05}}$ as LA8:35, namely 8, the closed itemset {GB8:10, HB8:15, JB8:25, KB8:30, LA8:35} is discovered. Since this closed itemset both meets the minimum length and n–support requirements it is recorded as a pattern. In the deletion part of this PDD phase, item LA8:35 is deleted from *TF* as the only item that have the same n–support in $TF_{|\text{LA8:35}}$ as in *TF*. The results of this deletion are shown on Figure 4(c).

The third and final PDD phase is implicitly shown in Figure 4(c). Since after the second PDD phase all the items in *TF* have the same n–support, the next projection is performed on any one of the items, i, and the resulting item–projected database, $TF_{|i}$, is identical to the current state of *TF*, depicted on Figure 4(c). Since all the items in $TF_{|i}$ have the same n–support as i, the closed itemset {GB8:10, HB8:15, JB8:25, KB8:30} is discovered. Since this closed itemset meets both the minimum length and n–support requirements, it is recorded as a pattern. Finally, items having the same n–support in $TF_{|i}$ as in *TF*, which in this case means all the items in $TF_{|i}$, are deleted from *TF*. After this deletion part of the final PDD phase, *TF* becomes empty and the mining terminates. Figure 5 shows the three patterns that are discovered during the mining. Supporting *oid*s, n–supports, and length for each discovered patterns are shown in the legend.

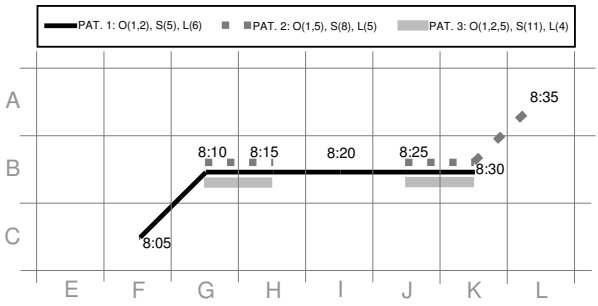


Figure 5: Three patterns in the sample DB

LSP mining algorithm

Using the observations and the associated steps, the complete algorithm for mining LSPs in trajectories is given in Figure 6. Since item–projected databases are constructed at every level of the recursion and are modified across levels when deleting unnecessary items, the level of recursion L is passed as an argument in the recursive procedure, and is used as a superscript to associate databases to the levels they were constructed in.

Lines 2 and 3 in the MineLSP procedure represent step 1 and 2 of the method, and they construct the filtered database of transactions at the initial level, level 0. Line 4 processes frequent items in TF^0 in ascending order of n–support. Line 5 represent step 3 of the method, and for each such frequent item i, it constructs the item–conditional database of transactions $TF^0_{|i}$ at level 0. Line 6 calls procedure FindLSP to extract all LSPs from $TF^0_{|i}$ recursively.

Lines 2 and 3 in the FindLSP procedure represent steps 1 and 2 of the method, and they construct the filtered database of transactions at the current level L. Line 4 represents step 4 of the method, and it finds the SMFCI P in TF^L_{long}. Line 5 represents step 5 of the method, and it deletes all items from the filtered database of transactions of the previous level, TF^{L-1}_{long}, that have the same n-support in TF^{L-1}_{long} as in TF^L_{freq}, the current level. Lines 6 and 7 check if the single most frequent closed itemset P meets the minimum requirements and store it accordingly. Lines 8 and 9 processes frequent items in TF^L_{long}, which are not in P, in ascending order of n–support. Line 10 represent step 3 of the method, and for each such frequent item i it constructs the item–conditional database of trans-

```
(1)  procedure MineLSP (T, MinSupp, MinLength, n)
(2)     TF^0_freq ← MinSupportFilter (T, MinSupp, n)
(3)     TF^0_long ← MinLengthFilter (TF^0_freq, MinLength)
(4)     for each freq item i in TF^0_long ordered by asc n–supp
(5)        TF^0_|i ← ConstructConditionalDB (TF^0_long, i)
(6)        FindLSP (TF^0_|i, 1, MinSupp, MinLength, n)
(7)     end for each

(1)  procedure FindLSP (T,L,MinSupp,MinLength,n)
(2)     TF^L_freq ← MinSupportFilter (T, MinSupp, n)
(3)     TF^L_long ← MinLengthFilter (TF^L_freq, MinLength)
(4)     (P, P.supp(n)) ← FindSMFCI (TF^L_long)
(5)     TF^{L-1}_long ← DeleteUnnecessaryItems (TF^{L-1}_long, TF^L_freq)
(6)     if P.supp(n) ≥ MinSupp and |P| ≥ MinLength
(7)        StorePattern (P, P.supp(n))
(8)     for each freq item i in TF^L_long ordered by asc n–supp
(9)        if i is not in P
(10)          TF^L_|i ← ConstructConditionalDB (TF^L_long, i)
(11)          FindLSP (TF^L_|i, L + 1, MinSupp, MinLength, n)
(12)    end for each
```

Figure 6: The LSP algorithm

actions $TF^L_{|i}$ at the current level L. Finally, line 11 recursively calls procedure FindLSP to find LSPs in $TF^L_{|i}$ at the next level.

The structure and functionality of procedures MineLSP and FindLSP have a significant overlap. While the two functions can be merged into one, the separation of the two is used to emphasize the facts that (1) DeleteUnnecessaryItems presumes the existence of databases constructed at the previous level, and (2) FindSMFCI correctly operates only on an item–projected database, and hence it can only be applied at level 1 and above.

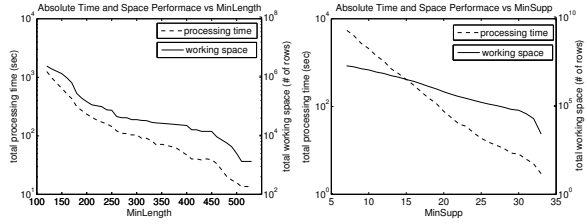
Several implementation details are worth mentioning. First, DeleteUnnecessaryItems deletes items from TF^{L-1}_{long} based on the n–support of items in TF^L_{freq}, not TF^L_{long}. This is important, as it was noted that MinLengthFilter decreases the n–support of items in TF^L_{freq}, thereby possibly making an unnecessary item appear to be necessary. Second, arguments to functions and operands in statements are logical, i.e., the functions and statements can be more efficiently implemented using previously derived tables. For example, both FindSMFCI and DeleteUnnecessaryItems are implemented using previously derived n–support count tables not the actual trajectory tables. Third, simple shortcuts can significantly improve the efficiency of the method. For example, during the derivation of TF^L_{freq}, if the number of unique frequent items in TF^L_{freq} is less than *MinLength*, no further processing is required at that level, since none of the CFIs that can be derived from TF^L_{freq} are long. To preserve clarity, these simple shortcuts are omitted from Figure 6.

5 Experimental evaluation

The proposed method was implemented using MS-SQL Server 2000 running on Windows XP on a 3.6GHz Pentium 4 processor with 2GB main memory. The method was tested on the publicly available INFATI data set, which comes from intelligent speed adaptation experiments conducted at Aalborg University. This data set records cars moving around in the road network of Aalborg, Denmark over a period of several months. 20 distinct test cars and families participated in the INFATI experiment; Team–1 consisting of 11 cars operated between December 2000 and January 2001 and Team–2 consisting of 9 cars operated between February and March 2001. Each car was equipped with a GPS receiver, from which GPS positions were sampled every second whenever the car was operated. Additional information about the experiment can be found in [8].

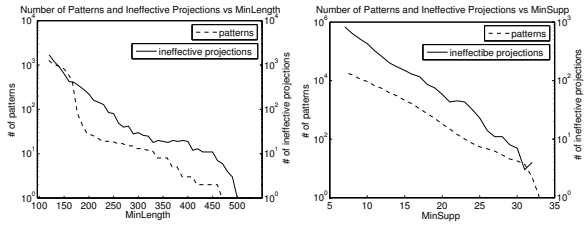
The method presented in Section 3.1 identifies trips from continuous GPS measurements, which is not the case in the INFATI data. Hence in this case, a trip was defined as sequence of GPS readings where the time difference between two consecutive readings is less than 5 minutes. Using the definition, the INFATI data contains 3699 trips. After projecting the temporal dimension to the time–of–day domain and substituting the noisy GPS readings with 100 meter by 100 meter by 5 minutes spatio-temporal regions, the resulting trajectory database has the following characteristics. There are 200929 unique items in the 3699 transactions. The average number of items in a transaction is approximately 102. The average n–support of 1–, 2–, and 3–frequent items is 1.88, 4.2 and 6.4 respectively. Notice that the averages only include the n–supports of 1–, 2–, and 3– frequent items.

Two sets of experiments were performed, each varying one of the two parameters of the algorithm, *MinSupp* and *MinLength*. The performance of the algorithms was measured in terms of processing time and working space required, where the working space required by the algorithm was approximated by the sum of the rows in the projected tables that were constructed by the algorithm. Both measures were evaluated in an absolute and a relative, per pattern, sense. Figures 7(a), 7(c), and 7(e) show the results of the first set of experiments, where $MinSupp = 2$, $n = 2$ and *MinLength* is varied between 120 and 530. Lower settings for *MinLength* were also tested, but due to the very low *MinSupp* value these measurement were terminated after exceeding the 2 hour processing time limit. Noting the logarithmic scale in Figure 7(a) it is evident that both the running time and the working space required by the algorithm exponentially increase as the *MinLength* parameter is decreased. Examining the same quantities in a relative, per pattern sense, Figure 7(e) reveals that the average running time and average working space required per pattern is approx-



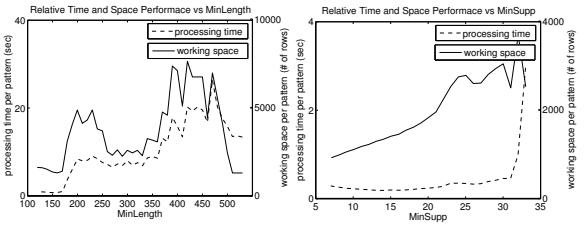
(a) Absolute time and space

(b) Absolute time and space



(c) Number of patterns

(d) Number of patterns



(e) Relative time and space

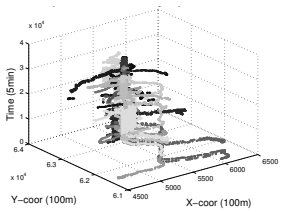
(f) Relative time and space

Figure 7: Performance evaluation for various *MinLength* (a,c,e) and *MinSupp* (b,d,f) settings

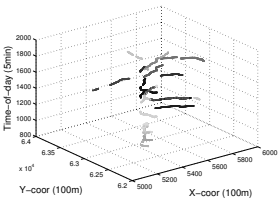
imately *linearly decreasing* as the *MinLength* parameter is decreased. The presence of the two irregular bumps in Figure 7(e) can be explained in relation to the number of patterns found, and the number of ineffective projections that yield no patterns, shown in Figure 7(c). The sharp increases in relative processing time and working space are due to the fact that the algorithm is unable to take some of the shortcuts and it performs relatively more ineffective projections yielding no pattern discovery. The sharp decreases can be explained by the presence of an increasing number of patterns that share the total pattern discovery cost.

Similar observations can be made about the second set of experiments, shown in Figures 7(b), 7(d), and 7(f), where $MinLength = 50$, $n = 2$ and *MinSupp* is varied between 7 and 33. For example, the sharp decrease in relative processing time in Figure 7(f) when going from $MinSupp = 33$ to $MinSupp = 32$ is simply due to the sudden appearance of patterns in the data for the given parameters. While there is only 1 pattern for $MinSupp = 33$, and an order of magnitude more number of patterns for $MinSupp = 32$, the projections performed and hence the absolute processing time to discover these patterns is approximately the same in both cases. Hence, the relative processing time for $MinSupp = 33$ is an order of magnitude larger than that for $MinSupp = 32$.

Figure 8(a) shows a 50–fold down–sampled version



(a) Moving object trajectories



(b) 28 discovered LSPs

Figure 8: LSP discovery results in INFATI

of the trajectories of the 20 moving objects in the INFATI data set. While some regularities are apparent in it to the human eye, to find LSPs in it seems like a daunting task. Figure 8(b) shows 28 LSPs in it that are at least 200 long, sharable for at least 2 distinct objects, and have a support of at least 2.

In conclusion, the experiments show that the method is effective and robust to changes in the user–defined parameter settings, *MinLength* and *MinSupp*, and is a useful analysis tool for finding LSPs in moving object trajectories.

6 Conclusions and future work

The herein presented work, for the first time, considers the problem of mining LSPs in trajectories, and transforms it to a framework similar to that used in frequent itemset mining. A database projection based method, and its simple SQL–implementation is presented for mining LSPs in trajectories. The effectiveness of the method is demonstrated on a real world data set.

The large number of patterns discovered are difficult to analyze. To reduce this number, future work will consider the mining of a compressed patterns in trajectories [14].

In future work, we also plan to perform additional experiments on larger real world data sets when such become available. These experiments will include the investigation of scale–up properties of the algorithm as the number of moving objects are increasing and/or as the granularity of the spatio–temporal regions is varied.

Acknowledgments

This work was supported in part by the Danish Ministry of Science, Technology, and Innovation under grant number 61480.

References

[1] R. Agrawal, T. Imilienski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. of SIGMOD*, pp. 207–216, 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of VLDB*, pp. 487–499, 1994.

[3] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proc. of ICDE*, pp. 3–14, 1995.

[4] G. Gidofalvi and T. B. Pedersen. Spatio-Temporal Rule Mining: Issues and Techniques. In *Proc. of DaWaK*, pp. 275–284, 2005.

[5] G. Gidofalvi and T. B. Pedersen. Mining Long, Common Patterns in Trajectories of Moving Objects. A DB Technical Report (15), 2006: `www.cs.auc.dk/DBTR/DBPublications/DBTR-15.pdf`

[6] B. Goethals. Survey on frequent pattern mining. `citeseer.ist.psu.edu/goethals03survey.html`

[7] C. S. Jensen, D. Pfoser, and Y. Theodoridis. Novel Approaches to the Indexing of Moving Object Trajectories. In *Proc. of VLDB*, pp. 395-406, 2000.

[8] C. S. Jensen, H. Lahrmann, S. Pakalnis, and S. Runge. The INFATI data. Time Center TR–79, 2004: `www.cs.aau.dk/TimeCenter`

[9] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung. Mining, Indexing, and Querying Historical Spatiotemporal Data. In *Proc. of KDD*, pp. 236–245, 2004.

[10] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proc. of DMKD*, pp. 11-20, 2000.

[11] X. Shang, K.-U. Sattler, and I. Geist. Efficient Frequent Pattern Mining in Relational Databases. In *Proc. of LWA*, pp. 84–91, 2004.

[12] I. Tsoukatos and D. Gunopulos. Efficient Mining of Spatiotemporal Patterns. In *Proc. of SSTD*, pp. 425–442, 2001.

[13] M. Vlachos, D. Gunopoulos, and G. Kollios. Discovering Similar Multidimensional Trajectories. In *Proc. of ICDE*, pp. 673–685, 2002.

[14] D. Xin, J. Han, X. Yan, and H. Cheng. Mining Compressed Frequent-Pattern Sets. In *Proc. of VLDB*, pp. 709–720, 2005.

[15] X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large datasets. In *Proc. of SDM*, pp. 166–177, 2003.

Fusion Based Methodology for Spatial Clustering

Pavani Kuntala and Vijay V. Raghavan
The Center for Advanced Computer Studies
University of Louisiana at Lafayette, LA 70504-4330
pavani@louisiana.edu, raghavan@cacs.louisiana.edu

Abstract

In this paper, a novel clustering algorithm is proposed to address the clustering problem within both spatial and non-spatial domains by employing a fusion-based approach. The motivation for this work is to overcome the limitations of the existing spatial clustering methods. In most conventional spatial clustering algorithms, the similarity measurement mainly takes the geometric attributes into consideration. However, in many real applications, there is a need to fuse the information from both the spatial and the non-spatial attributes. The goal of our approach is to create and optimize clusters, such that the data objects satisfy both spatial and non-spatial similarity constraints. The proposed algorithm first captures the spatial cores having the highest structure and then employs an iterative, heuristic mechanism to determine the optimal number of spatial cores and non-spatial clusters that exist in the data. Such a fusion-based framework allows for comparing clusters in spatial and non-spatial contexts. The correctness and efficiency of the proposed clustering algorithm is demonstrated on real world data sets.

1. Introduction and Motivation for Spatial Clustering

Clustering is one of the prominent data mining tasks, which has been studied in detail[14]. Spatial data mining or knowledge discovery in spatial databases refers to the extraction, from spatial databases, of implicit knowledge, spatial relations, or other patterns that are not explicitly stored [6]. The large size and high dimensionality of spatial data make the complex patterns that lurk in the data hard to find. It is expected that the coming years will witness very large number of objects that are location-enabled to varying degrees. Spatial clustering has been used as an important process in the areas such as geographic analysis, exploring data from sensor networks, traffic control, and environmental studies. Spatial data clustering has been identified as an important technique for many applications and several techniques have been proposed over the past decade based on density-based strategies, random walks, grid based strategies, and brute-force exhaustive searching methods[8]. This paper deals with spatial clustering using a fusion-based approach.

Spatial data is about instances located in a physical space. Spatial clustering aims to group similar objects into the same group considering spatial attributes of the object. The existing spatial clustering algorithms in literature focus exclusively either on the spatial distances or minimizing the distance of object attributes pairs. i.e., the locations are considered as another attribute or the non-spatial attribute distances are ignored. Much activity in spatial clustering focuses on clustering objects based on the location nearness to each other[12]. Finding clusters in spatial data is an active research area, and the current non-spatial clustering algorithms are applied to spatial domain, with recent application and results reported on the effectiveness and scalability of algorithms [9, 12]. One of the earliest and most prominent clustering algorithms applied for spatial data mining are partition based approaches. Partitioning algorithms are best suited to such problems where minimization of a distance function is required and a common measure used in such algorithms is the Euclidian distance. Recently new set of spatial clustering algorithms has been proposed, which represents faster method to find clusters with overlapping densities. DBSCAN, GDBDCAN and DBRS are density-based spatial clustering algorithms, but they each perform best only on particular types of datasets[12]. However, these algorithms also ignore the non-spatial attribute participation and require user defined parameters. For large-scale spatial databases, the current density based cluster algorithms can be found to be expensive as they require large volume of

Proceedings of the 3rd Workshop on STDBM
Seoul, Korea, September 11, 2006

memory support due to its operations over the entire database. Another disadvantage is the input parameters required by these algorithms are based on experimental evaluations. There is a large interest in addressing the automation of the general purpose clustering approach without user intervention[13]. However, it is difficult to adapt these algorithms to spatial data.

Spatial dimensions (e.g., latitude and longitude) cannot simply be treated as two additional non-spatial dimensions because of several important reasons. Spatial data are generally multi-dimensional, requires the adoption of real-world dissimilarity measures, and are autocorrelated. Spatial data includes topological relationships. These distinctions put spatial and non-spatial data into different categories with far-reaching implications for conceptual, processing, and storage issues. General-purpose clustering methods mainly deal with non-spatial feature spaces and have very limited power in recognizing spatial patterns that involve identification of dense spatial neighborhoods.

In section 2 we present the proposed fusion based spatial clustering. The results of the application of proposed clustering on real-world problems are examined in section 3. The conclusions are presented in section 4.

2. Proposed Fusion-Based Spatial Clustering

Data fusion is a process in which the available data is combined to find representations of higher quality. The U.S. Department of Defense [2] stated that "data fusion is a multilevel, multifaceted process dealing with the automatic detection, association, correlation, estimation, and combination of data and information from multiple sources." More recent concepts of data fusion are discussed in [11]. Data Fusion is used in different context in different fields. For example in fields of sensors and image analysis it is viewed as target clustering problem [4]. We illustrate in our proposed approach, that effective data fusion can be achieved through clustering and vice versa.

The method proposed for the fusion is a two step procedure comprising the following steps: (i) an initial phase for locating the initial spatial cores based on the spatial densities and (ii) an optimization phase for finding the optimal spatial and non-spatial fusion. In the initialization phase, we find the initial data tessellation, called spatial cores, based on spatial densities. The second phase is the fusion of spatial and non-spatial constraints. The goal of the optimization phase is to iteratively provide an intuitive grouping of the data objects, such that these groups satisfy both the spatial and non-spatial constraints. In every iteration of the optimization phase, we check if the algorithm is improved. The algorithm is considered improved, when the spatial distance is minimized and the non-spatial dissimilarity is also minimal in each spatial partition. Next, we describe the details of the initialization and optimization phases of our proposed algorithm.

2.1 Initialization

We apply a density-based approach to get reasonable neighborhood for the spatial core initialization process. These approaches hold that, for given radius each object within a cluster, the neighborhood of a give cluster must exceed a specified threshold. Density clustering methods depend on the proper selection of the neighborhood size and the density size. Without prior knowledge of the structure of the input data set, proper parameter selection is cumbersome.

The initialization step of our proposed algorithm differs from the existing approaches mainly in the following two aspects.

Instead of choosing a fixed radius, we employ a heuristic approach to find a sequence of varying neighborhoods to catch the cluster structures of the input data.

The clusters are grown in the direction of maximum movement of objects. Here we assume the object is attracted to the strongest neighbor, which is the neighbor with highest densities.

We assume the input data set X is $X = \{\vec{X}_1, \vec{X}_2, ..., \vec{X}_i, ..., \vec{X}_n\}$, where $\vec{X}_i$ is an object represented as a vector of non-spatial features, and n is the total number of objects. $\vec{X}_i = (X_{i1}, X_{i2}, .., X_{ij,..}, X_{id}) \in \Re^d$, where d is the total number of non-spatial features, and X_{ij} represents the value of attribute j of object $\vec{X}_i$.

Let $L = \{L_1, L_2, ...L_\lambda\}$ be the set of spatial locations in the data space. The set of objects at location s, L_s *are* $\{L_p = \{\overrightarrow{l(X_{i'})}, ..., \overrightarrow{l(X)}_{i''}\} \mid 1 \le i', i'' \le n, |X_s| \le |X|\}$. If we assume each location as a representation of singleton set of objects, i.e., one object per location, the object information at any location can be represented as $\overrightarrow{l(X_i)}$. That is, the cardinality of L is the same as the cardinality of X, *i.e.,* $\lambda = n$. We scan the input data set X once to find the spatial densities of each object, within a neighbourhood of radius φ. Let z be the number of spatial cores, and set it to a small value initially. Initially we find the area of the minimum bounding rectangle of the objects, and tessellate the data into z cores. For each object $\vec{X}_i$ we find the density ξ_i of that object as the number of the other objects that are within a geometric neighbourhood of radius φ.

Initially, we assume that the objects are uniformly distributed among all initial spatial cores, and the spatial clusters have a uniform neighborhood area. This initial assumption makes the algorithm independent of the input order of the data, and provides a better approach, than partitioning the data using random seeds. Note that in later stages of our algorithm, the number of cores varies based on the optimization stage. At this stage, since we are finding dense

partitions based only on spatial dimensions, a Minimum Bounding Rectangle, MBR provides a good approximation for the total area of the objects.

Minimum Bounding Rectangles: The initial tessellation of data is based on Minimum Bounding Rectangles. MBRs have been used extensively to approximate objects in spatial data structures and spatial reasoning, because they need only two objects for their representation; in particular, for a set of objects $X^{'}$, where $X^{'} \subseteq X$, $X^{'}$ is represented as an ordered pair $\left(l_1(X^{'}), l_2(X^{'})\right)$. Such approximations naturally extended to finding topological relations[7]. $l_1(X^{'}), l_2(X^{'})$ correspond to the lower left, and the upper right object of the MBR that covers all objects $X^{'}$. $l_1(X^{'})$ and $l_2(X^{'})$ are called the edge objects.

Definition 1. *Given a set of objects* $X^{'}$, *let* $l_\alpha(X^{'}) = \left(l_{1\alpha}(X^{'}), l_{2\alpha}(X^{'})\right)$ *and* $l_\beta(X^{'}) = \left(l_{1\beta}(X^{'}), l_{2\beta}(X^{'})\right)$ *be the intervals of* $X^{'}$, *created by projecting all the objects on spatial dimensions α and β, respectively. The MBR of the set of objects X' is defined as the domain* $\left(l_1(X^{'}), l_2(X^{'}), X^{'}\right)$, *where* $l_1(X^{'}) = \left(l_{1\alpha}(X^{'}), l_{1\beta}(X^{'})\right)$, *and* $l_2(X^{'}) = \left(l_{2\alpha}(X^{'}), l_{2\beta}(X^{'})\right)$, *and* $l_{1\alpha}(X^{'}) <= \alpha <= l_{2\alpha}(X^{'}) \wedge l_{1\beta}(X^{'}) <= \beta <= l_{2\beta}(X^{'})$.

Initial Tessellation of Data Space: We define dense region as a set of data objects, clustered using only spatial information. The density cluster as defined by Ester et. al in[3], requires both surrounding radius of an object φ, within which at least ξ neighboring objects should be found. The main differences between our proposed spatial core approach and the current density clustering approaches are: 1) Our proposed algorithm heuristically calculates the radius φ, and, 2) the dense regions are evolved only in the direction in which the objects naturally grow.

We propose a heuristic approach to compute φ. Assume that objects are uniformly distributed in the clusters are equally separated in the MBR. Then assuming the clusters are circular, $\varphi = f(MBR,z) = \sqrt{(1/\pi z) A(X')}$ where $A(X^{'})$ is the area of the MBR $\left(l_1(X^{'}), l_2(X^{'}), X^{'}\right)$. By employing such a heuristic approach, we can find clusters where the data points are not densely packed and hence might have a low ξ.

Since the number of spatial clusters evolve as the algorithm evolves, and the shapes of the clusters change, these assumptions are good to estimate the initial cores. The densities of all the objects within radius φ are calculated and sorted. Let $\overrightarrow{X_i}$ be the object with the maximum density. We assign all the objects within radius of φ to one spatial cluster. For the other $z-1$ clusters we grow the cluster in the direction of its movement. This allows for outlier detection and identifying a dense region that is surrounded by another uniformly distributed sparse region.

Any objects belonging to $X^{'}$, and not part of initial dense regions are considered as unclassified objects. In the optimization stage detailed in next section, the objects in the above initial cores and unclassified objects are optimized to arrive at the clusters satisfying both the spatial and non-spatial constraints.

2.2 Optimization

The optimization stage aims at dynamically combining the best features of both spatial and non-spatial distances in the manner of alternating, fusing spatial and non-spatial clusters, and arriving at heuristically computed number of spatial cores and non-spatial clusters. We introduce some more notations and definitions to explain the concepts of cores and $\wp$-clusters. In previous section we mentioned that each data object $\overrightarrow{X_i}$ is a vector of d non-spatial features. We define $\wp$-clusters are clusters satisfying non-spatial optimization criterion, and core as a representation of a set of objects which satisfy the spatial constraints. i.e., all objects in a core are geometrically proximate.

Definition 2. *A* $\wp$*- cluster* C_f *is a set of objects which are homogenous only in non-spatial dimensions and are well-separated from the other* $\wp$*-clusters only in non-spatial dimensions.*

The non-spatial distances are dependent on the application domain and types of features. Assuming $d_{ns}\left(X_{ij}, X_{i'j}\right)$ is the non-spatial distance function, and Opt_{ns} is the non-spatial optimization function, the objective function for $\wp$- clusters is to minimize

$$Opt_{ns}(X,K) = \sum_{f=1}^{k}\sum_{i=1}^{n}\sum_{j=1}^{d} d_{ns}\left(X_{ij}, Z_{fj}\right),$$

where X represents the object matrix, K is the cluster membership matrix, and k is the number of $\wp$-clusters.

The following statements hold when we refer to two kinds of centers, *gravity* and *centroid* as defined in definition 4. The geometrical centers of the spatial cores or $\wp$-clusters will be referred to as the gravity. The centers based on non-spatial features, of spatial cores or $\wp$-clusters, will be referred to as centroid. Z_f is the centroid of the $\wp$-cluster f, and Z_{fj} is the centroid value of non-spatial fea-

ture j for $\wp$-cluster f. The centroid Z_f is the average of the non-spatial characteristics.

Definition 3. *A core C^r is a combination of $\wp$- clusters. Two distinct cores can have the $\wp$- clusters that are non-spatially similar. Each $\wp$- cluster of a core indicates the objects in a core that are non-spatially homogenous and well-separated from the other $\wp$- clusters in the core. C_f^r represents a $\wp$- cluster in core C^r.*

The following criterion holds for cores and $\wp$- clusters.

1. $C_f{}^r \neq \phi, r = 1...z;\ f = 1...k$
2. $\left| \bigcup_{r=1}^{z} \bigcup_{f=1}^{k} C_f^r \right| <= \ z \times k$
3. $C^r \bigcap C^{r'} = \phi, 1 \le r, r' \le z, r \neq r'$
4. $C_f{}^r \bigcap C_{f'}{}^r = \phi, 1 \le f, f' \le k, f \neq f'$
5. $\left| \bigcup_{r=1}^{z} C_f^r \right| \le k$

Let object $\overrightarrow{X_i}$ belonging to core C^r be represented as X_i^r, and object $\overrightarrow{X_i}$ belonging to $\wp$-cluster C_f^r be represented as X_i^{rf}.

Definition 4. *The gravity Z^r for core C^r (or Z_f^r for $\wp$-cluster C_f^r) , is defined as the geometrical center for the set of objects in the core (or $\wp$-clusters) . It is computed as the mean of the locations of all objects in the core.*

$$Z^r = \frac{\sum_{i=1}^{p} l(X_i^r)}{p_r},$$

where p_r is the number of objects in C^r.

The objective function for core C^r is to minimize the spatial distances among the $\wp$-clusters belonging to that core. Depending on spatial dimensions, any spatial distance metric, for example rectilinear distance function[1] for two dimensional spatial features can be used to find the spatial proximity. Assuming $d_{sp}(X_i, X_{i'})$ is the spatial distance function, and Opt_{sp} is the spatial optimization function, the objective function for core C^r is to minimize

$$Opt_{sp}(C_f^r) = \sum_{f=1}^{q_r} \sum_{i=1}^{p_f} d_{ns}\left(X_i^{rf}, Z_f^r\right),$$

where q_r is the number of $\wp$-clusters in a core C^r and p_f is the number of objects in $\wp$-*cluster* C_f^r.

The intuition of the fusion approach is to use the information derived from spatial proximity distance to merge-split the non-spatial clusters and vice-versa. We define the distance function for the fusion based approach as

$$d_{fusion} = \left(\sum_{1}^{z} d_{ns}(Z^{r_t}, Z^{r_{t'}}) + \sum_{1}^{k} d_{sp}(Z_{f_a}, Z_{f_{a'}})\right)^{1/2},$$

where $t, t' \in \{1..z; t \neq t'\}$ and $a, a' \in \{1..z; a \neq a'\}$. After, the initial clusters have been created, the optimization phase of the algorithm iteratively tries to improve the object distribution among the cores and the non-spatial distribution of the $\wp$-clusters. i.e., $Opt_{fusion}(X) = \min(d_{fusion})$.

The pseudo code of clustering based on co-learning and fusion of spatial and non-spatial algorithm is depicted as shown in Fig. 1. In the beginning of the optimization phase, we consider each dense region formed in the initialization phase as a core. A representative non-spatial $\wp$-cluster is found for each core. All the objects belonging to a representative cluster should satisfy the following criteria.

$$rep(C_f^r) = \{X_i : i = 1...p_r \mid X_i \in C^r \wedge X_i \in C_f\},$$

where

$$X_i \in C^r \quad iff \quad d_{sp}\left(X_i, Z^r\right) < d_{sp}\left(X_i, Z^{r'}\right), 1 \le r, r' \le z; r \neq r'$$

$$X_i \in C_f \quad iff \quad d_{ns}\left(X_i, Z^{r,f}\right) < d_{ns}\left(X_i, Z^{r'f'}\right), 1 \le f, f' \le k; f \neq f'$$

i.e., find the objects that have the non-spatial characteristics similar to the centroid of the current core. In the iterative phase of the optimization algorithm, new cores and $\wp$-clusters are formed based on defined merge-split rules. Each object $\overrightarrow{X_i}$ is categorized as follows: *C-Similar, $\wp$-similar*, and *joint*. *C-Similar* implies that the object is spatially proximate to the same core but not to any of the existing $\wp$-clusters, i.e., it is non-spatially similar to the object closest to the gravity of the unclassified objects. *$\wp$-similar* implies that the object is similar to a $\wp$-cluster but not geometrically proximate to existing cores, i.e., it is geometrically proximate to the object closest to the centroid of unclassified objects. An object is considered *joint* if and only if it is similar to an existing $\wp$-cluster of a core. All the unclassified objects after each iteration, which satisfy the *joint* criteria, are merged with the existing $\wp$-cluster of a core. All the objects of a core which are *C-similar,* are considered as tentative $\wp$-clusters of the core that they are geometrically proximate. Similarly all objects, which are *$\wp$-similar*, form new tentative cores. Based on the MBRs, if the tentative cores are non-overlapping they are considered singleton cores, and are merged with a core whose gravity is closer than the gravity of the unclassified objects. If not singleton, and the MBRs are overlapping, a new core is formed. After each iteration of the optimization phase, we compute the new fusion distance, and if the rate of decrease d_{fusion} is greater

than the percentage of unclassified objects, we terminate the algorithm.

The algorithm is guaranteed to converge, because the d_{fusion} ensures that the points are geometrically proximate to only one core, and non-spatially proximate to only one $\wp$-cluster of a core, at any given time. An added benefit of the above fusion based clustering process is that the cores can be explored to retrieve region specific $\wp$-clusters. Further we can compare two cores, by comparing the $\wp$-clusters of any two given cores.

Algorithm Optimize
Input *: Initial cores of objects*
Output*: Spatial Cores and $\wp$-clusters in each core*
Step 1:
Initial z dense regions;
Find the gravities and centroids of each core;
Step 2:
$d_{f\_prev} = d_{fusion}$;
Find the rep $\wp$-clusters, $rep(C_f^r)$, of each core;
Compute C-similar, $\wp$-similar, and joint;
Step 3:
For all unclassified objects {
Add the joint objects to existing $\wp$-cluster of a core;
In each core, create $\wp$-clusters with all objects c- similar;
Create tentative cores with all objects $\wp$-similar;
}
Step 4:
if non-overlapping singleton cores {
merge with an existing core;}
compute $d_{fusion;}$
Step 5:
Find representative $\wp$- clusters $rep(C_f^r)$
Objects that do not belong to core or $\wp$-cluster considered unclassified;
if $[(d_{f\_prev} - d_{fusion})/ d_{f\_prev} - d_{fusion}] > [$ unclassified / n)]
go to step 2
else exit;
}

Figure 1. Algorithm Optimize

3. Experimental Evaluation and Results

Comprehensive experiments were conducted to assess the accuracy and efficiency of the proposed approach. By comparing to existing algorithms, such as purely spatial clustering (GraviClust[5]), Non-Spatial(k-means), and spatial clustering algorithm with partial capability to support non-spatial attributes(GDBSCAN), we demonstrate accuracy of the proposed approach. Our choice of comparative algorithms is based on their effectiveness and popularity in literature. Since our algorithm emphasizes on the need for looking at both the spatial distances and attribute values, we evaluate our algorithm in the three evaluation setups, as shown in section 3.1.

We evaluated both the effectiveness and efficiency of our approach using two real world datasets. In the spatial clustering literature, there are no fixed benchmarking datasets for evaluation of the algorithms. Our choice of datasets is based on the number of non-spatial dimensions, and the distribution of classes in both spatial and non-spatial dimensions. To demonstrate the functionality of the fusion based approach, we will details the experiments conducted using a data set with two spatial dimensions. Next, we show our experimental results on high-dimensional data sets to show the scalability of our approach.

The first data set is US census tract housing data. The observations contain 4 non-spatial attributes land area, population, median per capita income, median year built, as well as the latitude and longitude of the centroid of the tract from the 1990 Census. This resulted in 57,647 observations with complete data. The associated class information is the log of the median price of housing. After looking at the Gaussian distribution of the class values, we binned the log of median price of housing into 7 classes. The class distribution for this dataset is equally influenced by the spatial and non-spatial features. The second data set is a multivariate GIS data set. The actual forest cover type for a given observation (30 x 30 meter cell) was determined from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. Independent variables were derived from data originally obtained from US Geological Survey (USGS) and USFS data. The total number of instances (observations) are around 500,000. The attributes are distributed as 54 columns of data. The data represents 7 types of forest cover types. We found for this data set the class distribution is spatially biased.

3.1 Evaluation Metrics

Since the final results of an algorithm are clusters based on the fusion of different distance metrics, our evaluation metric should be independent of any particular distance metric. We identified three evaluation criterion to measure the quality of our clusters.

1. Cluster Accuracy: r-value (purity). The r-value as defined in [10] is

$$r = (1/n)\sum_{i=1}^{c} a_i \quad ,$$

where, c stands for the number of classes , a_i denotes the number of objects of the cluster i with the largest representation in class *i*, and *n* is the total number of objects in the database. This criterion measures the degree of correspondence between the resulting clusters and the classes assigned a priori to each object.

2. Total number of cluster assignment errors: Number of objects that co-occur in a class versus that appear in different clusters.

3. Number of clusters found: Total number of cores or clusters discovered versus the actual number of classes in the data.

3.2 Effectiveness

In this section, we demonstrate the effectiveness of our algorithm, in comparison to the KM, GDBSCAN and GraviClust, and for varying data set sizes. We chose the census dataset to demonstrate the effectiveness our algorithm, since the natural grouping of the data is equally influenced by the spatial and non-spatial features.

We varied the dataset sizes by gradually increasing the size of a region. We incrementally increased the longitude by one degree, thus obtaining varying data set sizes (and number of classes). This provides a natural spatial partition of the data, without losing the inherent spatial and non-spatial information. Instead if we have chosen random samples of the objects in census data, the spatial information would become meaningless. Table 1 shows the data set sizes and the number of classes of each sample.

Table 1. Sample Data Set Sizes and Number Of Classes

Degrees of Longitude	Number of Objects	Number of Classes
4	694	5
4.2	1191	5
5	1931	5
6	2750	6
7	5984	6
8	7953	6
10	11841	7
13	15615	7

Table 2 illustrates the effectiveness of our algorithm in comparison with traditional k-means(KM) and the GDBSCAN algorithm. Higher r-value indicates better accuracy. Since the competitor algorithms are input order and parameter dependent, we heuristically chose the settings that resulted in maximum accuracy for the competitors.

Table 2. Accuracy of the clusters based on r-value

DataSet Size	KM	GDBSCAN	Gravi-Clust	Proposed Fusion based Clustering (Cores)	Proposed Fusion based Clustering ($\wp$-Clusters)
694	0.49424	0.20749	0.56052	0.69308	0.67147
1191	0.39463	0.59446	0.69353	0.82284	0.70109
1931	0.37856	0.56499	0.4811	0.90005	0.6753
2750	0.31927	0.39673	0.41127	0.80655	0.51709
5984	0.30348	0.52741	0.39205	0.8735	0.54596
7953	0.26619	0.58758	0.4382	0.87476	0.55363
1184	0.31771	0.50452	0.45182	0.91031	0.56397

The last two columns of Table 2, are the accuracy results separately for the cores and $\wp$-clusters. The values in these columns illustrate that the cores we obtained are effective, considering either the spatial or non-spatial constraints. The high accuracy values of cores in comparison with the $\wp$-clusters illustrate that the way the classes (in this case the range of house prices) are grouped, are based more on their spatial coordinates than the non-spatial features.

The next effectiveness measure is cluster assignment errors. Table 3 and Fig. 2 illustrate the number of misclassified objects using varying approaches. From Table 3 and Fig. 2 we observe by using fusion based approach the number of objects misclassified are significantly lower than the competitor algorithms. Note that even though k-means used the spatial attributes it performed very poorly in comparison to the proposed fusion based approach.

Table 3. Total Number of Misclassified objects

#Objects	KM	GDBSCAN	GraviClust	Proposed Fusion Based Clustering
694	351	550	305	213
1191	721	483	365	211
1931	1200	840	1002	193
2750	1872	1659	1619	532
5984	4168	2828	3638	757
7953	5836	3280	4468	996
11841	8079	5867	6491	1062

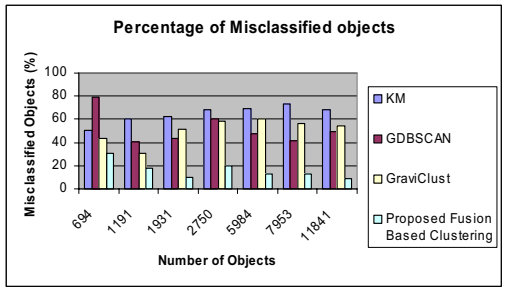


Figure 2. The percentage of misclassified objects

The final evaluation of effectiveness is based on the number of discovered clusters. Table 4 demonstrates the number of spatial partitions we found, are closer to the actual number of classes in the original data. In each core there are varying numbers of $\wp$-clusters, since all the regions might not have same number of classes (in this case household price range).

Table 4. The Number of Actual Classes Compared to The Number of Discovered Cores/ρ-Clusters

DataSize	Actual Classes	Number of discovered Cores	Non-Spatial Clusters /Core	Number of Discovered Clusters
108	4	3	(3,1,2)	3
694	5	5	(4,3,1,1)	4
1191	5	6	(5,4,4,3,1)	6
1931	5	8	(5,4,3,1,1)	6
2750	6	8	(2,3,2,1,1)	6
5984	6	5	(3,2,2)	5

3.3 Efficiency

As we iterate for an optimal solution, the number of data objects that need to be clustered decreases significantly. Hence our algorithm is more efficient, because the algorithm works on progressively reduced problem space. The

efficiency can be further improved by using spatial indexing for the core clustering. Fig. 3 illustrates the efficiency of our algorithm in comparison to the other spatial clustering algorithms.

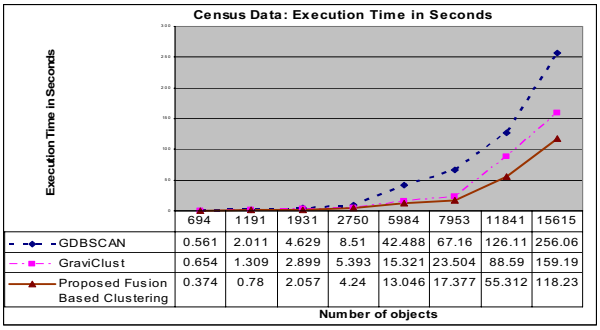


Figure 3. Execution Time in Seconds on Dataset1 with Varying Number of Objects

The efficiency results as applied on forest dataset are illustrated Table 5. Here we show the iterations as opposed to time in secs, because we keep the dataset size constant to 5000 points, and vary the number of classes. Here it can be seen that the proposed approach converges much faster than KM. The iterations do not apply for GDBSCAN as it is a single pass algorithm.

Table 5. DataSet 2: Number of iterations for convergence for 3,5& 7 clusters

#Classes	KM	GraviClust	Proposed Fusion Based Clustering
3	14	12	11
5	21	20	19
7	46	17	15

In Table 6, we summarize the characteristics of our proposed approach to some of the other classes of clustering algorithms supporting spatial features.

Table 6. Advantages of the Proposed Fusion based Spatial Clustering

	Distance Metrics	K-required	Detects Noise	Requires parameters/thresholds	Arbitary shape clusters	Support for Incremental
Partition	Spatial ,or, Non-Spatial	Y	N	Y	N	Y
Density	Non-Spatial Filter	N	Y	Y	Y	N
Gravity	Spatial	Y	N	Y	N	N
Proposed Fusion Clustering	Fusion Based	N	Y	N	Y	Y

4. Conclusions

In this paper, we proposed an iterative and automated spatial clustering algorithm based on the spatial and non-spatial attribute fusion. The experimental evaluation shows that our approach is more effective than the existing approaches, which either consider location attribute as if it is another non-spatial feature, or ignore non-spatial attribute information altogether. The proposed clustering is automated because it does not require any user input, and employs a heuristic approach to find the optimal number of spatial core clusters. We also proposed a fusion based metric to find the optimal clusters and a stopping criterion for algorithm convergence. In summary, the proposed algorithm has following characteristics : i*nsensitive* to input order; a*utomatic*, that is the number of clusters need not be specified as input; iteratively clusters the data based on *fusion* of spatial and non-spatial attributes; i*ntuitive* spatial cluster comparison; permits the use of separate *distance metrics* for spatial and non-spatial features.

5. References

[1] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Introduction to Algorithms: MIT Press, 1990.

[2] U. S. D. o. Defence, "Data fusion lexicon," 1991.

[3] M. Ester, H. P. Kriegel, J. Sander, and X. X, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," Proc. 2nd International Conference on Knowledge Discovery and Data Mining, pp. 226-231, 1996.

[4] P. S. Huang and T.-M. Tu, "A Target Fusion-Based Approach for Classifying High Spatial Resolution Imagery," Proc. 2003 IEEE Workshop on Advances in Techniques for Analysis of Remotely Sensed Data, pp. 175 - 181, 2003.

[5] M. Indulska and M. E. Orlowska, "Data Clustering: Gravity Based Spatial Clustering," Proc. 10th ACM International Symposium on Advances in Geographic Information Systems, pp. 125 - 130, 2002.

[6] K. Koperski and J. Han, "Discovery of Spatial Association Rules in Geographic Information Databases," Proc. 4th International Symposium on Large Spatial Databases, pp. 47-66, 1995.

[7] D. Papadias, Y. Theodoridis, T. Sellis, and M. Egenhofer, "Topological Relations in the World of Minimum Bounding Rectangles: A study with R-trees," Proc. ACM SIGMOD International Conf. on Management of Data, pp. 92-103, 1995.

[8] J. Roddick and B. G. Lees, "Paradigms for Spatial and Spatio-Temporal Data Mining," in Geographic Data Mining and Knowledge Discovery, H. Miller and J. Han, Eds.: Taylor & Francis, 2001.

[9] Y. Tao, J. Zhang, D. Papadias, and N. Mamoulis, "An Efficient Cost Model for Optimization of Nearest Neighbor Search in Low and Medium Dimensional Spaces," IEEE Transactions on Knowledge and Data Engineering, vol. 16, no. 10, pp. 1169-1184, 2004.

[10] D. K. Tasoulis and M. N. Vrahatis, "Generalizing the K-Windows Clustering Algorithm in Metric Spaces," Mathematical and Computer Modelling (accepted for publication), 2005.

[11] L. Wald, "Some Terms of Reference in Data Fusion," IEEE Transactions on Geoscientific Remote Sensing, vol. 37, no. 3, pp. 1190-1193, 1999.

[12] X. Wang and H. J. Hamilton, "Clustering Spatial Data in the Presence of Obstacles," International Journal on Artificial Intelligence Tools, vol. 14, no. 1-2, pp. 177-198, 2005.

[13] Y. Xie, V. V. Raghavan, P. Dhatric, and X. Zhao, "A New Fuzzy Clustering Algorithm For Optimally Finding Granular Prototypes," International Journal of Approximate Reasoning, vol. 40, no. 1-2, pp. 109-124, 2005.

[14] R. Xu and D. Wunsch, II, "Survey of clustering algorithms," IEEE Transactions on Neural Networks, vol. 16, no. 3, pp. 645- 678, 2005.

A Storage Scheme for Multi-dimensional Databases Using Extendible Array Files

Ekow J. Otoo and **Doron Rotem**

Lawrence Berkeley National Laboratory
1 Cyclotron Road
University of California
Berkeley, CA 94720, USA
ekw,@hpcrd.lbl.gov,d_rotem@lbl.gov

Abstract

Large scale scientific datasets are generally modeled as k-dimensional arrays, since this model is amenable to the form of analyses and visualization of the scientific phenomenon often investigated. In recent years, organizations have adopted the use of on-line analytical processing (OLAP), methods and statistical analyses to make strategic business decisions using enterprise data that are modeled as multi-dimensional arrays as well. In both of these domains, the datasets have the propensity to gradually grow, reaching orders of terabytes. However, the storage schemes used for these arrays correspond to those where the array elements are allocated in a sequence of consecutive locations according to an ordering of array mapping functions that map k-dimensional indices one-to-one onto the linear locations. Such schemes limit the degree of extendibility of the array to one dimension only. We present a method of allocating storage for the elements of a dense multidimensional extendible array such that the bounds on the indices of the respective dimensions can be arbitrarily extended without reorganizing previously allocated elements. We give a mapping function $\mathcal{F}_*()$, and its inverse $\mathcal{F}_*^{-1}()$, for computing the linear address of an array element given its k-dimensional index. The technique adopts the mapping function, for realizing an extendible array with arbitrary extendibility in main memory, to implement such array files. We show how the extendible array file implementation gives an efficient storage scheme for both scientific and OLAP multi-dimensional datasets that are allowed to incrementally grow without incurring the prohibitive costs of reorganizations.

Proceedings of the third Workshop on STDBM
Seoul, Korea, September 11, 2006

1 Introduction

Datasets used in large scale scientific applications, are generally modeled as multidimensional arrays and matrices. Matrices are 2-dimensional rectangular array of elements (or entries) laid out in rows and columns. Although the logical view of a rectangular array of elements need not be the same as the actual physical storage, the elements of the arrays are mapped into storage locations according to some linearization function of the indices. An array file is simply a file of the elements of an array in which the k-dimensional indices of the elements map into linear consecutive record locations in the file. The mapping of k-dimensional indices to linear addresses may either be by a computed access function, an indexing mechanism, or a mixture of both. Operations on these array files involve accessing and manipulating sub-arrays (or array chunks) of one or more of these array files. Such models for manipulating datasets are typical in large scale computing performed by various Scientific and Engineering Simulations, Climate Modeling, High Energy and Nuclear Physics, Astrophysics, Computational Fluid Dynamics, Scientific Visualization, etc.

On-line analytical processing (OLAP) and decision support systems depend highly on efficiently computing statistical summary information from multi-dimensional databases. A view of the dataset as a multi-dimensional array model forms the basis for efficient derivation of summary information. The literature distinguishes OLAP on *relational model* of data called *ROLAP* and that on multi-dimensional model termed *MOLAP*.

Consider a dataset that maintains monitored values of *temperature* at defined locations given by the *latitude* and *longitude* at various time-steps. Figure 1 illustrates a simple multi-dimensional view of a sample data as a 3-dimensional array A[4][3][3] with assigned ordinal coordinates. The entries shown in the cells correspond to the linear addresses of the cells as the relative displacements from cell $A\langle 0,0,0\rangle$.

Each cell stores the temperature value (not shown) as the *measure*. Suppose we wish to maintain information on

sales of some enterprise that has stores at different locations. The locations are defined by their spatial coordinates of *latitude* and *longitude* often abbreviated as *Lat/Long*. The sales information recorded for say three consecutive months can be represented by Figure 1 which gives the same view of the data as before but with values for sales as the *measures*. A multi-dimensional array correctly models the dataset in both domains of scientific applications and MOLAP. Further common characteristics associated with multi-dimensional model of data under these domains are that:

1. The data incrementally grow over time by appending new data elements. These may reach orders of terabytes, as in data warehousing.
2. The datasets are mainly read-only. However, they may be subject to expansions in the bounds of the dimensions, e.g., as new monitoring locations are enabled or new stores are opened.
3. The number of dimensions (i.e., the *rank*) of the array may be extended.
4. The array representation can be either dense or sparse.

In the illustration of Figure 1, the array may grow by appending data for new time-steps. When new locations are added, the bounds of the Lat/Long dimensions may be required to be extended. The mapping function depicted in the above figure (represented by the label inside each cell) corresponds to that of conventional array mapping that allows extendibility in one dimension only; namely the dimension in the mapping scheme that is least varying. We desire an array addressing scheme that allows extensions in both the bound and rank of the array file and also efficiently manages sparse arrays. In this paper we address primarily the storage scheme for managing dense extendible array efficiently. We discuss how the technique is tailored for handling sparse arrays as well without detailed experimentation due to space limitation.

Over the years special file formats have been extensively studied and developed for storing multi-dimensional array files that support sub-array accesses in high performance computing. These include NetCDF [10], HDF5 [7] and disk resident array (DRA) [11]. DRA is the persistent counterpart of the distributed main memory array structure called Global Array [12]. Except for HDF5, these array files allow for extendibility only in one dimension. We say an array file is extendible if the index range of any dimension can be extended by appending to the storage of previously allocated elements, so that new elements can be addressed from the newly adjoined index range. The address calculation is done without relocating elements of the previously allocated elements and without modifying the addressing function.

Let $A[\mathbb{N}_0][\mathbb{N}_1]\ldots[\mathbb{N}_{k-1}]$, denote a k-dimensional array where $\mathbb{N}_j, 0 \leq j < k-1$, is the bound on the indices of dimension j. An element, denoted by $A\langle i_0, i_1 \ldots, i_{k-1}\rangle$, is referenced by a k-dimensional index $\langle i_0, i_1 \ldots, i_{k-1}\rangle$. Let $\mathcal{L} = \{\ell_0, \ell_1 \ldots, \ell_{\mathbb{M}-1}\}$, be a sequence of consecutive storage locations where $\mathbb{M} = \prod_{j=0}^{k-1} \mathbb{N}_j$. An allocation (or mapping) function $\mathcal{F}()$, maps the k-dimensional indices one-to-one, onto the sequence of consecutive indices $\{0, 1, \ldots, \mathbb{M}-1\}$, i.e., $\mathcal{F} : \mathbb{N}_0 \times \mathbb{N}_1 \times \cdots \times \mathbb{N}_{k-1} \rightarrow \{0, 1, \ldots, \mathbb{M}-1\}$. Given a location index j, the inverse mapping function $\mathcal{F}^{-1}(j)$, computes the k-dimensional index $\langle i_0, i_1 \ldots, i_{k-1}\rangle$ that corresponds to j. The significance of inverse mapping functions have not been much appreciated in the past but has important use in computing the addresses of the neighbors of an element given its linear storage address and also for managing a general k-dimensional sparse array as opposed to sparse matrices which is 2-dimensional.

Modern programming languages provide native support for multidimensional arrays. The mapping function $\mathcal{F}()$ is normally defined so that elements are allocated either in *row-major* or *column major* order. We refer to arrays whose elements are allocated in this manner as conventional arrays. Conventional arrays limit their growth to only one dimension. We use the term *row-major order* in a general sense, beyond row-and-column matrices, to mean an order in which the leftmost index of a k-dimensional index is the least varying. This is also sometimes referred to as the *lexicographic order*. A *column-major order* refers to an ordering in which the rightmost index varies the slowest.

In conventional k-dimensional arrays, efficient computation of the mapping function is carried out with the aid of a vector that holds k multiplicative coefficients of the respective indices. Consequently, an N-element array in k-dimensions, can be maintained in $O(N)$ storage locations and the computation of the linear address of an element, given its k-dimensional coordinate index, is done in time $O(k)$ since this is done by $k-1$ multiplications and $k-1$ additions. We achieve similar efficiency in the management of extendible arrays by maintaining vectors of multiplying coefficients each time the array expands. The computation of the linear address, corresponding to a k-dimensional index, uses these stored vectors of coefficients. The vectors of multiplicative coefficients maintain records of the history of the expansions and are organized into *Axial-Vectors*. There is one *Axial-Vector* for each dimension and holds entries that we refer to as *expansion records*. The fields of an expansion record are described later.

Consider a k-dimensional array of N elements and after some arbitrary extensions dimension j maintains $\mathcal{E}_j$ records of the history of expansion for dimension j. Our approaches computes the linear address from the k-dimensional index in time $O(k + \sum_{j=0}^{k-1} \mathcal{E}_j)$ using $O(k \sum_{j=0}^{k-1} \mathcal{E}_j)$ additional space.

The technique being presented in this paper can be adopted by compiler developers for generating mapping functions for dense multidimensional extendible arrays in memory as well. The Standard Template Library (STL) in C++ also provides support for resizable vectors. However, the general technique for allowing array extensions without the extensive cost of reallocating storage is what we desire. The approach we propose may be used in implementing

special libraries such as the Global Array (GA) library [12] and the disk resident array (DRA) [11] that manage dense multidimensional arrays.

To handle sparse arrays, we adopt the technique of *array chunking* [4, 7, 5], where the linear address of a chunk is computed as for a dense array. The generated address forms the key of an array chunk that is used in an index scheme that stores only non-empty chunks. Partial chunks can be further compressed and grouped into physical buckets to maintain efficient storage utilization. A thorough discussion of our approach is in a follow-up paper.

The main contributions in this paper are:

- A definition of a mapping function for extendible arrays that is applicable for both dense in-core arrays and out-of-core arrays. We call the approach the *The Axial-Vector* method. The general principle is not new. The idea was first proposed with the use of an *auxiliary array*. However, the storage overhead using an auxiliary array could be prohibitive. The new *Axial-Vector* method obviates the storage overhead and also gives a new method for computing the linear addresses.

- When extremely large array files are generated and stored, any dimension can still be expanded by appending new array elements without the need to reorganize the already allocated storage which could be of the order of terabytes.

- The mapping function proposed in this paper incurs very little storage overhead and can be used as a replacement for the access function for array files such as *NetCDF* and the data chunks in the *HDF5* file format.

- We discuss an extension of the technique to handle sparse extendible arrays for both in-core and out-of-core arrays.

The organization of this paper is as follows. In the next section we present some details of the definition of the mapping function for dense multidimensional extendible arrays in main memory since the same mapping function is adopted for accessing elements of extendible array files. In section 3 we describe how an array file is implemented. We describe how sparse multidimensional array files are managed in section 4. We give some experimental comparison of the array mapping functions for extendible and conventional arrays in section 5. We conclude in section 6 and give some directions for our future work.

2 The Mapping Function for an Extendible Array

2.1 Addressing Function for a Conventional Array

First we explore some details of how conventional arrays are mapped onto consecutive storage locations in memory. Consider a k-dimensional array $A[\mathbb{N}_0][\mathbb{N}_1]\ldots[\mathbb{N}_{k-1}]$, where $\mathbb{N}_j, 0 \leq j < k-1$, denote the bounds of the index ranges of the respective dimensions. Suppose the elements of this array are allocated in the linear consecutive storage locations $\mathcal{L} = \{\ell_0, \ell_1 \ldots, \ell_{\mathbb{M}-1}\}$, in row-major order according to the mapping function $\mathcal{F}()$. In the rest of this paper, we will always assume row-major ordering and for simplicity we will assume an array element occupies a unit of storage. A unit of storage could be one word of 4 bytes, a double word of 8 bytes, etc. An element $A\langle i_0, i_1, \ldots i_{k-1}\rangle$ is assigned to location ℓ_q, where q is computed by the mapping function defined as

$$q = \mathcal{F}(\langle i_0, i_1, \ldots i_{k-1}\rangle) = i_0 * C_0 + i_1 * C_1 + \cdots + i_{k-1} * C_{k-1}$$
$$\text{where} \quad C_j = \prod_{r=j+1}^{k-1} \mathbb{N}_r, 0 \leq j \leq k-1. \tag{1}$$

and $A\langle 0, 0, \ldots 0\rangle$ assigned to ℓ_0.

In most programming languages, since the bounds of the arrays are known at compilation time, the coefficients $C_0, C_1, \ldots, C_{k-1}$ are computed and stored during code generation. Consequently, given any k-dimensional index, the computation of the corresponding linear address using Equation 1, takes time $O(k)$.

Suppose we know the linear address q of an array element, the k-dimensional index $\langle i_0, i_1, \ldots i_{k-1}\rangle$ corresponding to q can be computed by repeated modulus arithmetic with the coefficients $C_{k-2}, C_{k-3}, \ldots, C_1$ in turn, i.e., $\mathcal{F}^{-1}(q) \rightarrow \langle i_0, i_1, \ldots i_{k-1}\rangle$. When allocating elements of a dense multidimensional array in a file, the same mapping function is used where the linear address q gives the displacement relative to the location of the first element in the file, in units of the size of the array elements. The limitation imposed by $\mathcal{F}$ is that the array can only be extended along dimension 0 since the evaluation of the function does not involve the bound of $\mathbb{N}_0$.

We can still circumvent the constraint imposed by $\mathcal{F}$ by shuffling the order of the indices whenever the array is extended along any dimension. The idea of extending the index range of a dimension is simply to adjoin a block (or a hyperslab) of array elements whose sizes on all dimensions remain the same except for the dimension being extended.

2.2 The Mapping Function for an In-Core Extendible Array

The question of organizing an extendible array in memory, such that the linear addresses can be computed in the manner similar to those of a static array described above, is a long standing one [15]. Some solutions exist for extendible arrays that grow to maintain some predefined shapes [14, 15]. A solution was proposed that uses an auxiliary array [13] to keep track of the information needed to compute the mapping function. In [16], a similar solution was proposed that organized the content of the auxiliary array with a B-Tree. The use of auxiliary arrays can be prohibitively expensive in storage depending on the pattern of expansions of the array. We present a new approach to

organizing the content of the auxiliary array with the use of *Axial-Vectors*. The idea of using axial-vectors to replace the auxiliary array was introduced in [20] but only for 2-dimensional arrays. The method introduced maintains the same information as in the auxiliary-array approach and does not generalize easily to k-dimensional arrays. Furthermore since it requires that information be stored for each index of any dimension, the method incurs the same prohibitive cost for certain array shapes. The method introduced in this paper avoids these problems. First we show how an in-core extendible array is organized with the aid of axial-vectors since the same mapping function is used for array files.

2.3 The Axial-Vector Approach for Extendible Arrays

Consider Figure 2a that shows a map of the storage allocation of 3-dimensional extendible array. We denote this in general as $A[\mathbb{N}_0^*][\mathbb{N}_1^*][\mathbb{N}_2^*]$, where $\mathbb{N}_j^*$ represents the fact that the bound has the propensity to grow. In this paper we address only the problem of allowing extendibility in the array bounds but not its rank. The labels shown in the array cells represent the linear addresses of the respective elements, as a displacement from the location of the first element.

Suppose initially the array is allocated as $A[4][3][1]$, where the corresponding axes of Latitude, Longitude and Time have the instantaneous respective bounds of $\mathbb{N}_0^* = 4, \mathbb{N}_1^* = 3$ and $\mathbb{N}_3^* = 1$. The array was extended by one time-step followed by another time-step. The sequence of the two consecutive extensions along the same time dimension, although occurring at two different instances, is considered as an *uninterrupted extension* of the time dimension. Repeated extensions of the same dimension, with no intervening extension of a different dimension, is referred to as an interrupted extension and is handled by only one expansion record entry in the axial-vector.

The labels shown in the array cells represent the linear addresses of the respective elements. For example, in Figure 2a the element $A\langle 2,1,0\rangle$ is assigned to location 7 and element $A\langle 3,1,2\rangle$ is assigned to location 34. The array was subsequently extended along the longitude axis by one index, then along the latitude axis by 2 indices and then along the time axis by one time-step. A *hyperslab* of array elements can be perceived as an array *chunk* (a term used in [17, 7]), where all but one of the dimensions of a *chunk* take the maximum bounds of their respective dimensions.

Consider now that we have a k-dimensional extendible array $A[\mathbb{N}_0^*][\mathbb{N}_1^*]\ldots[\mathbb{N}_{k-1}^*]$, for which dimension l is extended by λ_l, so that the index range increases from $\mathbb{N}_l^*$ to $\mathbb{N}_l^* + \lambda_l$. The strategy is to allocate a *hyperslab* of array elements such that addresses within the *hyperslab* are computed as displacements from the location of the first element of the *hyperslab*. Let the first element of a hyperslab of dimension l be denoted by $A\langle 0,0,\ldots,\mathbb{N}_l^*,\ldots,0\rangle$. Address calculation is computed in row-major order as before, except that now dimension l is the least varying dimension in the allocation scheme but all other dimensions retain their relative order. Denote the location of $A\langle 0,0,\ldots,\mathbb{N}_l^*,\ldots,0\rangle$ as $\ell_{\mathbb{M}_l^*}$ where $\mathbb{M}_l^* = \prod_{r=0}^{k-1}(\mathbb{N}_r^*)$. Then the desired mapping function $\mathcal{F}_*()$ that computes the address q^* of a new element $A\langle i_0, i_1, \ldots i_{k-1}\rangle$ during the allocation is given by:

$$q^* = \mathcal{F}_*(\langle i_0, i_1, \ldots i_{k-1}\rangle) = \mathbb{M}_l^* + (i_l - \mathbb{N}_l^*)C_l^* + \sum_{\substack{j=0 \\ j\neq l}}^{k-1} i_j C_j^*$$

$$\text{where} \quad C_l^* = \prod_{\substack{j=0 \\ j\neq l}}^{k-1} \mathbb{N}_j^* \quad \text{and} \quad C_j^* = \prod_{\substack{r=j+1 \\ r\neq l}}^{k-1} \mathbb{N}_r^* \tag{2}$$

We need to retain for dimension l the values of $\mathbb{M}_l^*$ - the location of the first element of the hyperslab, $\mathbb{N}_l^*$ - the fisrt index of the adjoined hyperslab, and $C_r^*, 0 \leq r < k$ - the multiplicative coefficients, in some data structure so that these can be easily retrieved for computating an element's address within the adjoined hyperslab. The *axial-vectors* denoted by $\Gamma_j[\mathcal{E}_j], 0 \leq j < k$, and shown in Figure 2b, are used to retain the required information. $\mathcal{E}_j$ is the number of stored records for axial-vector Γ_j. Note that the number of elements in each axial-vector is always less than or equal to the number of indices of the corresponding dimension. It is exactly the number of uninterrupted expansions. In the example of Figure 2b, $\mathcal{E}_0 = 2, \mathcal{E}_1 = 2$, and $\mathcal{E}_2 = 3$.

The information of each expansion record of a dimension is a record comprised of four fields. For dimension l, the i^{th} entry denoted by $\Gamma_l\langle i\rangle$ consists of $\Gamma_l\langle i\rangle.\mathbb{N}_l^*$; $\Gamma_l\langle i\rangle.\mathbb{M}_l^*$; $\Gamma_l\langle i\rangle.C[k]$ - the stored multiplying coefficients for computing the displacement values within the hyperslab; and $\Gamma_l\langle i\rangle.S_{i_l}$ - the memory address where the hyperslab is stored. Note however that for computing record addresses of array files, this last field is not required, since new records are always allocated by appending to the existing array file. In main memory an extendible array may be formed as a collection of disjoint hyperslabs since a block of memory acquired for each new hyperslab may not necessarily be contiguous to a previously allocated one. Contiguiety in memory allocation is only guranteed for unintrrupted expansion of a dimension, i.e., when the same dimension is repeatedly expanded.

Given a k-dimensional index $\langle i_0, i_1, \ldots, i_{k-1}\rangle$, the main idea in correctly computing the linear address is in determining which of the records $\Gamma_0\langle z_0\rangle, \Gamma_1\langle z_1\rangle \ldots \Gamma_{k-1}\langle z_{k-1}\rangle$, has the first maximum starting address of its hyperslab. The index z_j is given by a modified binary search algorithm that always gives the highest index of the axial-vector where the expansion record has a maximum starting address of its hyperslab less than or equal to i_j.

For example, suppose we desire the linear address of the element $A\langle 4,2,2\rangle$, we first note that $z_0 = 1, z_1 = 0$, and

Array Method	Address calculation	Inverse addr. calculation	Storage overhead
Conventional	$O(k)$	$O(k)$	0
Axial Vectors	$O(k+$ $k\log(k+\mathcal{E})$ $-k\log k)$	$O(k+\log\mathcal{E})$	$O(k\mathcal{E})$

Table 1: Summary of features of extendible array realization.

$z_2 = 1$. We then determine that

$$\begin{aligned} \mathbb{M}_l^* &= \max(\Gamma_0\langle 1\rangle.\mathbb{M}_0^*, \Gamma_1\langle 0\rangle.\mathbb{M}_1^*, \Gamma_2\langle 1\rangle.\mathbb{M}_2^*) \\ &= \max(48, -1, 12); \end{aligned} \quad (3)$$

from which we deduce that $\mathbb{M}_l^* = 48, l = 0$, and $\mathbb{N}_l^* = \mathbb{N}_0^* = 4$. The computation $\mathcal{F}_*(\langle 4,2,2\rangle) = 48 + 12 \times (4-4) + 3 \times 2 + 1 \times 2 = 48 + 0 + 6 + 2 = 56$. The value 56 is the linear address relative to the starting address of 0. The main characteristics of the extendible array realization is summarized in the following theorem

Theorem 2.1. *Suppose that in a k-dimensional extendible array, dimension j undergoes $\mathcal{E}_j$, uninterrupted expansions. Then if $\mathcal{E} = \sum_{j=0}^{k-1} \mathcal{E}_j$, the complexity of computing the function $\mathcal{F}_*()$ for an extendible array using axial-vectors is $O(k + k\log(k+\mathcal{E}) - k\log k)$, using $O(k\mathcal{E})$ worst case space.*

Proof. The worst case sizes of the axial-vectors occur if each dimension has the same number of uninterrupted expansions; i.e., $\mathcal{E}_j = \mathcal{E}/k$. The evaluation of $\mathcal{F}_*()$ involves $k \log \mathcal{E}_j$ followed by k multiplications k additions and 1 subtraction, giving a total of $O(k + k(\log(1 + \mathcal{E}/k)))$ = $O(k + k\log(k+\mathcal{E}) - k\log k)$.

The additional space requirement for the k axial-vectors is $O((k+3)\sum_{j=0}^{k-1} \mathcal{E}_j) = O(k\mathcal{E})$. □

Given a linear address q^*, it is easy to find, an entry in the axial vectors that has the maximum starting address less than or equal to q^* using a modified binary search algorithm as in the address computation. The repeated modulus arithmetic as described in section 2.1 is then used to extract the k-dimensional indices. We state without a formal proof the following.

Theorem 2.2. *Given a linear address q of an element of an extendible array realized with the aid of k axial-vectors, the the k-dimensional index of an element is computable by the function $\mathcal{F}_*^{-1}$ in time $O(k + \log\mathcal{E})$.*

The main results of the extendible array realization are summarized in Table 1.

3 Managing Multidimensional Extendible Array Files

An array file is simply a persistent storage of the corresponding main memory resident array in a file, augmented with some meta-data information, either as a header in the same file or in a separated file. We will consider array files as formed in pairs: the primary file F_p and the meta-data file F_m. For extremely large files, the content in memory at any time is a subset, or a subarray of the entire disk resident array file. Scientific applications that process these arrays consider array chunks as the unit of data access from secondary storage. Most high performance computations are executed as a parallel program either on a cluster of workstations or on massively parallel machines. The model of the data is a large global array from which subarrays are accessed into individual workstations. The subarrays of individual nodes together constitute tiles of the global array. Actual physical access of array elements is carried out in units of array chunks for both dense and sparse arrays. We discuss this in some detail in the next section.

For the subsequent discussions, we will ignore most of the details of the organization of the array files and also the details of the structure of the array elements. For simplicity, we consider the array files as being composed of fixed size elements, where each element is composed of a fixed number of attributes. Each attribute value has a corresponding ordinal number that serves as the index value. Mapping of attribute values to ordinal numbers is easily done for array files. The primary file F_p, contains elements of the multidimensional array that continuously grows by appending new data elements whenever a dimension is extended. For example, a dimension corresponding to time may be extended by the addition of data from new time-steps. A meta-data file F_m stores the records that correspond to the axial vectors. The contents of the meta-data file F_m are used to reconstruct the memory resident axial vectors. Each expansion of a dimension results in an update of an axial vector and consequently the meta-data file as well.

Suppose an application has already constructed the memory resident axial vectors from the meta-data file, then the linear address of an element (i.e., a record), of the array file given its k-dimensional coordinates, is computed using the mapping function $\mathcal{F}_*()$. Essentially, $\mathcal{F}_*()$ serves as a hash function for the elements of the array file. Conversely, if the linear address q^*, of an element is given and one desires the neighbor element that lies some units of coordinate distances along some specified dimensions from the current, such an element can be easily retrieved with the aid of the inverse function $\mathcal{F}_*^{-1}()$. First we compute the k-dimensional coordinate values from $\mathcal{F}_*^{-1}(q)$, adjust the coordinate values along the specified dimensions and compute the address of the desired element by $\mathcal{F}_*()$. The relevant algorithms for these operations are easily derived from the definitions of the functions presented in the preceding sections. One of the main features of our scheme is that when extremely large array files are generated, each dimensions can still be expanded by appending new array

elements without the need to reorganize the allocated storage of the files.

Two popular data organization schemes for large scale scientific datasets are *NetCDF* [10] and *HDF5* [7]. The *NetCDF* maintains essentially an array file according to a row-major ordering of the elements of a conventional array. Consequently, the array can only be extended in one dimension. The technique presented in this paper can be easily adopted as the mapping function of the *NetCDF* storage scheme to allow for arbitrary extensions of the dimensions of the NetCDF file structure, without incurring any additional access cost. *HDF5* is a storage scheme that allows array elements to be partitioned in fixed size sub-arrays called *data-chunks*. A chunk is physically allocated on secondary storage and accessed via a B^+-tree index. Chunking allows for extendibility of the array along any dimension and also for the hierarchical organization of the array elements, i.e., elements of a top level array is allowed to be an array of a refined higher resolution and so on. The mapping function introduced can be used as a replacement of the B^+-tree indexing scheme for the *HDF5* array chunks. Other applications of the mapping function introduced here include its use for the *Global-Arrays* [12] data organization and *Disk-Resident-Array* files [11].

4 Managing Sparse Extendible Array Files

Besides the characteristics that multi-dimensional databases incrementally grow over time, they also have the unkind property of being sparse. Techniques for managing large scale storage of multi-dimensional data have either addressed the sparsity problem of the array model [2, 3, 4, 5, 17, 18] or the extendibility problem [16, 19] but not both simultaneously. The sparsity of multidimensional array is managed by array *chunking* technique. An array chunk is defined as a block of data that contains extents of all dimensions from a global multi-dimensional array. Even for dense array, an array chunk constitutes the unit of transfers between main memory and secondary storage.

Figure 3a shows a 3-dimensional array partitioned into chunks using index-intervals of 3. Addressing elements of the array is computed in two levels. The first level address computation gives the chunk address of the element. The second level address is computed as the displacement of the array element within the array chunk. The extendible array addressing method maps the k-dimensional index of each chunk into a *Table Map*. The table map of the chunks contains pointers to the physical addresses of the data blocks that hold chunks. An array chunk forms the unit of data transfer between secondary storage and main memory. A table map pointer is set to *null* if the chunk holds no array elements. As in extendible hashing, the use of table map to address chunks guarantees at most 2 disk accesses to locate a chunk. Array chunks that have less than some defined number of array elements can be compressed further.

The table map is one of the simplest techniques for handling sparse extendible array but suffers from the problem of a potential exponential growth for high dimensional datasets. Instead of a table map for maintaining array chunks, we use a dynamically constructed PATRICIA trie [9] to manage the table map of the chunks. Other methods of handling sparse multi-dimensional arrays have been described in [4, 8]. Some of the techniques for handling sparse matrices [1] can also be adopted.

4.1 Alternative Methods for Addressing Array Chunks

The use of a table map for locating the physical locations of array chunks relaxes the need for directly addressing array elements with an extendible array mapping function. One can further relax this constraint by doing away with the extendible array mapping function entirely. Rather, a method for constructing a unique address $I_{\langle i_0,\ldots i_{k_0}\rangle}$ of an array chunk from the k-dimensional index $\langle i_0,\ldots i_{k_0}\rangle$ is all that is required. One such method is given by concatenating the binary representation of the coordinate indices of an array chunk. The unique address generated is then used in an index scheme such as a B^+-tree, to locate the physical chunk where an array element is stored. This approach is actually implemented in the HDF5 storage format. There are two problems with this approach;

1. Either the k-dimensional index or the generated identifier for the chunk must be stored with the chunk. For the latter case, an inverse function for computing the k-dimensional chunk index from the chunk identifier is needed but is less space consuming.

2. It does not handle both memory resident array and disk resident arrays uniformly.

In general the table map can be replaced with any indexing scheme that maintains $O(N)$ chunk address for exactly N non-empty chunks. The use of a PATRICIA trie guarantees that. Using extendible array mapping function for computing the linear address of a chunk has the advantage of:

1. giving us a uniform manner of managing extendible arrays resident both in-core and out-of-core.

2. allowing the replacement of the global k-dimensional address of an array element by one which only defines its linear location within an array chunk and yet enables us to compute the global k-dimensional index from the linear address.

4.2 Operations on Multidimensional Array Files

Besides the creation of the multi-dimensional array files, and operations for reading, writing (i.e., accessing) and appending new element, the application domain dictates the type of operations the array files are subjected to. While multi-dimensional OLAP applications see the efficient computation of the *Cube* operator [6, 21] as a significant operation, applications in other scientific domains

require efficient extractions of sub-arrays for analysis and subsequent visualization. In both domains, efficiently accessing elements of a sub-array is vital to all computations carried out.

The mapping function provides direct addressing for array chunks and subsequently to the array elements. A naive approach to performing sub-array extraction operation would be to iterate over the k-dimensional coordinates of the elements to be selected and retrieve each array elements independently. Suppose the cardinality of the response set of the first selection class is $\Re$. The naive approach performs $\Re$ independent disk accesses. But one can do better than this worst case number of disk accesses. An efficient method for processing any of the above queries is to compute the chunk identifies of the element; and for each chunk retrieved, extract all elements the chunk that satisfies the request. Due to space limitation, detailed discussions on the structures, algorithms and experiments on extendible sparse multidimensional arrays is left out in this paper.

5 Performance of Extendible Array Files

The theoretical analysis of the mapping function for extendible arrays indicates that it is nearly of the same order of computational complexity as that of conventional arrays. The main difference being the additonal time requied by the mapping function for an extendible array to perform binary searches in the axial-vectors. We experimentally tested this by computing the average access times of both the conventional array and extendible array for an array size of approximately 10^8 elements of double data types. We varied the rank of the array from 2 to 8 while keeping the size of the array about the same. We plotted the average time over 10000 random element access for static arrays. These experiments were run on a 1.5GHz AMD Athlon processor running Centos-4 Linux with 2GByte memory. Figure 4a show the graphs of the access times averaged over 10000 element access.

The graphs indicate that for static arrays, the cost of computing the extendible array access function varies more significantly with the rank of the array than for the conventional array access function. Even though the complexity of computing the access functions are both $O(k)$, the extendible array access function shows strong dependence on the array's rank k due to the fact that k binary searches are done in the axial-vectors.

We also considered the impact on the average access cost when the arrays undergo interleaved expansions. The experiment considered arrays of ranks 2 and 3 where the initial array size grew from about 10000 elements to about 10^6 elements. For each sequence of about 10000 accesses the array is allowed to undergo up to 16 expansions. A dimension selected at random, is extended by a random integer amount of between 1 and 10. Each time the conventional array is extended, the storage allocation is reorganized. The average cost of accessing array elements with interleaved expansions is shown in Figure 4b. Under this model, we find that the average cost of accessing array elements for extendible arrays is significantly less than for conventional arrays that incur the additional cost of reorganization.

Similar experiments were conducted, for accessing elements of array files instead of memory resident arrays. Figure 5a compares the average time to access elements for 2, 3 and 4 dimensional static files. There is very little variation in the times of the conventional and extendible array functions. However, when these times are computed with interleaved expansions, the extendible array clearly outperforms the conventional array methods by several orders of magnitude. Figure 5b shows the graphs for $2, 3$ and 4-dimensional extendible array files only. The extra time and storage required to reorganize the conventional array files with interleaved expansions become prohibitive as the array becomes large. One can infer from these results that, in handling large scale dense multi-dimensional dataset that incrementally grow by appending elements, the extendible array mapping function should be the choice for addressing storage.

We should mention that a competitive model for comparisons of multi-dimensional array file implementations should be with the HDF5 data schemes. Our implementation currently does not include array caching of data pages which the HDF5 implementation uses. Work is still ongoing to add main memory buffer pools for data caching at which time a fair comparison would be made.

6 Conclusion and Future Work

We have shown how a k-dimensional extendible array file can be used to implement multi-dimensional databases. The technique applies to extendible arrays in-core just as much as for out-of-core extendible arrays that can be either dense or sparse. The method relies on a mapping function that uses information retained in axial-vectors to compute the linear storage addresses from the k-dimensional indices. Given the characteristics of multi-dimensional databases that they incrementally grow into terabytes of data, developing a mapping function that does not require reorganization of the array file as the file grows is a desirable future.

The method proposed is highly appropriate for most scientific datasets where the model of the data is perceived typical as large global array files. The mapping function developed can be used to enhance current implementations of array files such as *NetCDF*, *HDF5* and *Global Arrays*. Work is still on-going to incorporate our proposed solution to multidimensional array libraries and to extend the technique for multi-dimensional datasets whose dimensions or ranks are allowed to expand. We are also conducting comparative studies on the different techniques for managing sparse multi-dimensional extendible arrays.

Acknowledgment

This work is supported by the Director, Office of Laboratory Policy and Infrastructure Management of the U. S. Department of Energy under Contract No. DE-AC03-76SF00098. This research used resources of the National Energy Research Scientific Computing (NERSC), which is supported by the Office of Science of the U.S. Department of Energy.

References

[1] J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst. *Sparse Matrix Storage Formats, in Templates for the solution of algebraic eigenvalue problems: A practical guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[2] P. M. Deshpande, K. Ramasamy, A. Shukla, and J. F. Naughton. Caching multidimensional queries using chunks. In *Proc. ACM-SIGMOD, 1998*, pages 259–270, 1998.

[3] P. Furtado and P. Baumann. Storage of multidimensional arrays based on arbitrary tiling. In *Proc. of 15th Int'l. Conf. on Data Eng. (ICDE'99)*, page 480, Los Alamitos, CA, USA, 1999. IEEE Computer Society.

[4] S. Goil and A. N. Choudhary. Sparse data storage schemes for multidimensional data for olap and data mining. Technical Report CPDC-TR-9801-005, Center for Parallel and Dist. Comput, Northwestern Univ., Evanston, IL-60208, 1997.

[5] S. Goil and A. N. Choudhary. High performance multidimensional analysis of large datasets. In *Int'l. Wkshp on Data Warehousing and OLAP*, pages 34–39, 1998.

[6] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *J. Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[7] Hierachical Data Format (HDF) group. *HDF5 User's Guide*. National Center for Supercomputing Applications (NCSA), University of Illinois, Urbana-Champaign, Illinois, Urbana-Champaign, release 1.6.3. edition, Nov. 2004.

[8] Nikos Karayannidis and Timos Sellis. Sisyphus: The implementation of a chunk-based storage manager for olap data cubes. *Data snf Knowl. Eng.*, 45(2):155–180, 2003.

[9] D. E. Knuth. *The Art of Computer Programming: Fundamental Algorithms*, volume 1. Addison-Wesley, Reading, Mass., 1997.

[10] NetCDF (Network Common Data Form) Home Page. http://my.unidata.ucar.edu/content/software/netcdf/index.html.

[11] J. Nieplocha and I. Foster. Disk resident arrays: An array-oriented I/O library for out-of-core computations. In *Proc. IEEE Conf. Frontiers of Massively Parallel Computing Frontiers'96*, pages 196 – 204, 1996.

[12] J. Nieplocha, R. J. Harrison, and R. J. Littlefield. Global Arrays: A nonuniform memory access programming model for high-performance computers. *The Journal of Supercomputing*, 10(2):169 – 189, 1996.

[13] E. J. Otoo and T. H. Merrett. A storage scheme for extendible arrays. *Computing*, 31:1–9, 1983.

[14] M. Ouksel and P. Scheuermann. Storage mappings for multidimensional linear dynamic hashing. In *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 90 – 105, Atlanta, March 1983.

[15] A. L. Rosenberg. Allocating storage for extendible arrays. *J. ACM*, 21(4):652–670, Oct 1974.

[16] D. Rotem and J. L. Zhao. Extendible arrays for statistical databases and OLAP applications. In *8th Int'l. Conf. on Sc. and Stat. Database Management (SSDBM '96)*, pages 108–117, Stockholm, Sweden, 1996.

[17] S. Sarawagi and M. Stonebraker. Efficient organization of large multidimenional arrays. In *Proc. 10th Int'l. Conf. Data Eng.*, pages 328 – 336, Feb 1994.

[18] Kent E. Seamons and Marianne Winslett. Physical schemas for large multidimensional arrays in scientific computing applications. In *Proc. 7th Int'l. Conf. on Scientific and Statistical Database Management*, pages 218–227, Washington, DC, USA, 1994. IEEE Computer Society.

[19] T. Tsuji, A. Isshiki, T. Hochin, and K. Higuchi. An implementation scheme of multidimensional arrays for molap. In *DEXA, Workshop*, pages 773 – 778, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[20] T. Tsuji, H. Kawahara, T. Hochin, and K. Higuchi. Sharing extendible arrays in a distributed environment. In *IICS '01: Proc. of the Int'l. Workshop on Innovative Internet Comput. Syst.*, pages 41–52, London, UK, 2001. Springer-Verlag.

[21] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proc. ACM-SIGMOD Conf.*, pages 159–170, 1997.

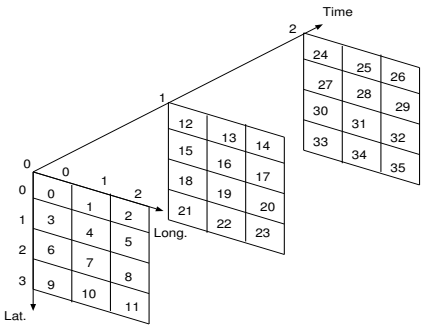
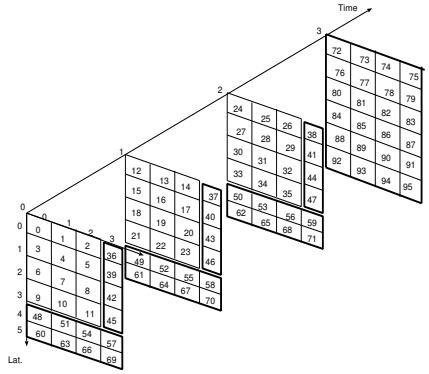
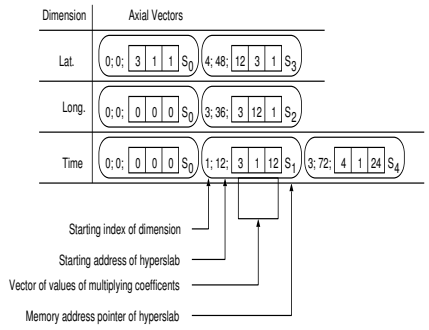


Figure 1: A Lat., Long. and Time 3-D model of the data

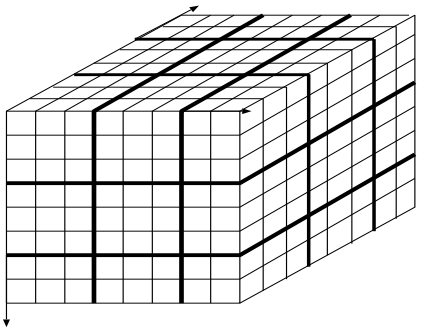


(a) A storage allocation a 3-dimensional array varying in both space and time

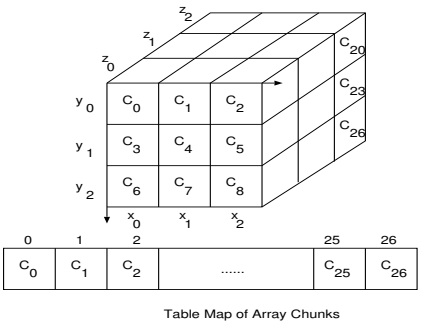


(b) The corresponding 3 distinct Axial-Vectors

Figure 2: An example of the storage allocation scheme of a 3-D extendible array

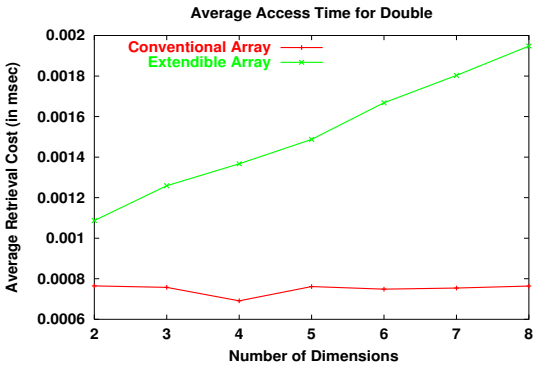


(a) A 3-dimensional array partitioned into chunks

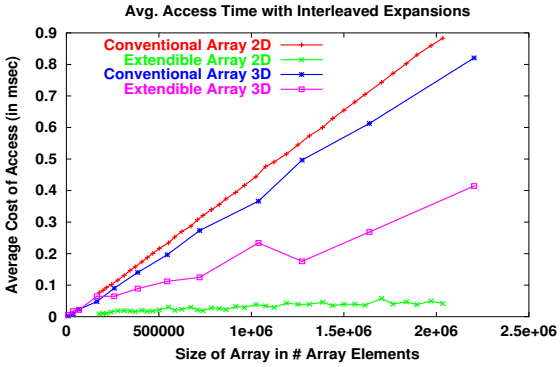


(b) Chunked Array with its Table Map

Figure 3: An allocation scheme of a 3-D sparse extendible array

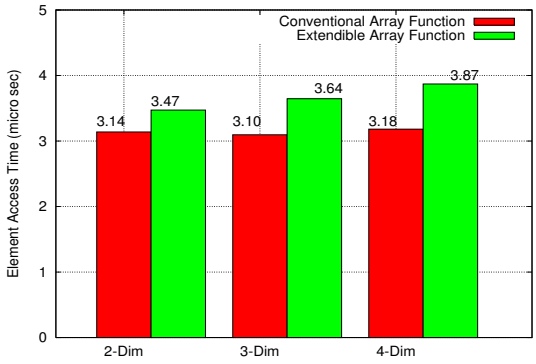


(a) Element access costs for static array

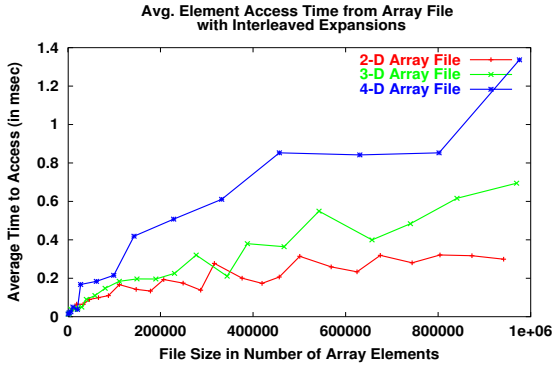


(b) Element access costs with interleaved extensions

Figure 4: Average time to access an element in an array of type double



(a) Access cost from a static array file



(b) Access cost from a file with interleaved extensions

Figure 5: Average time to access an element from an extendible array file of type double