© 2008 Xiaolei Li

MULTIDIMENSIONAL ANALYSIS OF MOVING OBJECT DATA

BY

XIAOLEI LI

B.S., University of Illinois at Urbana-Champaign, 2002
M.S., University of Illinois at Urbana-Champaign, 2004

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2008

Urbana, Illinois

Doctoral Committee:

Professor Jiawei Han, Chair
Professor Marianne Winslett
Assistant Professor ChengXiang Zhai
Professor Walid G. Aref, Purdue University

Abstract

The collection of historical or real-time data on moving objects is quickly becoming a ubiquitous task. With the help of GPS devices, RFID sensors, RADAR, satellites, and other technologies, mobile objects of all sizes, whether it be a tiny cellphone or a giant ocean liner, can be easily tracked around the globe. Many fundamental problems in the database field have found their parallels in the moving object domain. They include indexing and query processing of moving objects over static or continuous queries and similarity search between moving objects. The same has happened with data mining problems as well. Clustering of moving objects is one popular topic; spatial association patterns is another.

However, even with the recent attention, there are still many unexplored areas in moving objects research. Specifically, higher semantic level problems remain mostly untouched. One example is anomaly detection. With the ever-increasing focus on video surveillance, many cities are tracking and analyzing vehicles as they move throughout the city. With the ultimate goal of automated reporting and alerting, sophisticated algorithms are needed to evaluate the moving object trajectories. Furthermore, associations with other multi-dimensional features will need to be considered as well. Another example is periodic traffic pattern detection. Everyone is familiar with rush hour traffic in big cities, but extracting and representing them in an efficient and concise manner has not been addressed.

To this end, we present our studies in this thesis. With regards to anomaly detection, we present three models to automatically detect moving object anomaly, traffic anomaly, and subspace anomaly. The last of which detects anomalies in a multidimensional space, which is often the case in real world datasets. Additionally, we also address problems that could occur due to sampling in a multidimensional space and how to summarize moving object trajectories for more efficient processing.

Table of Contents

List of Tables . . . . **vii**

List of Figures . . . . **viii**

Chapter 1 Introduction **1**

Chapter 2 Related Work **6**
- 2.1 Indexing and Query Processing 6
  - 2.1.1 Stationary Spatial Data 7
  - 2.1.2 Moving Object Data 8
- 2.2 Data Mining 10
- 2.3 Clustering 12

Chapter 3 Hot Route Detection **14**
- 3.1 Solution Overview 16
- 3.2 Traffic Behavior in Road Networks 18
  - 3.2.1 Traffic Complexity 18
  - 3.2.2 Splitting/Joining Hot Routes 18
  - 3.2.3 Overlapping Hot Routes 19
  - 3.2.4 Slack within Hot Routes 20
- 3.3 Density-based Hot Route Extraction 21
  - 3.3.1 Traffic-Density Reachability 21
  - 3.3.2 Discovering Hot Routes 24
  - 3.3.3 Algorithm 25
  - 3.3.4 Determining Parameters 29
- 3.4 Experiments 30
  - 3.4.1 Data Generation 30
  - 3.4.2 Extraction Quality 31
  - 3.4.3 Efficiency 34

Chapter 4 Sampling in Multidimensional Data **37**
- 4.1 Definitions 40
  - 4.1.1 Data Cube Definitions 40

4.1.2 Problem Definition 42
4.1.3 Confidence Interval 42
4.2 The Sampling Cube Framework 43
4.2.1 Materializing the Sampling Cube 43
4.2.2 Boosting Confidence for Small Samples 44
4.3 The Sampling Cube Shell 51
4.3.1 Building the Sampling Cube Shell 52
4.3.2 Query Processing 56
4.4 Performance Evaluations 59
4.4.1 Shell Construction Efficiency 60
4.4.2 Query Effectiveness 61

Chapter 5 Moving Object Outliers 68
5.1 Key Insights 70
5.1.1 Motif-based Feature Space 70
5.1.2 Multi-Resolution Feature Hierarchies 72
5.2 Framework 72
5.2.1 Motif Extractor 73
5.2.2 Feature Generator 75
5.2.3 Classification 80
5.3 Experiments 83
5.3.1 Real Data 84
5.3.2 Synthetic Data 86

Chapter 6 Subspace Outliers in Multidimensional Data 93
6.1 Problem Definition 94
6.1.1 Preliminaries 94
6.1.2 The Anomaly Search Problem 96
6.1.3 Ranking Anomalies in Data Cube 97
6.2 Mining Top-K Anomalies in Data Cubes 98
6.2.1 Retrieving $\sigma_p(R)$ 99
6.2.2 Selecting Candidate Subspaces 100
6.2.3 Discovering Top-K Anomaly Cells 104
6.2.4 Iterative Search 104
6.3 Experiments 105
6.3.1 Real World Data 105
6.3.2 Synthetic Data 108

Chapter 7 Temporal Traffic Outliers 110
7.1 Road Segment Representation 112
7.1.1 Input Data 112
7.1.2 Feature Space 113
7.2 Temporal Outlier Detection 114

    - 7.2.1 Temporal Neighborhood Vector 116
    - 7.2.2 Temporal Vector Update Rules 118
    - 7.2.3 Temporal Outlier Scoring 121
  - 7.3 The TOD Framework 122
    - 7.3.1 Complexity 123
    - 7.3.2 Setting Parameters 123
    - 7.3.3 Other Applications 124
  - 7.4 Experiments 126
    - 7.4.1 Outliers 126
    - 7.4.2 Naïve Outlier Detection 128
    - 7.4.3 Efficiency 129

Chapter 8 Conclusion 132

References 135

Author's Biography 143

List of Tables

4.1 Two models of OLAP application 38
4.2 Sample survey data 39
4.3 Query expansion experimental results 67

6.1 Input market segment data 95
6.2 Time Anomaly Matrix 101
6.3 Attribute value AL scores 103
6.4 Run times of trend anomaly query with low dimensional data ($10 \leq |R| \leq 11$) 106
6.5 Magnitude anomaly query run times 106

7.1 Sample Feature Space 114
7.2 Edge Similarity Values for Day 1 117
7.3 Day 1–3 Temporal Neighborhood Vectors 120
7.4 Outlier Scores 122

List of Figures

1.1 Thesis Framework 4

3.1 Snapshot of San Francisco Bay Area traffic 15
3.2 Spectrum view of FlowScan and alternative methods 17
3.3 Splitting hot routes: $A \to B \to C$ and $A \to B \to D$ 19
3.4 Overlapping hot routes: $A \to B$ and $B \to C$ 20
3.5 Slack within a hot route: $A \to B$ 20
3.6 1-neighborhood of r. 22
3.7 Two sides of the same street 22
3.8 Route traffic density-reachable 24
3.9 Indexing structures of FlowScan 28
3.10 Hot routes in San Francisco data map 31
3.11 Hot routes in San Joaquin data map 32
3.12 Splitting hot routes 34
3.13 Overlapping hot routes 35
3.14 Efficiency with respect to number of objects 35
3.15 Disk I/O improvement of clustered index on Edge Table 36

4.1 Cuboid lattice 41
4.2 Query expansion within sampling cube 46
4.3 Sampling Cube Shell 52
4.4 Sampling cube shell construction example 57
4.5 `(Age, Occupation)` query 59
4.6 Materialization time vs. dimensionality 61
4.7 Materialization time vs. number of tuples 61
4.8 Query accuracy vs. *shell_size* for average household income dataset 64
4.9 Query accuracy vs. query dimensionality for average household income dataset 65

5.1 Motif representation 71
5.2 ROAM Framework 73
5.3 Two objects moving with the same trajectory. 75
5.4 Two sample motif-attribute hierarchies 79
5.5 Path of ship Point Sur from 16:00 to 24:00 on 8/23/00 starting at point $(0, 0)$. 84

5.6 Extracted motifs from MBARI data. 85
5.7 Effect of ω on classification accuracy. 85
5.8 Effect of number of motifs on classification accuracy. 86
5.9 GSTD $N2000B200M30$: Accuracy with respect to difference in variance. . . 87
5.10 $N4kB200A3S5.0L20$: Accuracy with respect to number of motifs. 88
5.11 $N4kB200A3S5.0L20$: Accuracy with respect to length of motif-trajectories. . 89
5.12 $N5kB100M20A3L20$: Accuracy with respect to standard deviation. 89
5.13 $N500B100M20A3S25L20$: Accuracy with respect to β. 90
5.14 $N15kB500A3S25L20$: Accuracy with respect to number of motifs. 90
5.15 GSTD: $B200M20$: Efficiency with respect to number of trajectories. 91
5.16 $N20000B1000M20A3S10.0$: Efficiency with respect to length of motif-trajectories. 91
5.17 GSTD: $N2000B100M10$: Efficiency with respect to length of trajectories. . . 91

6.1 Cuboid lattice 95
6.2 SUITS Framework 99
6.3 Running time vs. number of dimensions 107
6.4 Running time vs. number of tuples 108
6.5 Running time vs. number of dimensions 109
6.6 Running time vs. length of time series 109

7.1 Historical Speed on Road Segment X 111
7.2 Historical Speed on Many Road Segments 111
7.3 Stability of Average Daily Speed and Load 114
7.4 Speed and Load Neighborhood Stability 116
7.5 Similarity-based Reward/Penalty to $v_{i,j}$ 119
7.6 TOD Data Flow 123
7.7 Temporal Similarity 125
7.8 Speed Outlier (14th St.) 127
7.9 Speed Outlier (Webster Ave.) 127
7.10 Load Outlier 127
7.11 Load Outlier 128
7.12 Non-Outlier 128
7.13 Average Speeds of I-880 130
7.14 Average Speeds of San Jose Ave. 131
7.15 Efficiency vs. Number of Days 131
7.16 Efficiency vs. Neighborhood Radius 131

8.1 Thesis Framework & Future Direction 133

Chapter 1

Introduction

In recent years, analysis of moving object data [24] has emerged as a hot topic both academically and practically. On a macro level, trajectories of airplanes or ships are being tracked constantly by either the company who runs them or the government. On a micro level, GPS devices embedded in vehicles or RFID sensors on the streets can track a vehicle as it moves throughout the city traffic grid. GPS devices in cellphones can even track an individual person as he/she walks around the city. There are many useful applications with such data. For instance, the OnStar system in General Motors vehicles notifies police of the vehicle's GPS location when a crash is detected. Google Ride Finder provides real-time location of taxi cabs in many major cities through GPS devices installed in the cars. E-ZPass sensors (using RFID technology) automatically pay tolls so traffic is not disturbed. GPS navigation systems offer driving directions in real-time. Several cellphone services provide GPS tracking of children for parents or just tracking of nearby friends. On a more aggregate level, average speeds or traffic density is used to update driving time estimates in real-time or warn police of potential problem areas.

With such data and immediate applications, there are many exciting opportunities in research. Some basic ones include index and query. They are very important and fundamental questions to tackle. But in addition to the basic real-world applications of simply asking "Where is this object?" or "Where has this object been?", there are many higher level questions one could ask for moving objects data. Example 1 shows one particular problem which is of great interest in homeland security and surveillance.

Example 1 *At any one time, there are approximately 160,000 vessels traveling in the United States' waters. They include military cruisers, private yachts, commercial liners, and so on. The US Navy and Coast Guard are constantly on the lookout for suspicious behaviors. The traditional model of surveillance has been manual inspection of RADAR. However, as the number of vessels increases and the cost of manpower increases, it is becoming unrealistic to manually examine each of these vessels and identify the suspicious ones. Thus, it is highly desirable to create automated tools that can evaluate the behavior of all maritime vessels and develop situational awareness on the abnormal ones.*

Example 1 shows an example of anomaly detection in the military. More close to home are surveillance applications. Recently in Chicago, millions of dollars have been invested to install hundreds of cameras at street corners. Though they are being used to many different purposes, one is to track the movement of vehicles. By reading the license plate number, the cameras can track a vehicle as it moves throughout the city. This tracking can be used for criminal investigations and also anti-terrorism purposes. One simple example the city gave was that if a vehicle is observed at a particularly sensitive street corners three times within a short period of time, the system would automatically raise an alarm. This is a rather simple rule; one could imagine a much more sophisticated system at work.

On a less security-focused front, imagine the following application of moving object technology.

Example 2 *Many modern-day cellphones are equipped with GPS technology that can pinpoint the user anywhere on Earth. Even without GPS, one could use triangulation of the cellphone towers to narrow down the location of the user to within a few city blocks. This effectively says that all cellphone users' locations can be tracked. For the purpose of studying and predicting consumer behavior, this is a great source of data. By knowing the location of people in various stores and malls across the city, one can very accurately measure and predict the sales of goods at all types of stores.*

The above example shows the great power of tracking moving objects. If one can predict the sales of goods at a chain of stores, one can fairly accurately predict the quarterly sales number of the company. This leads to better prediction of stock prices and much better "playing" of the stock market than conventional analysts.

Similarly, but on a much more micro level, one can imagine tracking customers inside a single store. RFID technology can be installed on shopping carts and baskets and be used to track customers as they wander the store. Such trajectories can be used to better model consumer behavior and possibly better store layouts to maximize revenue.

The above two examples give relatively high level problems in the moving object domain. Problems of this type are the focus of this thesis. Instead of asking basic questions such as "Where is object X?", we ask higher level and more aggregated questions such as "What are the behaviors of objects of type X?" In the context of previous examples, type X could be "suspicious vehicles" or "potential buyers." The much more loaded term in that question is "behaviors." Behavior can be defined in an infinite number of ways. In the context of spatiotemporal data, it could be a movement pattern in space (*e.g.*, right turn), an attribute of movement (*e.g.*, speed), or an attribute in time (*e.g.*, 10:00am). Such behaviors or patterns are easy to describe in human terms but to have an algorithm extract them automatically is another question. Furthermore, implied in our question is the term "interesting" in front of "behaviors." Since the goal is to learn more about objects of type X, uninteresting behaviors are fairly useless. For examples, knowing that many vehicles drive on the highway is common knowledge and does not add much to the study. But to define interestingness is a tricky issue. Is it something that is frequent? Does it have to be unique to type X? Depending on the application, different interestingness measures can be employed.

In this thesis, we are looking for *patterns in moving object trajectories for the purpose of extracting high level knowledge*. To this end, we present several studies that fit into an overall framework. Figure 1.1 shows the flow of data in the framework. At the top is the original raw moving object data. The middle level serves as a pre-processing step which may clean the data in some form. The bottom layer is the analysis layer which then extracts the final patterns.

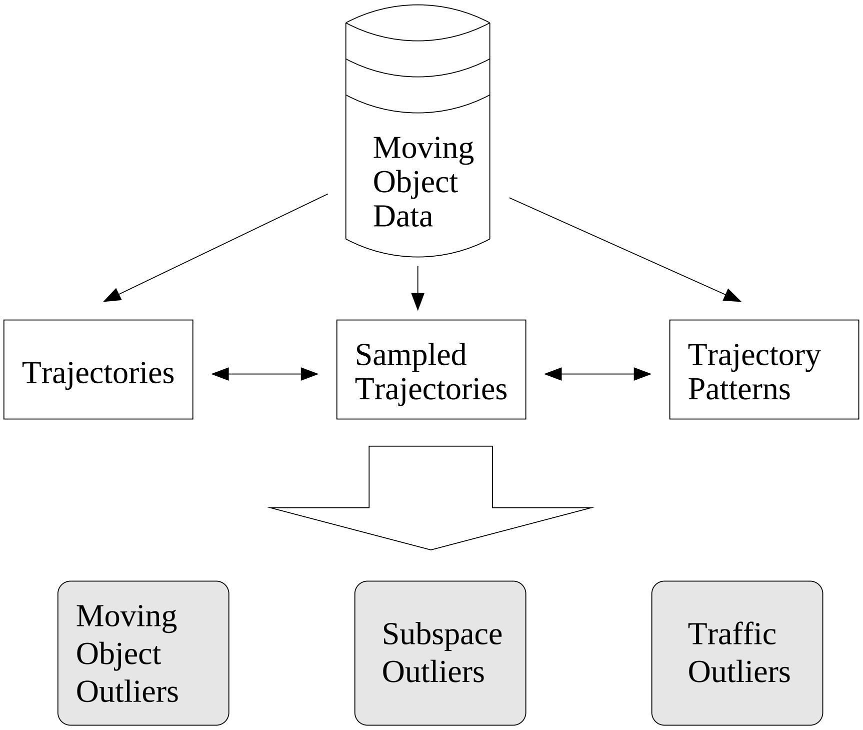


Figure 1.1: Thesis Framework

The rest of this thesis will describe the modules in Figure 1.1 in detail. First, to allow all our analysis studies to operate on a higher semantic level, Chapter 3 studies the extraction of hot routes in traffic data [51]. This step lets algorithms operate on a more aggregated information rather than individual road segments or trajectory pieces. Next, in many studies, only a sample of the population is available for study. How to deal with the lack of data in a multidimensional space [52] is addressed in Chapter 4. The next three chapters correspond to the three outlier detection modules in Figure 1.1. Moving from left to right, we first present ROAM [50], which detects moving object outliers in a free-moving environment and will be discussed in Chapter 5. Next, we consider outliers in arbitrary subspaces of a multidimensional space. Analysis of moving object data is often multidimensional. That is, there are multiple dimensions of attributes attached to the data and one can analyze them from arbitrary angles. This introduces outliers in multidimensional space [47], which is the subject of our study in Chapter 6. Lastly, we present TOD [53], which detects temporal outliers in the traffic of a road network and will be discussed in detail in Chapter 7. We conclude our studies and discuss some future work in Chapter 8.

Chapter 2

Related Work

Compared to other more mature fields, research with moving objects is still in its infancy. Much of the work is still focused on relatively fundamental, yet very important, questions such as indexing and query processing. In this section, we will review the current research in all related areas and also how they relate to our own planned work. For completeness, we will include work from the field of spatiotemporal research. Because it considers all spatial objects (not just moving ones), it can be viewed as a superset of moving objects research.

2.1 Indexing and Query Processing

A basic task people like to perform on data is indexing. Given a large amount of data, the most basic question to ask is "What is in it?" For spatiotemporal data, it is more like "Where is it?" or "Where was it at this particular time?" Questions such as these can only be answered efficiently if there is a proper index on the data.

In discussing various types of indices and query processing, we are going to divide them into two main groups. The first are *stationary objects* (either having size or not) and the second are *moving objects*. While stationary objects may seem unrelated to our work, many of the techniques in that field are quite useful. Further, because our work involves spatiotemporal data in general, it will often require analysis on stationary spatial objects. Thus, we will discuss both types of data in the following two sections.

2.1.1 Stationary Spatial Data

The R-tree index [25] was invented for the purpose of accessing points in multi-dimensional space. Though it can handle all types of spatial data whether the points have size or not. Based on the B-tree, it uses the concept of *minimum bounding rectangles.* Each non-leaf node in the tree is a bounding rectangle that spatially contains all nodes (*i.e.*, objects) in its subtree. To search for an object, one would start at the root node (the bounding rectangle that contains all objects) and progressively drill down to smaller rectangles until the query is satisfied.

Many extensions of the original R-tree have been proposed; some can handle moving objects as well. They include the R*-tree [6], R+-tree [70], X-tree [8], Hilbert R-tree [34], CR-tree [37], CT-R-tree [16], CR*-tree, Hilbert CR-tree, and more. A performance comparison can be found at [29].

In relation to our work of data mining in moving objects, such indexing algorithms can be very helpful. In some of our work [51], spatial queries are needed to find stationary objects (namely street intersections) near a particular location. Having a faster index would only make our algorithm more efficient.

In additional to basic window queries, algorithms [67, 72, 15] have also been proposed to efficiently process nearest-neighbor queries. These are well-known problems in traditional database research that has been ported to spatiotemporal research. But, the nature of spatiotemporal data has also introduced a new kind of query, one where the query (be it a region or a point) moves. A natural real-world application is to find the nearest restaurant as one moves through a city.

A naïve approach to this problem is to continuously apply the traditional stationary algorithms repeatedly as the query moves. This would surely work but could be very inefficient. For instance, if the query moves very little, the results might not change at all. By knowing this, one could save an entire query.

Several algorithms [76, 91] have been proposed to tackle this problem. In one of the earlier work [76], it was proposed to process the query by issuing the entire query segment (*i.e.*, the object's trajectory) to an R-tree instead of point-by-point. This reduces many redundant queries. The authors also introduce heuristics for pruning parts of the R-tree by calculating distances between nodes and the query segment. In [91], the concept of a *valid region* for the query location is introduced. As long as the query object remains in the valid region, there is no need to re-evaluate the query (either nearest-neighbor or window). By being aware, this greatly reduces the number of evaluations on the query.

And finally, related to index and query is data warehousing [59, 62, 46, 74, 75]. Instead of focusing on individual object's locations, queries seek aggregate information on historical data. Other studies have focused on approximate aggregate queries [73]. In it, multi-dimensional histograms are used to approximate the aggregation queries.

2.1.2 Moving Object Data

The previous section discussed indexing structures and queries for spatial objects that did not move. The more relevant scenario to our work is when do they move. One of the first work to touch on this topic was the TPR-tree [69], which is a *time-parametrized* R-tree. Like the R-tree, non-leaf nodes in the tree are bounding rectangles that contain all points in the subtree. But additionally, each rectangle also has a velocity vector. Then, depending on the setting of the time parameter, one could calculate the new position of the rectangle for some time in the future. By doing the same calculation for a possibly moving query rectangle, one can then check if the query rectangle and any object rectangle will intersect at any time. This framework has been extended [77, 68] and used [7, 17, 64, 43].

A different class of solutions for the same problem uses the idea of dual spaces. Trajectories are converted into points in a different space and indexed and queried accordingly. Some examples of this idea include algorithms which convert trajectories to points in higher dimensional spaces [39, 1, 61, 40]. One particular example is STRIPES [61]. Instead of using the R-tree and modeling linear trajectories, entire trajectories are transformed to points in a higher dimensional space. Specifically, given a moving object in d-dimensional space, the maximum velocity vector (d-dimensional vector) combined with reference position vector (d-dimensional vector) forms a new $2d$-dimensional point. These points are then stored in a multi-dimensional quadtree. And finally, there is work that transforms trajectories to points in a lower dimensional space. The B^x-tree [32, 55] and B^{dual}-tree [86] are examples in which the moving objects are converted to 1-dimensional values using space-filling curves and then stored in a B$^+$-tree.

With continuously moving objects, many traditional queries can be converted to "continuous" as well. More specifically, instead of just executing the query once and getting the results, the query executes continuously and the results are updated as the objects moves. First, consider the case where the movements of the objects are known a priori. Nearest neighbor queries on them have been studied in [30, 31, 66]. In general, one might not know any information before evaluating the query. Some studies [32, 57, 58, 89, 83] basically re-evaluate the query periodically using the new moved data. A recent work [28] also works on this problem without making any assumptions. It extends the idea of valid regions [91] to both the query and the objects themselves. One of the goals of the framework is reduce the amount of communication between objects (both the querying object and the queried objects) and the central database, which keeps track of queries and moving objects. In this framework, each moving object is aware of a *safe region*. As long as the object moves inside this region, it will not affect the result of the queries that the central database is currently handling. But, if the object moves outside the region, it will update the central server of its new location and the central database will take care of updating query results and also new safe regions. By doing this, not only is the number of query calculations reduced, the amount of communication between the object and the central database is also reduced.

One particular type of moving object data that is of interest to us is moving objects on *road networks* or *constrained networks*. New techniques [63, 20, 4, 13] are needed to take advantage of the constraints. In [63], a dimensionality reduction technique is used to reduce the 3D problem (*i.e.*, 2D spatial plus time) into lower-dimensional problems. The FNR-tree [20] also uses a dimensionality reduction technique. A 2D R-tree is used to index all the edges in the road network. Then, the objects within each object are further indexed by a different 1D R-tree. In other words, there is a main 2D R-tree for indexing the network's edges and many different 1D R-trees in the leaves of the main tree for indexing objects. Experimental results show that this indexing scheme is more efficient in terms of space utilization and search speed than a 3D R-tree. More recently, the idea of an adaptive unit (AU) was introduced in [13]. A single AU is similar to a MBR in the context of the TPR-tree [69]. However, in the traditional TPR-tree, objects with very different movement patterns may exist in a single MBR. If they happen to move in opposite directions, it will expand the MBR very quickly and require many updates. In contrast, an AU will contain only objects with similar movement patterns; it can be viewed as a moving object cluster. The AUs are then indexed by an R-tree, similar to the original TPR-tree. The authors show empirically that the AU-index is much more efficient with respect to updates than the TPR-tree.

2.2 Data Mining

Data mining in the spatiotemporal domain has also received some attention. One of the most immediate extensions from traditional data mining is the discover of *frequent patterns*. One of such studies finds frequent sequences in spatiotemporal data [79]. In their setting, data consists of sequences of events at spatial locations. For example, locations could be cities in the United States, and the sequential data could be temperature recordings. The goal is then to find frequent sequential patterns in the data. The authors tackle this problem in a two-step process. First, a depth-first mining algorithm is proposed to discover sequential patterns at individual locations. The algorithm uses the enumeration lattice structure proposed in SPADE to efficiently discover maximal frequent sequences. Then, to incorporate the spatial aspect of the data, the algorithm examines the hierarchical nature of the locations. In the United States for example, one could roll the locations up to the county or state level. This hierarchy information is pushed down to the mining algorithm and an apriori-like property is found to aid in the pruning of the algorithm.

Another popular problem with spatiotemporal data is *co-location* mining. This is very similar to the traditional association mining. The biggest difference is that the predicates in the pattern can be spatial or temporal or both. For example, in traditional association mining, itemset A might be found to be frequently associated with B in the transaction database. In co-location mining, the constraint of A being near B (where A and B are spatiotemporal events) could be enforced. This neighborhood constraint add to the complexities of the traditional mining algorithm. Some studies [41, 71, 88] have modified traditional association mining algorithms to incorporate the spatiotemporal information. For instance, the apriori algorithm has been extended [71, 88] to join smaller co-location patterns together. [88] performs a more complicated process by looking additionally at partial joins. [93] is a recent work that quickly finds co-location patterns by using multiway joins.

A problem related to nearest-neighbor queries is the computation of *influential* sites [81, 19]. The influence of a particular site is measured by how many other sites has it as its nearest neighbor. This has also been called the reverse nearest-neighbor problem. Though the concept of the problem is not unique to spatial data, the solution uses spatial solutions such as the R-tree. Related to influential queries are preference queries [87]. A preference query is like a regular query except that it considers extra features to the data. For instance, a query asking for a ranked list of houses in a neighborhood might uses a combinations of features (*i.e.*, size, price, school quality, etc) in its ranking. This combination of spatial information and other multi-dimensional features pose new challenges.

2.3 Clustering

Clustering of moving objects and trajectories is a useful problem in many contexts. Though very similar in goal, these two problems have a subtle difference. In moving object clustering, it is implied that the objects within a single cluster move with the same or similar trajectory *together*. That is, they travel in a tight group. In trajectory clustering, only the final resultant trajectory is considered. One could say that the temporal component is ignored. Objects can move in arbitrary speeds or times, but as long as their final trajectories are similar, they will be clustered together. Some examples of moving object clustering include [36, 21, 11, 33]. In [33], objects are grouped using a traditional clustering algorithm at each time snapshot and then linked together over time to form moving clusters. The assumption is that clusters will be stable across short time periods.

Trajectory clustering has also produced some interesting results [90, 12, 44]. In one such work [12], trajectories are converted to line segments and a similarity function is defined between line segments. This similarity function is then used to mine for sequences of line segments, which grows to the eventual pattern. A similar idea is employed in another work [44]. Though the goal in that work is slightly different: trajectory clustering vs. pattern discovering, the end result is quite similar. One could view the clusters as being the patterns. In [44], a minimum description length approach is taken to describe a trajectory as a sequence of lines. The lines are then fed into a density-based clustering algorithm to produce the final clusters. The similarity function used in clustering is like the one in [12] where the difference in angle and length factor into the final measure. Similarity functions between line segments or whole trajectories have also been explored in traditional similarity search [80, 14]. More recently, another work [22] attempted to solve the problem of discovering trajectory patterns.

In contrast to previous work, this one is more focused on higher level concepts. For instance, instead of discovering a pattern involving a specific spatial location, a general location (*e.g.*, airport) is found. Such general locations are called Regions-of-Interest or RoI's. Then, frequent movement patterns between the RoI's are discovered.

Finally, there is the area of trajectory modeling [54, 42]. Here, the goal is very different. Instead of just producing frequent patterns, an entire model is constructed that describes how the object will move under various situations. A Markov model is typically used for such problems. Also, each model is typically used to describe just a single object. Multiple objects with slightly varying transition probabilities can not be efficiently captured within a single model. This makes it unsuitable for hundreds or thousands of objects. Lastly, it is not clear that moving objects satisfy the Markovian property. Without it, the models will not be very effective.

Chapter 3

Hot Route Detection

As previously mentioned, clustering is a very attractive problem in moving objects research. It is a very natural question to ask about moving objects and algorithms can leverage much of the current work in traditional clustering. A related question is to ask what are the *hot routes* or the general traffic flow pattern in a city road network. The set of hot routes offers direct insight into the city's traffic patterns. City officials can use them to improve traffic flow. Store owners and advertisers can use them to better position their properties. Police officials can use them to maximize patrol coverage.

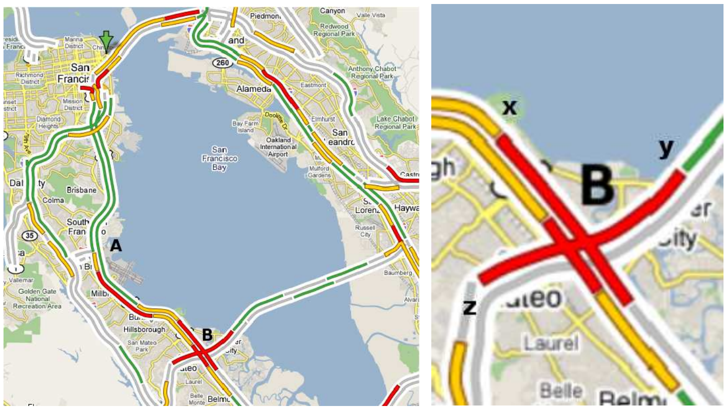
This is a challenging problem due to the complex nature of the data. If objects traveled in organized clusters, it would be straightforward to use a clustering algorithm to find the hot routes. But, in the real world, objects move in unpredictable ways. Variations in speed, time, route, and other factors cause them to travel in rather fleeting "clusters." These properties make the problem difficult for a naive approach. To this end, we propose a new density-based algorithm named FlowScan[51]. Instead of clustering the moving objects, road segments are clustered based on the density of common traffic they share. We implemented FlowScan and tested it under various conditions. Our experiments show that the system is both efficient and effective at discovering hot routes.

Example 3 *Figure 3.1(a) shows live traffic data*[1] *in the San Francisco Bay Area on a weekday at approximately 7:30am local time. Different colors show different levels of congestion (e.g., red/dark is heavy congestion).* `511.org` *in the Bay Area gathers such data in real-*

---

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

time from RFID transponders located inside vehicles[2]. *A likely hot route in Figure 3.1(a) is* $A \rightarrow B$ *(i.e., highway CA-101).* A *is near the San Francisco International Airport.* B *is near the San Mateo Bridge. Figure 3.1(b) shows a closeup view of location* B. *Three additional locations* x, y, *and* z *are shown. Without actually observing the flow of traffic, it is unclear whether* $y \rightarrow x$ *is a hot route, or* $y \rightarrow z$, *or* $x \rightarrow z$. FlowScan *aims to solve this problem.* ■



(a) (b)

Figure 3.1: Snapshot of San Francisco Bay Area traffic

At first glance, this may seem like an easy problem. A quick look at Figure 3.1(a) shows the high traffic roads in red. With some domain knowledge, we know that San Francisco, Oakland, and other densely populated regions are likely to be sources and destinations of traffic since many people live and work there. However, such domain knowledge is not always available. Additionally, real world traffic is a very *complex* data source. Objects do not travel in organized clusters. Two objects traveling from the same place to another place may take just slightly different routes at slightly different speeds and times. Random traffic conditions, such as a traffic accident or a traffic light, can cause even more deviations. Furthermore, hot routes do not have to be disjoint. Highways or major roads are popular pathways and

---

[2] http://www.bayareafastrak.org

several hot routes can share them. As a result, the mining algorithm must be robust to the variations within a hot route and amongst a set of hot routes.

We now state our problem as follows: *Given a set of moving object trajectories in a road network, find the set of hot routes.* A road network is represented by a graph $G(V, E)$. E is the set of directed edges, where each one represents the smallest unit of road segment. V is the set of vertices, where each one represents either a street intersection or important landmark. T is the set of trajectories, and each trajectory consists of an ID (tid) and a sequence of edges that it traveled through: $(tid, \langle e_1, \ldots, e_k \rangle)$, where $e_i \in E$. Objects can only move on E and must travel the entirety of an edge. T is assumed to be collected from a similar time window; otherwise, different time windows might blur meaningful hot routes.

Informally, a *hot route* is a general path in the road network which contains heavy traffic. It represents a general flow of the objects in the network. Formally, it is a sequence of edges in G. The edges need not to be adjacent in G, but they should be near each other. Further, a sequence of edges in a hot route should *share a high amount of traffic between them.*

3.1 Solution Overview

FlowScan extracts hot routes using the density of traffic on edges and sequences of edges. Intuitively, an edge with heavy traffic is potentially a part of a hot route. Edges with little or no traffic can be ignored. Also, two near-by edges that share a high amount of traffic between them are likely to be a part of the same hot route. This implies that the objects traveled from one edge to the other in a sequence. And lastly, a chain of such edges is likely to be a hot route.

We also list some possible alternative methods from related fields.

Alternative Method 1: Moving object clustering [21] discovers groups of objects that move together. The trajectory of each cluster can be marked as a hot route. We call this class of approaches AltMoving.

Alternative Method 2: Simple graph linkage is another possible approach. One could gather all the edges in G with heavy traffic and connect them via their graph connectivity. Then, each connected component is marked as a hot route. We call this class of approaches AltGraph.

Alternative Method 3: Trajectory clustering [44] discovers groups of similar sub-trajectories from the whole trajectories of moving objects. Each resultant cluster is marked as a hot route. We call this class of approaches AltTrajectory.

FlowScan and the three alternative methods offer very different approaches to the same data. One could view them in a spectrum. At one end of the spectrum are AltMoving and AltTrajectory where attention is paid to the individual objects. This is helpful in problems where the goal is to identify behaviors of individuals. At the other end of the spectrum is AltGraph where attention is paid to the aggregate. That is, objects' trajectories are aggregated into summaries and analysis is performed on the summary. This is helpful in problems where the goal is to learn very general information about the data. FlowScan can be viewed as an intermediate between these two extremes. The behaviors of the individuals (specifically, the common traffic between sequences of edges) are retained and affect high-level analysis about aggregate behavior.

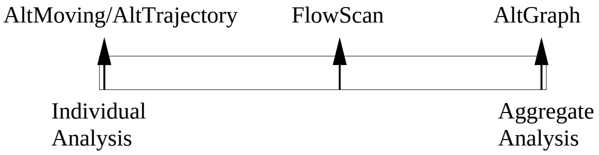


Figure 3.2: Spectrum view of FlowScan and alternative methods

3.2 Traffic Behavior in Road Networks

In this section, we list some common real world traffic behaviors and examine how FlowScan and the alternative approaches can handle them.

3.2.1 Traffic Complexity

A major characteristic of real world traffic is the amount of complexity. Instead of neat clusters, objects travel with different speeds and times even when they are on the same route. For example, in a residential neighborhood, many people leave for work in the morning and travel to the business district using approximately the same route. However, it is very unlikely that a group will leave at the same time and also travel together all the way to their destination. Various events (*e.g.*, traffic light) can easily split them up.

Algorithms in the AltMoving class will not work very well with such complex data. Clusters in the technical sense only last for a short period of time or short distance. The same is true for AltTrajectory if speed/time is encoded into the trajectories. These algorithms lack *aggregate analysis* and as a result, they are likely to find too many short clusters and miss the overall flow. FlowScan connects road segments by the amount of traffic they share. So even if the objects change slightly or if the objects do not travel in compact groups, the amount of common traffic between consecutive edges in a hot route will still be high.

3.2.2 Splitting/Joining Hot Routes

Figure 3.3 shows a sample city traffic grid. The shade on each road segment indicates the amount of traffic on it; the darker the shade, the heavier the traffic. Suppose the two correct hot routes are $A \rightarrow B \rightarrow C$ and $A \rightarrow B \rightarrow D$. Figure 3.3 shows a splitting of traffic at node B: some objects which moved from A to B go to C while others go to D. There are also other objects which move from C to D and vice-versa. This situation is very common in real world traffic. B could be the location in Chicago where I-90/94 splits into I-90 and I-94.

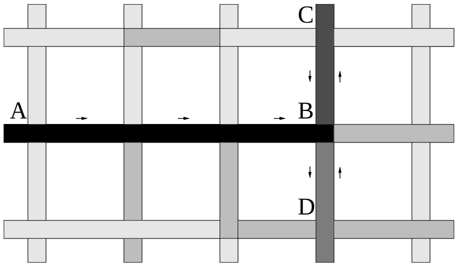


Figure 3.3: Splitting hot routes: $A \rightarrow B \rightarrow C$ and $A \rightarrow B \rightarrow D$

With AltGraph, since all edges from A to C and D are connected, they would be incorrectly identified as a single big hot route. Notice that there is no *individual analysis* in AltGraph. With AltTrajectory, $A \rightarrow B$ and $B \rightarrow C$ will not be joined together because the physical similarity between them is low (*i.e.*, hard left turn). Likewise for $A \rightarrow B \rightarrow D$. This flaw exists for the joining of traffic as well. In other words, if the arrows in Figure 3.3 were reversed, it would illustrate the problem of two hot routes joining at B.

In FlowScan, edges $B \rightarrow C$ and $B \rightarrow D$ will not be connected directly because they do not share any traffic. This is because physically, objects have to choose between the two edges and cannot travel on both. Further, $A \rightarrow B$ will be connected to both $B \rightarrow C$ and $B \rightarrow D$ because it shares traffic with both.

3.2.3 Overlapping Hot Routes

In addition to splits or joins, two hot routes may overlap each other. Figure 3.4 shows an example with two distinct hot routes: $A \rightarrow B$ and $B \rightarrow C$. Situations like this are common. Suppose B is a parking garage used by nearby residents during the night and incoming workers during the day. In the morning, residents drive out of the parking garage ($B \rightarrow C$) and other people arrive from various locations to park ($A \rightarrow B$).

Consider how AltGraph will handle this situation. Since $A \rightarrow B$ and $B \rightarrow C$ are connected in G at node B, the two hot routes will be joined together incorrectly. This is due to the

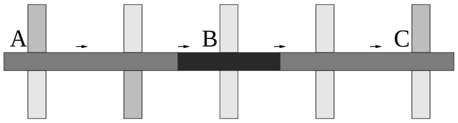


Figure 3.4: Overlapping hot routes: $A \rightarrow B$ and $B \rightarrow C$

lack of *individual analysis* on the edges during linking. The same happens with AltTrajectory, though for a different reason. $A \rightarrow B$ and $B \rightarrow C$'s shapes are similar and will be clustered together. With FlowScan, consecutive edges within a hot route must share a minimum number of common objects. If such edges were parts of different hot routes, this condition will not be satisfied, and thus, a single erroneous hot route will not be formed.

3.2.4 Slack within Hot Routes

Figure 3.5 shows a hot route with some slight slack. A hot route exists from A to B in the grid. At the intermediate locations, objects are faced with different choices in order to reach B. Suppose the choices are essentially equivalent in terms of distance and speed and that traffic is split equally between them.

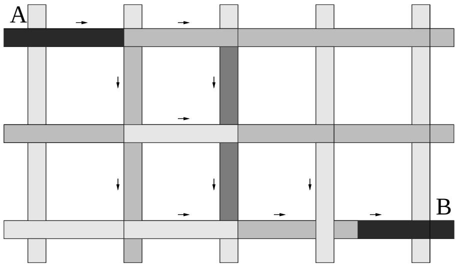


Figure 3.5: Slack within a hot route: $A \rightarrow B$

Consider how AltMoving will handle this deviation. Suppose the distance between the equivalent paths is larger than the maximum intra-cluster distance. This will cause the cluster at A to break into several smaller clusters when it reaches B. Next, consider AltGraph. Suppose the partitioning of traffic reduced the density on the intermediate edges to be below the "heavy" threshold. This would break the graph connectivity condition and miss the hot route from A to B. A similar error could occur with AltTrajectory if the traffic becomes too diluted between A and B or if the shapes become too dissimilar. With FlowScan, edge connectivity in G is not a required condition. Edges are connected in the hot route if they share common traffic and if they are near each other. Here, as long as A is within a given distance from B, the hot route will remain intact.

3.3 Density-based Hot Route Extraction

In this section, we will give formal definitions of FlowScan, which uses traffic density information in road networks to discover hot routes.

3.3.1 Traffic-Density Reachability

Definition 1 (Edge Start/End) *Given a directed edge r, let the $start(r)$ be the starting vertex of the edge and $end(r)$ be the ending vertex.*

Definition 2 (ForwardNumHops) *Given edges r and s, the number of forward hops between r and s is the minimum number of edges between $end(r)$ and $end(s)$ in G. It is denoted as $ForwardNumHops(r, s)$.*

Recall that G is directed. This implies that an edge that is incident to $start(r)$ in G will not have a $ForwardNumHops$ value of 0 unless it is also incident to $end(r)$.

Definition 3 (*Eps*-neighborhood) *The* Eps-neighborhood *of an edge r, denoted by $N_{Eps}(r)$, is defined by $N_{Eps}(r) = \{s \in E \,|\, ForwardNumHops(r, s) \leq Eps\}$ where $Eps \geq 0$.*

The *Eps*-neighborhood of r contains all edges that are within *Eps* hops away from r, *in the direction* of r. Semantically, this captures the flow of traffic and represents where objects are within *Eps* hops after they *exit* r. Figure 3.6 shows the 1-neighborhood of edge r.

Note that having the forward direction in the *Eps*-neighborhood makes the relation non-symmetric. In Figure 3.6 for example, the two edges in the circle are in the 1-neighborhood of r, but r is not in the 1-neighborhood of either of them. In fact, the only time when the *Eps*-neighborhood relation is symmetric is when two edges form a cycle within themselves. This is usually rare in road networks with one exception, and that is when one considers two sides of the *same* street. Figure 3.7 shows an example. In it, r_0 is in the 1-neighborhood of r_1 and vice-versa, because they form a cycle. This will happen for all two-way streets in the network. Though typically, an object will not travel on both sides of the same street within a trajectory. It could only happen with U-turns or if one end of the street is a dead end.

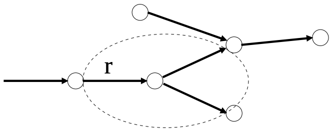


Figure 3.6: 1-neighborhood of r.

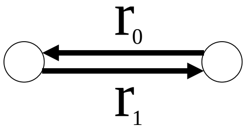


Figure 3.7: Two sides of the same street

We did not choose a spatial distance function (*e.g.*, Euclidean distance), because the number of hops better captures the notion of "nearness" in a transportation network. Consider a single road segment in the transportation network. If it were a highway, it might be a few kilometers long. But if it were a city block in downtown, it might be just a hundred meters. However, since an object has to travel the entirety of that edge, the two adjacent edges are the same "distance" apart no matter how long physically that intermediate edge is.

Definition 4 (Traffic) *Let traffic(r) return the set of trajectories that contains edge r. Recall trajectories are identified by unique IDs.*

Definition 5 (Directly traffic density-reachable) *An edge s is* directly traffic density-reachable *from an edge r wrt two parameters, (1) Eps and (2) MinTraffic, if all of the following hold true.*

1. $s \in N_{Eps}(r)$

2. $|\mathit{traffic}(r) \cap \mathit{traffic}(s)| \geq \mathit{MinTraffic}$

Intuitively, the above criteria state that in order for an edge s to be directly traffic density-reachable from r, s must be near r, and $traffic(s)$ and $traffic(r)$ must share some non-trivial common traffic. The "nearness" between the two edges is controlled by the size of the *Eps*-neighborhood. This directly addresses the slack issues in Section 3.2.4. As long as the slack is not larger than *Eps*, two edges will stay directly connected via this definition.

The second condition of two edges sharing traffic is intuitive. It is also at the core of FlowScan. The flaw of methods in the AltGraph class is that aggregation on the edges has erased the identities of the objects. As a result, two edges with high traffic on them and near each other will look the same regardless if they actually *share* common traffic. By having the second condition rely on the common traffic, one can get a better idea of how objects actually move in the road network.

Directly traffic density-reachable is not symmetric for pairs of edges because the *Eps*-neighborhood is not symmetric. Though, for the same reasons that two edges might be in the *Eps*-neighborhood of each other, two edges could be directly traffic density-reachable from each other.

Definition 6 (Route traffic density-reachable) *An edge s is* route traffic density-reachable *from an edge r wrt parameters Eps and MinTraffic if:*

1. There is a chain of edges r_1, r_2, …, r_n, $r_1 = r$, $r_n = s$, and r_i is directly traffic density-reachable from r_{i-1}.

2. For every Eps consecutive edges (i.e., r_i, r_{i+1}, …, r_{i+Eps}) in the chain, $|\mathit{traffic}(r_i) \cap \mathit{traffic}(r_{i+1}) \cap \ldots \cap \mathit{traffic}(r_{i+Eps})| \geq \mathit{MinTraffic}$.

Definition 6 is an extension of Definition 5. It states that two edges are route reachable if there is a chain of directly reachable edges in between and that if one were to slide a window across this chain, edges inside every window share common traffic. The sliding window directly address the overlapping behavior as described in Section 3.2.3. At the boundaries of two overlapping hot routes, the second condition will break down and thus break the overlapping hot routes into two. The *Eps* parameter is being reused here to control the width of a sliding window through the chain. The reuse is justified because their semantics are similar, but one could just as well use a separate parameter.

The reason for using a sliding window is based on our observation that a trajectory can contribute to only a portion of a hot route. This better matches real world hot route behavior. For example, a hot route exists from the suburb to downtown in the morning. Figure 3.8 shows an illustration. However, most people do not travel the entirety of the hot route. More often, they live and work somewhere in between the suburb and downtown. But in the aggregate, a hot route exists between the two locations.

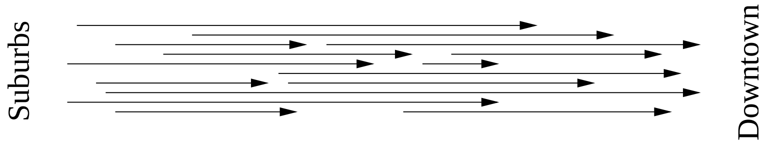


Figure 3.8: Route traffic density-reachable

3.3.2 Discovering Hot Routes

The hot route discovery process follows naturally from Definition 6. It is an iterative process. Roughly, one starts with a random edge, expands it to a hot route, and repeats until no more edges are left. The question is then with which edge(s) should each iteration begin. To this end, we introduce the concept of a **hot route start**, which is the first edge in a hot route. Intuitively, an edge is a hot route start if none of its preceding directly traffic density-reachable neighbors are part of hot routes.

Definition 7 (Hot Route Start) *An edge r is a* hot route start *wrt MinTraffic if the following condition is satisfied.*

$$\left|\, \mathit{traffic}(r) \setminus \bigcup \left\{\mathit{traffic}(x)\right\} \,\right| \geq \mathit{MinTraffic}$$

where $\{x \mid \mathit{end}(x) = \mathit{start}(r) \wedge |\mathit{traffic}(x)| \geq \mathit{MinTraffic}\}$.

The question is whether all hot routes begin from a hot route start. The following lemma addresses this.

Lemma 1 (Hot Route Start) *Hot routes must begin from a hot route start.*

Proof: There are two ways for a hot route to begin on an edge r. The first way is when *MinTraffic* or more objects start their trajectory at r. In this case, none of these objects will appear in $\mathit{start}(r)$'s adjacent edges because they simply did not exist then. As a result, the set difference will return at least *MinTraffic* objects and thus marking r as a hot route start. The second way is when *MinTraffic* or more objects converge at r from other edges. The source of traffic on r is exactly the set of edges adjacent to $\mathit{start}(r)$. Suppose one of these edges, x, contains more than *MinTraffic* objects on it. In this case, x is part of another hot route, and the objects that moved from x to r should not contribute to r. However, if it does not contain more than *MinTraffic* objects, it cannot be in a hot route and its objects are counted towards r. If more than *MinTraffic* objects are counted towards r, then it is the start of a hot route. ■

3.3.3 Algorithm

Definitions 6 and 7 form the foundation of the hot route discovery process. A simple approach could be to initialize a hot route to a hot route start and iteratively add all route traffic density-reachable edges to it. By repeating this process for all hot route starts in the data, one can extract all the hot routes. The question is then how to efficiently find all route traffic density-reachable edges given an existing hot route. If new edges are added in no particular order, then one would have to search through all existing edges in the hot route at every iteration. This is very inefficient. Further, if the hot route splits, it could become tricky if it is in the middle.

To alleviate this problem, we restrict the growth of a hot route to be only at the *last* edge. A hot route is a sequence of edges so the last edge is always defined. By growing the hot route at the end one edge at a time, only the *Eps*-neighborhood of the last edge needs to be extracted. This is much more efficient than extracting the *Eps*-neighborhoods of all edges in the hot route. This is still a complete search because all possible reachable edges are examined but just with some order. It is essentially a breadth-first search of the road network. Then for each neighboring edge, the *route* traffic density-reachability condition is checked against the last few edges of the hot route (*i.e.*, window). If the condition is satisfied, the edge is appended to the hot route; otherwise, the edge is ignored.

Sometimes, the number of directly traffic density-reachable edges from the last edge in the hot route is larger than one. There are two causes for this. The first cause is multiple edges within one hot route. This can happen when *Eps* is larger than 0, and multiple edges of the hot route are in the *Eps*-neighborhood. This can be detected by checking to see if the $start()$'s and $end()$'s match across edges. In this case, only the *nearest* edge is appended to the hot route. The other edges will just be handled in the next iteration. The second cause is when a hot route *splits*. In this case, the current hot route is duplicated, and a different hot route is created for each split. The difference between these two cases can be detected by checking the directly traffic density-reachability condition between edges in the *Eps*-neighborhood.

The overall algorithm proceeds as follows. First, all hot route starts are extracted from the data. This is done by checking Definition 7 for every edge in G. This step has linear complexity because only individual edges with their *Eps*-neighborhoods are checked. Then, for every hot route start, the associated hot routes are extracted. Algorithm 1 shows a pseudo-code description.

Algorithm 1 FlowScan

Input: Road network G, object trajectory data T, *Eps*, *MinTraffic*.
Output: Hot routes R

```
Initialize $R$ to $\{\}$
Let $H$ be the set of hot route starts in $G$ according to $T$
for every hot route start $h$ in $H$ do
    $r$ = new Hot_Route initialized to $\langle h \rangle$
    Add Extend_Hot_Routes($r$) to $R$
end for
Return $R$
```

Procedure *Extend_Hot_Routes(hot route r)*

```
Let $p$ be the last edge in $r$
Let $Q$ be the set of directly traffic density-reachable neighbors of $p$
if $Q$ is non-empty then
    for every split in $Q$ do
        if route traffic density-reachable condition is satisfied then
            Let $r'$ be a copy of $r$
            Append split's edges to $r'$
            Extend_Hot_Routes($r'$)
        end if
    end for
else
    Return $r$
end if
```

One point of concern is efficiency. Suppose the adjacency matrix or list of the road network fits inside main memory. Then, searching the graph is quite efficient. Retrieving the list of TID's at each edge will require disk I/O but traversing the graph will not. However, suppose the adjacency matrix is too big to fit inside main memory. In this case, we introduce two additional indexing structures to help the search process. Figure 3.9 shows an illustration.

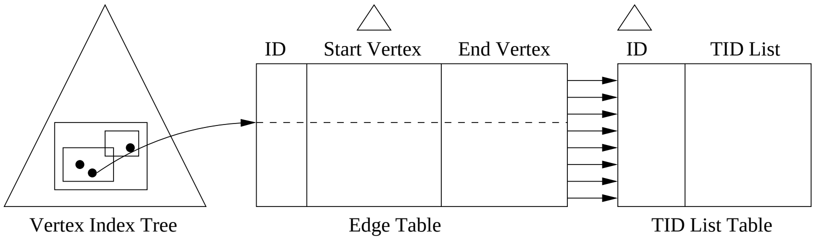


Figure 3.9: Indexing structures of FlowScan

All vertices of the road network are stored in a 2D index, *e.g.*, R-tree (Vertex Index Tree). All edges are stored on disk (Edge Table). Each edge record consists of the edge ID and its starting and ending vertices (each vertex is an (x, y) tuple). Using these two data structures, one can retrieve adjacent neighbors of an edge by querying the R-tree on the appropriate vertex and then retrieving the corresponding edges in the Edge Table. The R-tree is quite useful for finding adjacent neighbors of a specific edge since the coordinates of the adjacent neighbors tend to be close to that of the edge.

We note that the Edge Table may be accessed repeatedly to retrieve adjacent neighbors. This operation can be done more efficiently by exploiting locality. Specifically, if the physical locality of edges in the road network is preserved in the Edge Table, one can reduce the amount of disk I/Os. To this end, we create a *clustering* index on the Starting Vertex attribute of the Edge Table. Assuming that each value is 4 bytes, each edge record is then 20 bytes. Then, if a page is 4K, it will contain approximately 200 edges. The intuition is that these 200 edges will be physically close to each other in the road network. Because FlowScan traverses through *Eps*-neighborhoods, it is highly likely that an edge and its neighbors will be stored on the same page in the Edge Table. In this case, disk I/O will be reduced because the page has already been fetched.

Lemma 2 (Completeness) *The set of hot routes discovered by FlowScan is complete and unique wrt. Eps and MinTraffic.*

Proof: The above assertion is easy to see because the construction algorithm uses the definition word-for-word, specifically Definition 6, to build the hot routes. Thus, given a hot route start, the set of hot routes extending from it is guaranteed to be found. The question is more about if the set of hot route starts found is complete. Because every edge in a hot route must satisfy the *MinTraffic* condition, there must be a "first" in a sequence. The set of hot route starts is simply these "firsts." Lastly, ordering is not a factor in FlowScan, because no marking or removal is done to G. Thus, it does not matter in which order H is processed.

■

3.3.4 Determining Parameters

There are two input parameters to the FlowScan algorithm: *Eps* and *MinTraffic*. The first parameter, *Eps*, controls how lax FlowScan can be between directly reachable edges. A value of 0 is too strict since it enforces strict spatial connectivity. A small value in the range of 2–5 is usually reasonable. In a metropolitan area, this corresponds to 2–5 city blocks; and in a rural area, this corresponds to 2–5 highway exists.

As for *MinTraffic*, this is often application or traffic dependent. "Dense" traffic in a city of 50,000 people is very different from "dense" traffic in a city of 5,000,000 people. In cases where domain knowledge dictates a threshold, that value can be used. If no domain knowledge is available, one can rely on statistical data to set *MinTraffic*. It has been shown that traffic density (and many other behaviors in nature) usually obeys the power law. That is, the vast majority of road segments have a small amount of traffic, and a relative small number have extremely high density. One can plot a frequency histogram of the edges and either visually pick a frequency as *MinTraffic* or use the parameters of the exponential equation to set *MinTraffic*.

3.4 Experiments

To show the effectiveness and efficiency of FlowScan, we test it against various datasets. FlowScan was implemented in C++ and all tests were performed on a Intel Core Duo 2 E6600 machine running Linux.

3.4.1 Data Generation

Due to the lack of real-world data, we used a network-based data generator provided by [10][3]. It uses a real-world city road network as the road network and generates moving objects on it. Objects are affected by the maximum speed on the road, the maximum capacity of the road, other objects on the road, routes, and other external factors.

The default generator provided generates essentially random traffic: an object's starting and end locations are randomly chosen within the network. In order to generate some interesting patterns, we modified how the generator chooses starting and end locations. Within a city network, "neighborhoods" are generated. Each neighborhood is generated by picking a random node and then expanding by a preset radius (3–5 edges). Moving objects are then restricted to start and end in neighborhoods.

Hot routes form naturally because of the moving object's preference for the quickest path. As a result, bigger roads (*e.g.*, highways) are more likely to be chosen by the moving objects. However, if too many objects take a highway or a road, it will reach capacity and actually slow down. In such cases, objects will choose to re-route and possibly create secondary hot routes.

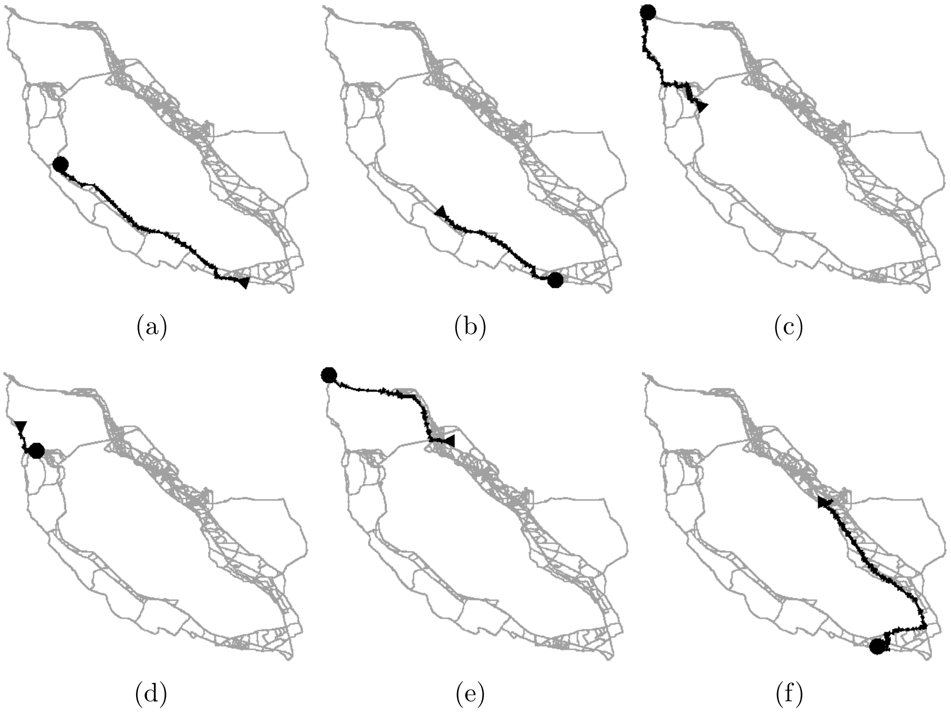
---

[3] http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/

3.4.2 Extraction Quality

General Results

To check the effectiveness of FlowScan, we test it against a variety of settings. First, we present the results for two general cases. Figure 3.10 shows several routes extracted from 10,000 objects moving in the San Francisco bay area. 10 neighborhoods of radius 3 each were placed randomly in the map. *Eps* and *MinTraffic* were set to 2 and 300, respectively. Each hot route is drawn in black with an arrow indicating the start and a dot indicating the end. The gray lines in the figures indicate all traffic observed in the input data (not the entire city map).


(a) (b) (c)

(d) (e) (f)

Figure 3.10: Hot routes in San Francisco data map

Even though the neighborhoods were completely random, we get realistic hot routes in this experiment. The hot routes in Figure 3.10(a) and 3.10(b) are CA-101 connecting San Francisco and San Jose, a major highway in the area. Figures 3.10(c) and 3.10(d) correspond to the Golden Gate Bridge connecting the city of San Francisco to the north. One of the random neighborhoods must have been across the bridge so objects had no choice but to use the bridge. Figure 3.10(e) shows a hot route connecting Oakland to that same neighborhood across the Richmond-San Rafael Bridge. Lastly, Figure 3.10(f) corresponds to a hot route connecting approximately Hayward to San Jose via I-880.

Next, Figures 3.11 shows three hot routes extracted from 5,000 objects moving in the San Joaquin network. Three neighborhoods were picked in this network, each with radius of 3. *Eps* and *MinTraffic* were set to 2 and 400, respectively. In Figures 3.11(a) and 3.11(b), the horizontal portions of the hot routes correspond to I-205. In Figure 3.11(b), the vertical portion corresponds to I-5. Both these roads are major interstate highways. The roads in Figure 3.11(c) are W. Linne Rd and Kason Rd. By looking at the city map, we observe that they make up the quickest route between the two neighborhoods.

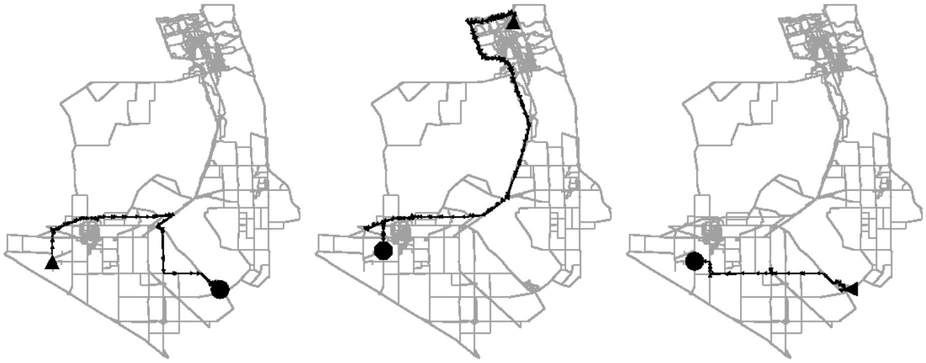

(a) (b) (c)

Figure 3.11: Hot routes in San Joaquin data map

Splitting Hot Route Behavior

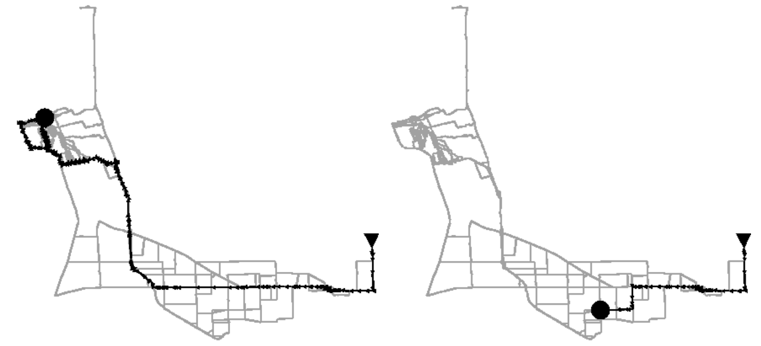
We also tested FlowScan with some specific traffic behaviors. First, we test the case of a hot route splitting into two. This data set is generated by setting the number of neighborhoods in a road network to three and fixing the start node to be in one of the three. Because start and destination neighborhoods cannot be the same, this forces the objects (1000 of them) to travel to one of two destinations. And because the objects like to travel on big roads (due to speed preference), they will usually leave the starting neighborhood using the same route regardless of the final destination and split sometime later.

Figures 3.12(a) and 3.12(b) shows the two hot routes extracted from the data. Both hot routes start at the green arrow at the lower right, move to the middle, and the split according to their final destinations. In Figure 3.12(c), the result from an AltGraph algorithm is shown. All edges that exceed the *MinTraffic* threshold (100) are connected if they are adjacent in the road network. Obviously, the two hot routes are connected together because the underlying objects are not considered. Figure 3.12(d) shows the result from a AltTrajectory algorithm [44]. In it, 14 clusters were found. Because shape is a major factor in trajectory clustering, the routes were broken into different clusters. The split is "detected" simply due to the hard left-turn shape, but the routes are not intact. One could post-process the results and merge near-by clusters, but this could run into the same problems as AltGraph since individual trajectories are ignored.

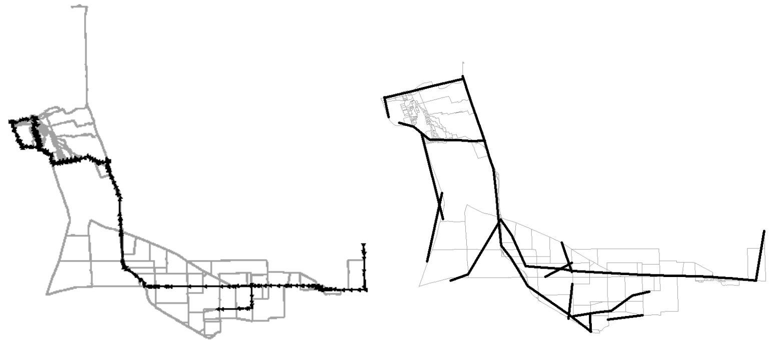
Overlapping Hot Route Behavior

Next, we test the case of two hot routes overlapping. That is, one starts at the same place as where the other one ends. To generate this data set, we also set the number of neighborhoods to three. Let them be known as A, B, and C. Then, for half of the objects, their paths are $A \rightarrow B$; and for the other half, their paths are $B \rightarrow C$. We set the radius of neighborhood B to 0 to ensure that the two hot routes overlap.

Figure 3.13 shows the results of this test. As the graphs show, two hot routes were extracted. Figure 3.13(c) shows a result with an AltGraph algorithm. Although $B \rightarrow C$ (not shown) is correctly extracted in that algorithm, $A \rightarrow B$ is not. It is incorrectly linked together with $B \rightarrow C$ and erroneously forms $A \rightarrow B \rightarrow C$. This is because individual identities are not considered in the algorithm. Figure 3.13(d) shows the result of AltTrajectory. 13 clusters



(a) FlowScan: $A \to B$ (b) FlowScan: $A \to C$



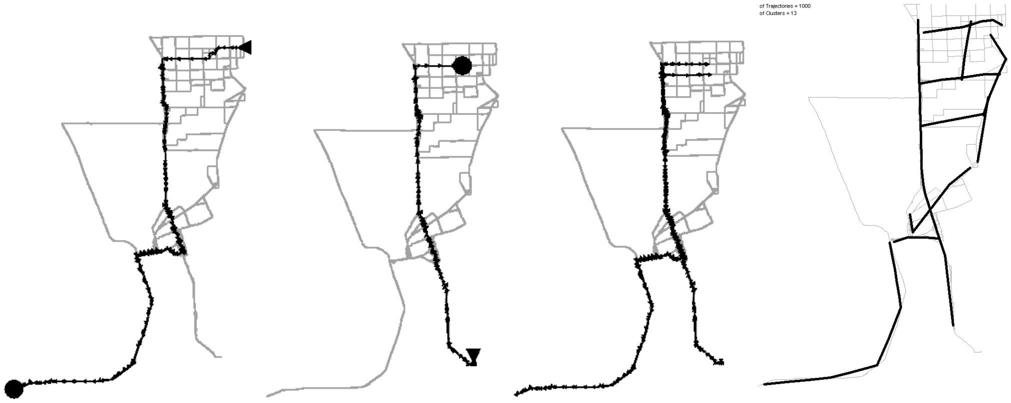
(c) AltGraph (d) AltTrajectory

Figure 3.12: Splitting hot routes

were discovered. Again, the routes are not intact. But more seriously, the trajectories near B are clustered into a single cluster because their shapes are similar.

3.4.3 Efficiency

Finally, we test the efficiency of FlowScan with respect to the number of objects. Figure 3.14 shows the running time as the number of objects increases from 2,000 to 10,000 with *MinTraffic* set to 10%. All objects were stored in memory, and time to read the input data is excluded. As the curve shows, running time increases linearly with respect to the number of objects. Next, we test the difference in disk I/O using a clustered Edge Table vs. an unclustered Edge Table. Figure 3.15 shows the result. Pages were set to 4K each and a buffer of 10 pages was used. We excluded the I/Os of the Vertex Index Tree and the TID



(a) $B \to C$ (b) $A \to B$ (c) AltGraph (d) AltTrajectory

Figure 3.13: Overlapping hot routes

List Table since they are the same in both cases. The figure shows the percent improvement of the clustered Edge Table. It is a significant improvement ranging from 588% to over 800%. This value is relatively stable because the percent improvement depends more on the structure of the network than the number of objects.

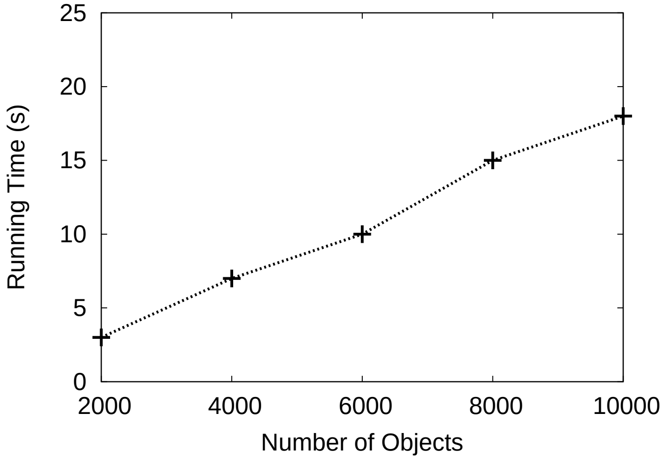


Figure 3.14: Efficiency with respect to number of objects

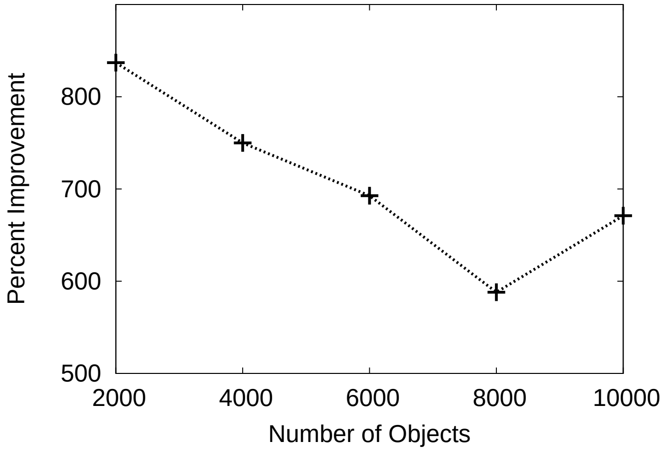


Figure 3.15: Disk I/O improvement of clustered index on Edge Table

Chapter 4

Sampling in Multidimensional Data

In many applications the complete source data is not available. Only a **sample** of the population is available for study. In moving object analysis, this is often the case due to either privacy protection or the size of the install base of the relevant technology. Nonetheless, multidimensional aggregates, *e.g.*, data cubes [23], are still needed for this data. This was the motivation for our recent study [52]. Consider the following example.

Example 4 (Sampling Data) Nielsen Television Ratings *in the United States are the primary source of measuring a TV show's popularity. Each week, a show is given a rating, where each point represents 1% of the US population. There are over 100 million televisions in the United States, and it is impossible to detect which shows they are watching every day. Instead, the Nielsen ratings rely on a statistical sample of roughly 5000 households across the country. Their televisions are wired and monitored constantly, and the results are extrapolated for the entire population.*

This example shows a typical use of sampling data. In many real world studies about large populations where the data collection is required, it is very difficult to gather the relevant information from everyone in the population. It would be too expensive and often simply impossible. Nonetheless, multidimensional analysis must be performed with whatever data is available.

Example 5 (OLAP on Sampling Data) *Advertisers are major users of TV ratings. Based on the rating, they can estimate the viewership of their advertisements and thus pay appro-*

priate prices. In order to maximize returns, advertisers want the maximum viewership of their target audience. For example, if the advertised product is a toy, the advertiser would want a TV show with children as its main audience. As a result, advertisers demand ratings be calculated in a multidimensional way. Popular dimensions include age, gender, marital status, income, etc.

To accommodate the multidimensional queries, attributes are recorded at the sampling level, *e.g.*, a recorded viewership for television show X might be attached to a "married male with two children." This leads directly to **OLAP on multidimensional sampling data**. Compared to traditional OLAP, there is a subtle and yet profound difference. In both cases, the intent or final conclusion of the analysis is on the population. But the input data are very different. Traditional OLAP has the complete population data while sampling OLAP only has a minuscule subset. Table 4.1 shows a summary of the differences.

Input Data	Analysis Target	Analysis Tool
Population	Population	Traditional OLAP
Sample	Population	*Not Available*

Table 4.1: Two models of OLAP application

The question is then "Are traditional OLAP tools sufficient for multidimensional analysis on sampling data?" The answer is "No" for several reasons. First is the *lack of data.* Sampling data is often "sparse" in the multidimensional sense. When the user drills down on the data, it is very easy to reach a point with very few or no samples even when the overall sample is large. Traditional OLAP simply uses whatever data is available to compute an answer. But to extrapolate such an answer for the population based on a small sample could be very dangerous. A single outlier or a slight bias in the sampling can distort the answer significantly. For this reason, the proper analysis tool should be able to make the necessary adjustments in order to prevent a gross error. Second, in studies with sampling data, statistical methods are used to provide a measure of reliability on an answer. Confidence intervals are usually computed to indicate the quality of answer as it pertains to the population. Traditional OLAP is not equipped with such tools. And lastly is the challenge of high dimensionality in the input data. While common to many other problems, sampling data analysis offers some new challenges such as the relationship between the quality of samples and an extremely large number of subspaces.

Example 6 (Usage Example) *To give a more concrete application of multidimensional sampling data, consider a retail outlet trying to find out more about its customers' annual income levels. In Table 6.1, a sample of the survey data collected is shown*[1]*. In the survey, customers are segmented by four attributes, namely* `Gender`*,* `Age`*,* `Education`*, and* `Occupation`*.*

Gender	Age	Education	Occupation	Income
Female	23	College	Teacher	$85,000
Female	40	College	Programmer	$50,000
Female	31	College	Programmer	$52,000
Female	50	Graduate	Teacher	$90,000
Female	62	Graduate	CEO	$500,000
Male	25	Highschool	Programmer	$50,000
Male	28	Highschool	CEO	$250,000
Male	40	College	Teacher	$80,000
Male	50	College	Programmer	$45,000
Male	57	Graduate	Programmer	$80,000

Table 4.2: Sample survey data

First, the manager tries to figure out the mean income level of programmers in their customer base (i.e., `Occupation` = Programmer*). The mean income for them is $55,400 with a 95% confidence interval of ± $12,265. This seems like a reasonable answer to the manager. Next, the manager queries for the mean income of teachers in the 30–40 age range. Although the data only contains one sample matching this criterion, an "expanded" answer of $85,000 ± $5,657 with 95% confidence is returned and the manager is satisfied.*

---

[1] For the sake of illustration, ignore the fact that the sample sizes are too small to be statistically meaningful.

To this end, this chapter proposes the **Sampling Cube** framework [52], which adds the following features to the traditional OLAP model: (1) Calculations of point estimates and confidence intervals for all queries are provided. Algebraic properties of the measures are exploited to make the calculations efficient. (2) When a query reaches a cell with too few samples, the query is "expanded" to gather more samples in order to improve the quality of the answer. The expansion takes advantage of the OLAP structure to look for semantically similar segments within the queried cuboid and also nearby cuboids. Two sample hypothesis testing is performed for expansion candidates, and the ones that pass are merged with query segment to produce the final answer. (3) Lastly, to handle the high dimensionality problem, the **Sampling Cube Shell** is proposed. Instead of materializing the full sampling cube, only a small portion of it is constructed. But unlike other cube compression schemes, the selection is based on the quality of sampling estimates. In tests with real world sampling data, the framework is shown to be efficient and effective at processing various kinds of queries.

The rest of the chapter is organized as follows. In Section 4.1, formal definitions of the problem are given. Section 4.2 describes the whole sampling cube framework of efficient aggregation and query expansion. Section 4.3 describes the sampling cube shell with optimizations for high dimensional data. Section 4.4 shows experimental results with respect to query efficiency and effectiveness.

4.1 Definitions

4.1.1 Data Cube Definitions

Before giving the proper problem definitions, we will review the data cube model. Given a relation R, a *data cube* (denoted as C_R) is the set of aggregates from all possible group-by's on R. In an n-dimensional data cube, a cell $c = (a_1, a_2, \ldots, a_n : m)$ (where m is the cube measure on some value) is called a k-dimensional group-by cell (*i.e.*, a cell in a k-dimensional cuboid) if and only if there are k ($k \leq n$) values among $(a_1, a_2, \dots, a_n)$ which are not $*$ (*i.e.*, all). Given two cells c_1 and c_2, let V_1 and V_2 represent the set of values among their respective $(a_1, a_2, \dots, a_n)$ which are not $*$. c_1 is the *ancestor* of c_2 and c_2 is a *descendant* of c_1 if $V_1 \subset V_2$. c_1 is the *parent* of c_2 and c_2 is a *child* of c_1 if $V_1 \subset V_2$ and $|V_1| = |V_2| - 1$. These relationships also extend to cuboids and form a structure called the *cuboid lattice*. An example is shown in Figure 4.1. The "All" or *apex* cuboid holds a single cell where all its values among $(a_1, a_2, \dots, a_n)$ are $*$. On the other extreme, the *base* cuboid at the bottom holds cells where none of its $(a_1, a_2, \dots, a_n)$ values is $*$.

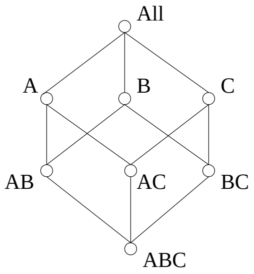


Figure 4.1: Cuboid lattice

Table 4.2 is an example of R. Each tuple corresponds to a person being sampled. The four attributes: `Gender`, `Age`, `Education`, and `Occupation`, segment the person. And finally, the value of the data is the `Income` of the person.

The query model follows the semantics of a data cube being constructed on the input relation R. More specifically, the user can pick any cell in C_R and ask for information about V for that cell. The measure of the cube (*e.g.*, average, sum) is computed on V and returned. In the motivating example, the first query of `Occupation`=*Programmer* is essentially a cell in the `Occupation` cuboid with all other dimensions set to $*$ or "don't care." The measure of the cube is the average of `Income`.

4.1.2 Problem Definition

As Example 6 showed, there are a few additional features than traditional OLAP. First, given α as a confidence level (*e.g.*, 95%), all cell queries to C_R should return a confidence interval of α confidence in addition to the measure as an answer. In comparison to traditional OLAP, the confidence level indicates the reliability of the measure to the user. Second, given *minsup* as a minimum support on the number of samples, if a cell's sample size does not satisfy *minsup* (*i.e.*, sample size $<$ *minsup*), "similar" cells in the data cube will be used to "boost" the confidence interval if certain criteria are satisfied. Lastly, if there is insufficient space to materialize the full data cube, a new methodology is needed to answer queries.

4.1.3 Confidence Interval

To make future discussions easier, a quick review of confidence interval calculation is given here [27].

Let x be a set of samples. The mean of the samples is denoted by $\bar{x}$, and the number of samples in x is denoted by l. Assuming that the standard deviation of the population is unknown, the *sample* standard deviation of x is denoted by s. Given some desired confidence level, the confidence interval for $\bar{x}$ is

$$\bar{x} \pm t_c \hat{\sigma}_{\bar{x}} \tag{4.1}$$

where t_c is the critical t-value associated with the confidence level and $\hat{\sigma}_{\bar{x}} = \frac{s}{\sqrt{l}}$ is the estimated standard error of the mean. To find the appropriate t_c, one has to know the desired confidence level (*e.g.*, 95%) and also the degree of freedom, which is just $l - 1$.

4.2 The Sampling Cube Framework

This section will describe the full framework of providing OLAP to sampling data. Specifically, it will describe how to store the sample data in a data cube structure named **Sampling Cube**, how to efficiently aggregate the data, and how to use similar neighboring cells to boost confidence. For now, only *full materialization* of this sampling cube is discussed. In other words, all cuboids and cells are constructed. The next section will discuss alternatives when full materialization is unrealizable.

4.2.1 Materializing the Sampling Cube

Given the base relation (base cuboid) in R, it is straightforward to construct C_R. There are many algorithms to efficiently do this [9, 82]. They all essentially provide an efficient way of traversing the cuboid lattice space in order to minimize the scanning of the data. In the sampling cube, the new question is *how to efficiently calculate the confidence interval at high-level cells.* The naïve way is to gather all the corresponding raw samples in the original input data and calculate the sample mean, sample standard deviation, and then the confidence interval. However, this is not very efficient since many cells contain a large number of samples. To repeat the computation for every cell in C_R is expensive and also redundant since the samples are shared between cells.

In traditional OLAP, this problem is solved by exploiting certain properties of the cube measure. There are two popular properties: **distributive** and **algebraic**. A measure is distributive if it can be computed solely based on the measures of its subsets, and a measure is algebraic if it can be computed based on a bounded number of measures of its subsets. Sum is an example of a distributive measure and mean is an example of an algebraic measure. These two properties are desirable because they facilitate very efficient aggregation. In this work, the measures of the cube are mean and confidence interval. Mean is known to be algebraic. The attention now turns to confidence interval. It is easy to see that it is definitely not distributive. But is it algebraic?

Lemma 3 *The confidence interval measure is algebraic.*

Proof 1 *There are three terms in the confidence interval computation. First is the mean of the sample set,* $\bar{x}$*. This was shown to be algebraic already. Second is the critical t-value, which is calculated by a lookup. With respect to* x*, it depends on* l *(count) and it is easy to see that count is distributive. The final term is* $\hat{\sigma}_{\bar{x}} = \frac{s}{\sqrt{l}}$*, which also turns out to be algebraic if one records the linear sum (*$\sum_{i=1}^{l} x_i$*) and squared sum (*$\sum_{i=1}^{l} x_i^2$*).*

To summarize, the mean and confidence interval measures of the data cube are algebraic. At every cell, exactly three values are sufficient to calculate them; all of which are either distributive or algebraic. They are the following:

1. l
2. $sum = \sum_{i=1}^{l} x_i$
3. $squared\ sum = \sum_{i=1}^{l} x_i^2$

With this knowledge in hand, constructing the sampling cube is very clear now. Any of the previously developed cubing algorithms can be used with just aggregating the three above values in each cell. At query time, the three values are then used to compute the mean and confidence interval.

4.2.2 Boosting Confidence for Small Samples

Now that the sampling cube is materialized, the next step is to use it. Recall that queries are point or range queries posed against the cube. Without loss of generality, consider the case of a point query, which corresponds to a cell in C_R. The goal is to provide an accurate point estimate (in the example, the sample mean of `Income`) for the samples in that cell. Because the cube also reports the confidence interval associated with the sample mean, there is some measure of "reliability" to the answer. If the confidence interval is small, the reliability is deemed good. But if the interval is large, the reliability is questionable.

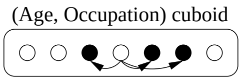
Consider what affects the size of the confidence interval. There are two main factors. The first is the variance of the sample data. A rather large variance in the cell may indicate that the chosen cube cell is not good for prediction and a better solution is probably to drill down on the query cell to a more specific one, *i.e.*, asking more specific queries. Second, a small sample size can cause a large confidence interval. When there are very few samples, the corresponding t_c is large because of the small degree of freedom. This in turn could cause a large confidence interval. Intuitively this makes sense. Suppose one is trying to figure out the average income of people in the United States. Just by asking 2 or 3 people does not give much confidence to the answer.

The best way to solve this small sample size problem is to simply get more data. This, however, is easier said than done. Gathering data is often the most expensive part of the analysis. So what can be done instead? Fortunately, there is an abundance of additional data available already. They do not match the query cell exactly, but they are conveniently organized in a structured data cube. Perhaps they can be used if certain criteria are met.

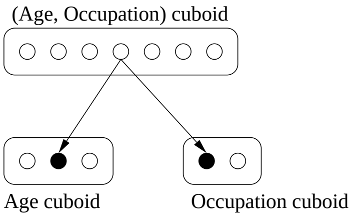
Figure 4.2 shows the two possible methods to "expand" the query and get more data to boost confidence. They both expand the original query in the data cube, just in different directions.

Intra-Cuboid Query Expansion

In the intra-cuboid case, the expansion occurs by looking at nearby cells in the same cuboid as the queried cell. But as mentioned before, careful consideration is needed before expansion. The new samples should only serve the purpose of increasing the confidence in the answer



(a) Intra-Cuboid Expansion



(b) Inter-Cuboid Expansion

Figure 4.2: Query expansion within sampling cube

and *not* change the semantic of the query. There are two primary questions to answer. First, "Which dimension(s) should be allowed to expand?" And second, "Which value(s) within those dimension(s) should the expansion use?"

To answer the first question, dimensions which are *uncorrelated* or *weakly correlated* with the measure value (*i.e.*, the value to be predicted) are the best candidates for expansion. Expanding within these dimensions will likely increase the sample size and not shift the answer of the query. Consider an example of a 2D query specifying `Education` = "College" and `Birth Month` = "July". Let the cube measure be `average Income`. Intuitively, education has a high correlation to income while birth month does not. As a result, it would be harmful to expand the `Education` dimension to include values such as "Graduate" or "High School." They are likely to alter the final result. However, expansion in the `Birth Month` dimension to include other month values could be helpful, because it is unlikely to change the result but will increase sampling size.

To mathematically measure the correlation of a dimension to the cube value, the correlation between the dimension's values and their aggregated cube measures is computed. *Pearson's correlation coefficient* for numerical data and χ^2 value for categorical data are popularly used correlation measures (although there are many other measures, such as *co-*

variance, can be used) [27]. A dimension that is strongly correlated with the value to be predicted should *not* be a candidate for expansion. Notice that since the correlation of a dimension with the cube measure is independent of a particular query, it should be precomputed and stored with the cube measure to facilitate efficient online analysis.

Now that the possible dimension(s) for expansion have been selected, the next step is to select the values within those dimensions. This relies on the semantic knowledge of the dimensions in question. The goal should be to select semantically similar values in order to minimize the risk of altering the final result. Consider the `Age` dimension, similarity of values in this dimension is clear. There is a clear order to the values. For dimensions with *numerical* or *ranked* data (*e.g.*, `Education`), such an ordering is clear and one should select values closer to the instantiated query value. For categorical data with its dimension organized in a multilevel hierarchy in a data cube (such as location), one should select those values located in the same branch of the tree (such as in the same district or city).

When such domain knowledge exists, semantically similar cells maybe used to boost the confidence interval of the query cell. Figure 4.2(a) shows an illustration. But when such knowledge does not exist, one has to be very careful in how other cells are used. A naïve approach could be to simply compare the query cell vs. all other cells in the dimension and use the most similar. But this could easily fall into the trap of "self-fulfilling prophecy." This is a term used in sociology where pre-existing beliefs about a false outcome evoke behavior that actually brings the outcome to fruition. In the case of intra-cuboid expansion with no domain knowledge, one has to be careful of this problem. Just blindly using cells that contain similar values may bring about an answer that does not semantically meaningful.

Even though strongly correlated dimensions are precluded from expansion, yet another precaution should be taken to ensure that expansion does not alter the answer of the query. In other words, the new samples should share the same cube value (*e.g.*, mean income) as the existing samples in the query cell. A method in statistics to determine whether two samples have the same mean (or any other point estimate) is the **Two Sample T-Test** [27]. This is a relatively simple and statistically sound test used in many applications. Due to space restrictions, we will skip its definitions. At a high level, the test will determine whether two samples have the same mean (the null hypothesis) with the only assumption being that they are both normally distributed. The test fails if there is evidence that the two samples do not share the same mean. Furthermore, the tests can be performed with a confidence level as an input. This allows the user to control how strict or loose the query expansion will be. As it turns out, the aforementioned three values recorded at each data cube cell are sufficient to perform the two sample t-test. This allows the test to be performed efficiently given any two cells in the data cube.

Example 7 (Intra-Cuboid Expansion) *Given the input relation in Table 4.2, let a query be "`Age` = 25" at 95% confidence. This returns an `Income` of $50,000 with a rather large confidence interval*[2]*. Since this confidence interval is larger than the preset threshold and the `Age` dimension was found to have little correlation with `Income` in this dataset, intra-cuboid expansion starts within the `Age` dimension. The nearest cell is "`Age` = 23," which returns an `Income` of $85,000. The two sample t-test at 95% confidence passes so the query expands; it is now "`Age` = {23, 25}" with a smaller confidence interval than initially. However, it is still larger than the threshold so expansion continues to the next nearest cell: "`Age` = 28", which returns an `Income` of $250,000. The two sample t-test between this cell and the original query cell fails; as a result, it is ignored. Next, "`Age` = 31" is checked and it passes the test. The confidence interval of the three cells combined is now below the threshold and the expansion finishes at "`Age` = {23, 25, 31}."*

In summary, dimensions not correlated with the cube measure are chosen as candidates for intra-cuboid expansion. Semantic similarity of the dimension's values are used to slowly

---

[2] For the sake of the example, suppose this is true even though there is only one sample. In practice, there should be a few more points to calculate a legitimate value.

expand a neighborhood of candidate cells around the query cell. For each candidate cell, the two sample t-test is performed to decide whether the candidate cell should be included in the expansion. When there is no semantic knowledge available, it might be unwise to expand unless the particular application calls for it.

Inter-Cuboid Query Expansion

The choices in inter-cuboid query expansion are slightly easier. Figure 4.2(b) shows an illustration. The expansion occurs by looking to a more general cell (drawn in black). In the figure, the cell in cuboid `Age, Occupation` can either use its parent in `Age` or `Occupation`. One can think of inter-cuboid as just an extreme case of intra-cuboid where *all* the cells within a dimension are used in the expansion. This essentially sets the dimension to $*$ and thus generalizes to a higher level cuboid.

Given a k-dimensional cell, there are k possible direct parents in the cuboid lattice. Though there are *many* more ancestor cells in the data cube if multiple dimensions are allowed to be rolled up simultaneously, only one is allowed here to make the search space tractable and also to limit the change in the semantics of the query. Using similar tests as the last section, correlated dimensions are not allowed in inter-cuboid expansions. Within the uncorrelated dimensions, the two sample t-tests can be performed to confirm that the parent and the query cell share the same sample mean. If multiple parent cells pass the test, the confidence level of the test can be adjusted progressively higher until only one passes. Alternatively, multiple parent cells can be used to boost the confidence simultaneously. The choice is application dependent.

Example 8 (Inter-Cuboid Expansion) *Given the input relation in Table 4.2, let the query be "`Occupation` = Teacher $\wedge$ `Gender` = Male." There was only one matching sample in Table 4.2 with `Income` = \$80,000. Suppose the corresponding confidence interval is larger than the preset threshold. There are two parent cells in the data cube: "`Gender` = Male" and*

"`Occupation` = Teacher." By moving up to "`Gender` = Male" (and thus setting `Occupation` to ∗), the mean `Income` is $101,000. A two sample t-test reveals that this parent's sample mean is not the same as the original query cell's. So it is ignored. Next, "`Occupation` = Teacher" is tried. It contains a mean `Income` of $85,000 and passes the two sample t-test. As a result, this new value is used and the query is expanded to "`Occupation` = Teacher."

Though the above method will work for inter-cuboid expansion, there is another method which makes more sense computationally. Instead of looking at the problem as expanding from the query cell *up* to a more general cell, one could look at it as the more general cell looking *down* at the query cells (*i.e.*, its children) and making its determinations about expansion. This method leads directly into the next section about the sampling cube shell.

Expansion Method Selection

Before moving on, some discussion is needed about intra-cuboid expansion vs. inter-cuboid expansion. This is a difficult question to answer a priori without knowing the data and the application. The first guideline in choosing between the two should be what is the tolerance for change in the semantics of the query. This depends on the specific dimensions chosen in the query. For instance, the user might tolerate a bigger change in semantics for the `Age` dimension than `Education`. The difference in tolerance could be so large that he/she is willing to set `Age` to $*$ (*i.e.*, inter-cuboid expansion) than letting `Education` change at all.

If no domain knowledge is available, the main quantitative guides are the correlation coefficients and the two sample t-test between the query cell and the possible expansion cells. The value of the correlation coefficient is an indication of expansion's safety. And by progressively setting higher confidence levels in the test, one could choose between the two expansion methods by seeing which one passes the higher confidence level test. This offers a numerical way of comparing between the two choices, but in a real world application, domain knowledge is definitely a better method of making the ultimate choice.

4.3 The Sampling Cube Shell

So far, the discussion has only focused on the full materialization of the sampling cube. In many real world problems, this is often impossible. Even given a modest number of dimensions in the base data, constructing the whole data cube can be prohibitive. Recall that the number of cuboids in a data cube is exponential with respect to the number of dimensions. So even with just 20 dimensions, 2^{20} cuboids can be quite a pain to handle. The real world survey data used in this work's experiments contains over 600 dimensions!

Clearly, another methodology is needed. Ideally, this new methodology should be able to provide the same or close to the same answers as the full sampling cube with a much smaller computation and storage requirement.

To motivate the proposal, first imagine what a typical query will be. An analyst will select some specific values in some dimensions and ask for the confidence interval in that cell. But most likely, the number of dimensions specified in the query will be low (≤ 5: *e.g.*, `Age`, `Gender`, `Marital Status`, `Income Level`, etc.). To specify a high dimensional cell is to target a very specific segment of the population. One that is probably too specific to be of any value. This means that high dimensional cuboids are probably not needed.

Second, consider what would happen if `Birth Month` were a dimension in Table 4.2. Clearly, there should not be any correlation between the month of a person's birth date and his or her income level. This can be statistically verified by checking the standard deviation or the confidence level (both of which should be large) of the cells in the `Birth Month` cuboid. Now recall the ultimate goal of the user. It is to extract meaningful information about the cube value (*e.g.*, `Income`). If the sample standard deviation of the value is high for many cells in a cuboid, it indicates that there is little information to be found in this cuboid. Therefore, there is probably little utility in presenting the cuboid to the user. Furthermore, additional higher level cuboids that combine with `Birth Month` can probably be skipped, too. This drastically cuts down on the size of the sampling cube since it essentially removes one dimension.

This motivation leads directly to the proposal of the **Sampling Cube Shell**. As the name suggests, it is a "shell" around the complete sampling cube that only computes some of the outer layers. Figure 4.3 shows a sample illustration. In it, only a portion of the cuboids are materialized (the shaded ones). They are the shell around the data cube and will be used to answer the queries.

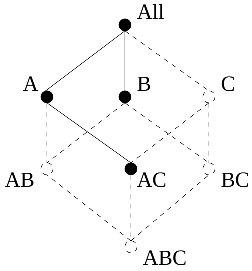


Figure 4.3: Sampling Cube Shell

4.3.1 Building the Sampling Cube Shell

The algorithm to build the sampling cube shell is top-down. It starts at the apex cuboid and proceeds down the cuboid lattice towards the base cuboid. The search in this space is iterative and greedy: in each iteration, the best candidate cuboid is chosen and added to the shell. This process halts until some stopping condition is met. The condition could be a space constraint, *i.e.*, number of cuboids built cannot exceed some value. Or it could be an information constraint, *i.e.*, the gain in building a new cuboid must exceed some minimum.

Cuboid Standard Deviation

The first question is how to compare cuboids (in order to get to the greedy selection). This requires a measure on the "goodness" of a cuboid.

Definition 8 (Cuboid Standard Deviation) *Given a cuboid B with m cells $\{c_1, \ldots, c_m\}$, the* Cuboid Standard Deviation *(CSD) is defined as*

$$CSD(B) = \frac{\sum_{i=1}^{m} s(c_i) \times \frac{n(c_i)}{n(B)}}{1 - \frac{\sum_{i=1}^{m} small(c_i)}{m}}$$

where $s(c_i)$ is the sample standard deviation of the samples in c_i, $n(c_i)$ is the number of samples in c_i, $n(B)$ is the total number of samples in B, and $small(c_i)$ is a function which returns 1 if $n(c_i) \geq minsup$ and 0 otherwise. If the denominator is 0, CSD is defined to be ∞.

The CSD of a cuboid measures the amount of variance with respect to the cube value in its cells. Obviously, the lower the value of CSD, the better the cuboid is at capturing the correlation between its cells and the value. The definition of CSD achieves this using a linear combination of its cells' standard deviations. In the final summation, the standard deviations are weighted by the sizes of the cells. As a result, if only a small percentage of the cells have low standard deviation but they hold the majority of the sampled data, they will have a large effect on the CSD of the cuboid.

For practical reasons, two minor adjustments are made to the CSD calculation. First, if $s(c_i)$ is very small ($< minsd$), it is set to 0. In other words, if the standard deviation of the cube values in a cell is already quite small, it is unnecessary to further examine that cell's subsets since they would contain the same behavior. Setting the $s(c_i)$ to 0 reflects this notion. Second, if $n(c_i) < minsup$, c_i is ignored. In such cases with so few samples in the cell, it is meaningless to measure any information from them. But, a situation could arise where a large portion of the cuboid's cells have small $n(c_i)$. For example, consider a dimension storing unique IDs of samples. In this case, the standard deviation of all cells would be 0 since each cell contains exactly one sample. The CSD is low at 0, but the cuboid is actually useless since its cells do not offer any generalization power. To penalize this type of situation, the denominator in the CSD calculation is set to reweigh the standard deviation calculations in the numerator by the percentage of so called “small” cells in the cuboid. Both these adjustments will come in handy later on during the construction of the cube shell.

Cuboid Standard Deviation Reduction

Given CSD as a measure of a cuboid’s cells’ correlation with the cube value, it is now possible to compare different cuboids quantitatively. However, in order to use it in the construction algorithm, another definition is needed to measure the *incremental* gain of a cuboid.

Definition 9 (Cuboid Standard Deviation Reduction) *Given a cuboid B and $parents(B)$ as the set of B's parents in the cuboid lattice, the* Cuboid Standard Deviation Reduction *(CSDR) is defined as*

$$CSDR(B) = \left[\min_{B' \in \, parents(B)} CSD(B') \right] - CSD(B)$$

The CSDR of a cuboid measures the *reduction* in CSD from one of its parents. Because the data cube is a lattice and not a tree, a cuboid can have multiple parents. To maximize the gain, the reduction is measured from the best parent.

Cube Shell Construction

Building the sampling cube shell is a top-down and greedy process. It uses CSDR to select the best cuboid in each step of growing the cube shell. Initially, only the *All* or apex cuboid exists. By definition, it contains exactly one cell and the standard deviation of it is the standard deviation of all the samples put together. The child cuboids of the apex cuboid are then added into a *candidate set*. The CSDR of each cuboid is computed. The candidate cuboid with the best CSDR is chosen and added to the shell. Its children in the cuboid lattice are added to the candidate set. This process iterates until a stopping criterion is met.

Note that the final data structure is not strictly a tree since a node could have multiple parents. It is just a portion (*i.e.*, shell) of the complete data cube lattice.

Two pruning methods are used in order to improve both the efficiency and the effectiveness of the resultant shell. They are directly related to the two adjustments made to the CSD calculation earlier.

First, if a cell's standard deviation is very low ($< minsd$), its descendant cells are removed from future consideration. The reason for this is that if the samples in a cell already share basically the same value for the point estimate, it is pointless to examine its sub-segments since most likely they will just produce the same values. This, in effect, achieves inter-cuboid query expansion. At runtime, if the query cell is one of the pruned descendant cells, it will not be found but the parent cell will be. The point estimate from the parent cell will then be used in substitute, but it will be fairly accurate since the standard deviation in its samples is so low. In essence, the query cell has been expanded to the parent cell a priori. Detailed discussion of query processing will be given in the next section.

Second, for every cell, if its sample size does not satisfy a minimum support threshold ($< minsup$), its descendant cells in the descendant cuboids are removed from future consideration. Intuitively, this supports the idea that if a segment is already very small, it is fruitless in analyzing its sub-segments, which could only get smaller. This is essentially the idea of the iceberg cube [9].

Algorithm 4.3.1 shows the shell construction algorithm in pseudo-code.

There are two possible ways the algorithm could halt, and the choice between them depends on the application. The first is the size of the shell. This is a natural halting criterion if storage space is the primary concern. As the next section and also later experiments will show, the bigger the shell, the higher the quality of query processing. The second criterion could be a minimum CSDR. At each iteration, if the largest CSDR in the candidate set does not exceed some minimum, the algorithm halts.

Example 9 (Shell Construction) *To make the algorithm more concrete, consider how it will work on the example data in Table 4.2. Initially, the candidate set only includes the apex cuboid of R. The CSD of this cuboid is simply the standard deviation of all rows in R, which is 143,760. Next, since it is the only cuboid in the candidate set, it is added to the shell. Its four descendants, which are the one-dimensional cuboids,* `Gender`*,* `Age`*,* `Education`*, and* `Occupation` *are added to the candidate set. Table 4.4(a) shows the CSD and CSDR of each candidate cuboid. For the sake of this example, values in the* `Age` *cuboid are binned into 4 ranges: 21–30, 31–40, 41–50, and 51+. This produces a more reasonable CSD value for the* `Age` *cuboid. Since these cuboids all share the same parent, their CSDR are all calculated with respect to the apex. This ends the first iteration of the algorithm.*

In the second iteration, the best candidate cuboid according to its CSDR value in Table 4.4(a), the `Occupation` *cuboid, is added to the shell. Its descendants, which are all the 2D cuboids that extend from* `Occupation`*, are added to the candidate set. Their CSDR values are calculated with respect to* `Occupation`*. Figure 4.4(b) shows the structure of the shell after the addition, and Table 4.4(c) shows the new candidate set with CSD and CSDR values.*

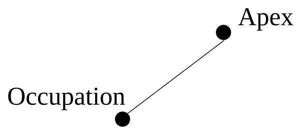
In the next iteration, the cuboid with the best CSDR, `Age`*, is added to the shell. Figure 4.4(d) shows the result of this addition. Its descendants are added to the candidate set and the algorithm continues.*

4.3.2 Query Processing

The final step is to process queries using the sampling cube shell. Recall that the query model consists of point queries in the data cube of the input relation. Because the sampling cube shell is not the complete data cube and does not contain all possible cuboids, there are three possibilities at query time.

Candidate Cuboid	CSD	CSDR
`Gender`	139,196	4,564
`Age`	102,836	40,924
`Education`	109,485	34,275
`Occupation`	43,852	**99,908**

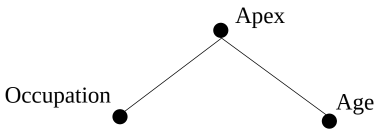
(a) Candidate set after first iteration



(b) Cube shell after first iteration

Candidate Cuboid	CSD	CSDR
`Gender`	139,196	4,564
`Age`	102,836	**40,924**
`Education`	109,485	34,275
`(Gender, Occupation)`	13,338	30,514
`(Age, Occupation)`	45,287	-1,435
`(Education, Occupation)`	6,261	37,591

(c) Candidate set after second iteration



(d) Cube shell after second iteration

Figure 4.4: Sampling cube shell construction example

Exact Dimension Match

First, the queried dimensions match one of the cuboids in the shell. In this case, the answer to the query exists in the shell already and it is simply returned.

Subset Dimension Match

Second, the queried dimensions are a subset of one of the cuboids in the shell. For instance, the queried dimension could be `Gender` and only the `(Age, Gender)` cuboid (and not the `Gender` cuboid) exists in the shell. This is entirely possible due to the build order of the construction algorithm. In this case, the exact answer is produced by scanning the superset cuboid in the shell for the appropriate rows and computing the necessary values on the fly.

Superset Dimension Match

So far, the answers produced have been lossless with respect to the full sampling cube. They either exist in the shell or can be computed in the shell from more specific cells. The last case to handle is when the queried dimensions are a *superset* of all cuboids in the shell. Though technically the query model allows any cell in the data cube to be queried, most likely it will be in the low dimensional ones ($\leq$5 dimensions). This was one of the motivations of the sampling cube shell in the first place. In other words, if this case occurs, most likely the sampling cube shell will contain a cuboid that is not very far off from the queried dimensions. For example, the queried dimensions could be `(Age, Gender, Occupation)`, and the largest cuboid in the shell only contains two dimensions.

In this case, a careful assessment is needed. In general, the queried dimensions could have a superset relationship to *multiple* cuboids in the shell. In the example above, both `(Age, Gender)` and `(Age, Occupation)` could exist in the shell and be used to answer the query. This raises two questions. First, which cuboid in the shell should be used? And second, how will the cuboid be used to answer the query?

In general, let there be k cuboids, $B_1 \ldots B_k$, in the sampling cube shell whose dimensions are subsets of the queried dimensions. Other than scanning base table, these k cuboids are the only sources of information about the query cell in the shell. But since they are all more general than the query cell, the final answer will have to be approximated from them.

The question is then which of the k cuboids should be used. The first goal should be to pick the cuboid that is closest to the query cuboid in the cuboid lattice. Semantically, this is the closest source of information. In general, multiple cuboids could tie for being the closest. Let there be k_0 of these where $k_0 \leq k$. Within these k_0 cuboids, the *average* of the point estimates at the relevant cells is then the point estimate answer for the query. In testing, several methods were tried, including weighted average by sampling size, weighted average by confidence interval size, choosing the cuboid with the highest confidence, and choosing the cuboid with the smallest sampling size. The simple average turns out to be the best due to the fact that it is not affected by any biases in sampling which could place an uneven number of samples in different cuboid cells. In testing, this was also confirmed to be the best on average.

Figure 4.5 shows a sample query in the `(Age, Occupation)` cuboid. Suppose that cuboid does not exist in the cube shell, but `Age`, `Occupation`, and the apex do. As a result, there are 3 ancestor cells in the cube shell. The cells in `Age` and `Occupation` are 1 hop away from the query cell in the cuboid lattice and the apex cell is 2 hops away. `Age` and `Occupation` tie for being closest to the query cell; the apex is ignored. The average of the `Age` and `Occupation` cells is then the answer to the `(Age, Occupation)` query.

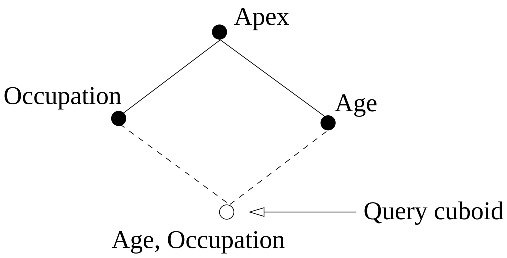


Figure 4.5: `(Age, Occupation)` query

4.4 Performance Evaluations

This section shows various evaluations of the sampling cube shell with real world data. Everything is implemented in C++ and compiled with GCC. Experiments were performed on a Linux machine with an Intel Pentium4 2.4GHz CPU and 2GB of memory.

Real sampling data from a Fortune 100 company was obtained for the tests. For confidentiality reasons, the name of the company, the names of products, or actual values cannot be revealed. The data contains over 750,000 samples and nearly 600 dimensions. Each sample is a record of a sale of a particular product to a single customer. The customer is then surveyed on various dimensions such as age, marital status, employment status, education level, etc.

Two subsets are extracted from the full data. One is a 21-dimensional dataset with the number of children (under age 16) as the cube value. The other is a 22-dimensional dataset with the household income as the cube value. In the original input data, income was already binned into 12 ranges. In testing, a random income value within the range is generated. As a result, some of the subtle patterns might have been lost but the big correlations should still hold.

In all tests, *shell_size* indicates the size of the sampling cube shell (*i.e.*, number of cuboids). It is the halting criterion for shell construction. Unless mentioned differently, *minsup* and *minsd* are both set to 0.

4.4.1 Shell Construction Efficiency

As mentioned previously, materializing the full sampling cube is often unrealistic in real world scenarios. It is known to be exponential in the number of dimensions. But what about the sampling cube shell? Figure 4.6 shows the time to compute cube shells of *shell_size* 20 and 50 as dimensionality increases from 5 to 20. For comparison, the full data cube (as computed by BUC [9]) is also shown. As expected, BUC explodes when the number of dimensions reaches higher than 15. In comparison, the cube shells grow linearly. This is not surprising because the time is constrained by *shell_size*. The reason the time does increase; however, is because the number of candidates to be examined increases as the number of dimensions does. For example, with the number of dimensions at 19 and *shell_size* at 50, the total number of candidates generated is 778. This is far smaller than the full cube (2^{15}) but it is also much larger than *shell_size*.

Next, efficiency with respect to the number of tuples in the input relation is checked. In traditional OLAP, this usually has a linear relationship to running time due to a linear increase in the number of overall cells. Figure 4.7 shows that this is also the case with the sampling cube shell. As the number of tuples increases from 20,000 to 100,000 in a 22 dimensional data set, the time to compute a cube shell (regardless of *shell_size*) is linear to

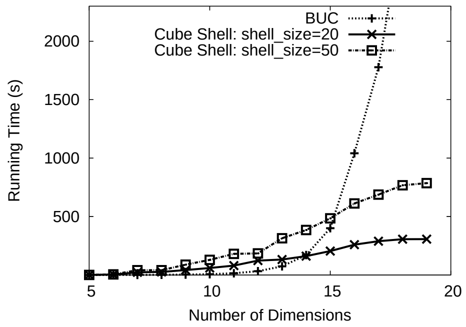


Figure 4.6: Materialization time vs. dimensionality

the size of the input.

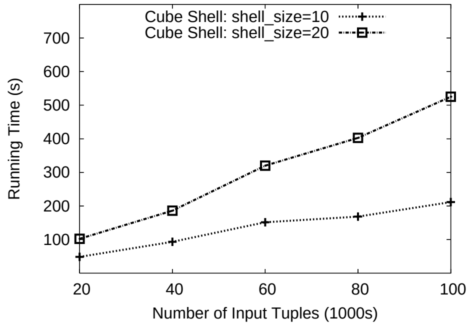


Figure 4.7: Materialization time vs. number of tuples

4.4.2 Query Effectiveness

Experiments in this section will address the two major claims on the effectiveness of the proposed framework. First, they will show that query expansion increases the reliability of the query answer. Second, they will show that query processing with the sampling cube shell produces negligible errors while reducing the space requirement significantly.

Query Expansion

To test the effectiveness of query expansion, the full input data of 750,000 tuples is taken to be the population and a random sample is taken from it. A full sampling cube is constructed on the sample, and query results are compared to the population in order to measure the accuracy.

In the first experiment, a sampling cube measuring the `average Number of Children` is built from a random 0.1% sample of the full input. Three dimensional point queries consisting of the `Gender`, `Marital`, and `Age` dimensions are given to the (1) full data, (2) sampling cube *without* query expansion, and (3) sampling cube *with* query expansion. The age dimension is allowed to expand to at most ± 2 years of the query age. The output below shows a sample query for `Gender` = *Female*, `Marital` = *Married* and `Age` = *36*.

```
> q GENDER FEMALE MARITAL MARRIED AGE 36

Population:            1.66 from 4783 points

Sample w/o expansion: 2.33 +/- 1.12 from 6 samples

Sample w/ expansion:  1.51 +/- 0.34 from 47 samples



  Difference in mean w/o expansion: 0.67

  Difference in mean w/ expansion:  0.15
```

As the output shows, 1.66 is the "correct" answer from (1). 2.33 ± 1.12 is the answer from (2) and 1.51 ± 0.34 is the answer from (3). Here, query expansion results in a significant improvement in the query answer. 1.51 is much closer to 1.66 than 2.33. The sample size also increases from 6 to 47, which reduces the 95% confidence interval.

Table 4.3(a) shows the full effect of intra-cuboid expansion in the `Age` dimension over many queries, similar to the above example. The first two columns of each line show the query values of the `Gender` and `Marital` dimensions. In each line, the `Age` dimension is enumerated over all possible values (approximately 80 distinct values), and each combination forms a distinct 3 dimensional point query in the sampling cube. The third and fourth columns show the average absolute error in the query results of the sampling cube without and with query expansion. As the fourth and fifth columns show, turning on intra-cuboid query expansion in the `Age` dimension improves the accuracy significantly. The last row in the table shows an average of 26% improvement from nearly 500 different queries. The last three columns in the table show the effect of query expansion on sampling size. Without expansion, the number of samples per query is only 1.4. With expansion, it increases to 13.4.

Table 4.3(b) shows a similar experiment with the `Age` dimension and the `average Household Income` as the cube measure. In this experiment, 0.05% of the input data is loaded into the sampling cube, and the age dimension is again allowed to expand ± 2 years from the query age. The two dimensions queried in addition to `Age` are `Gender` and `Education`. The result of this experiment is similar to the last one. In the nearly 650 queries executed, the average error reduction from no expansion to intra-cuboid expansion is 51%. Average sampling size also increases significantly.

Lastly, Table 4.3(c) shows another experiment. In this one, the `average Household Income` is still the cube measure. But the expansion is now within the `Number of Children` dimension, which is allowed to expand to ± 1 child. Three other dimensions were specified in the query, namely `Gender`, `Marital`, and `Education`. Due to limited space, only the final average is shown in Table 4.3(c). Again, both query accuracy and sampling size are improved. The experiment also shows the average reduction in the size of the confidence interval as a result of query expansion. With more sampling points, it is easy to see why the interval size would decrease. This, in addition to the reduction in the mean's error, improves the overall quality of the query results.

Sampling Cube Shell

Next, the query accuracy of the sampling cube shell is measured. The same query is given to the sampling cube shell and the full sampling cube, and the difference between the answers is the "error." The full sampling cube is simulated by scanning the input relation repeatedly since it was not possible to full materialize it.

First, a sanity check is performed by looking at the cuboids chosen by the cube shell according to CSD and CSDR. In the dataset measuring the number of children, the `Age`, `Age Cohort`, and `Generation` cuboids are chosen as the first three. Intuitively, these checkout to be sensible choices.

Figure 4.8 show the effect of *shell_size* on query accuracy using the dataset of average household income. 1000 random queries ranging anywhere from 1D to 5D were processed both by the sampling cube and the sampling cube shell. The absolute percent error is shown, which is defined as (|query answer from the shell cube − query answer from full cube|) ÷ query answer from the full cube.

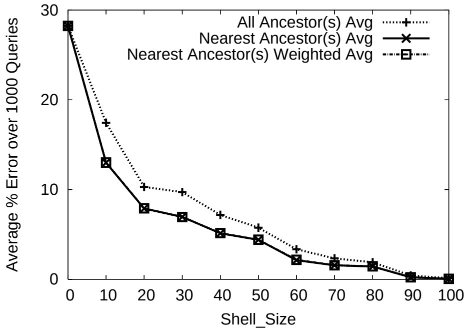


Figure 4.8: Query accuracy vs. *shell_size* for average household income dataset

In Figure 4.8, there are three curves. They show different methods of calculating the superset queries mentioned in Section 4.3.2. In "All Ancestor(s) Avg," the answer at the query cell is computed by taking the average of *all* ancestors in the cuboid lattice. In "Nearest Ancestor(s) Avg," the answer is computed by taking the average of the nearest ancestors in the cuboid lattice. And lastly, in "Weighted Nearest Ancestor(s) Avg," the answer is computed by taking the average of the nearest ancestors inversely weighted by the size of the sampling set. As all three curves show, as *shell_size* increases, the error decreases. This is expected because there are more cuboids in the shell to accurately answer queries. Amongst the three curves, "All Ancestor(s)" gives the largest amount of error while "Nearest Ancestor(s) Avg" gives the least. "Weighted Nearest Ancestor(s) Avg" is close but just slightly worse. The reason is that most of the time, there was only one nearest ancestor; thus the method of averaging only had a small effect.

Clearly, there is a tradeoff between storage space requirement and query processing quality. But considering that the full sampling cube size contains 2^{22} cuboids, a sampling cube shell with *shell_size* set to 100 uses less than 0.01% of the space required while producing less than 1% error. Even if one does not compute the full sampling cube and materializes only 5D or smaller cuboids, a sampling cube shell with *shell_size* set to 100 still uses less than 1% (100/35,442) of the space required.

Figure 4.9 shows an experiment testing the effect of a query's dimensionality on the error of the cube shell. The average household income dataset is used with *shell_size* fixed at 50. As expected, as the query drills further down into the data cube, the error increases. This is because the query cells are further away from the boundaries of the cube shell. As a result, the error in the parent cells' approximation of the answer increases. But most OLAP queries will not drill down very far. In most real world situations, the analyst will only specify 1D to 5D queries. In these cases, the average error is very low (< 5%) while using less than 0.01% of the space required by the full sampling cube.

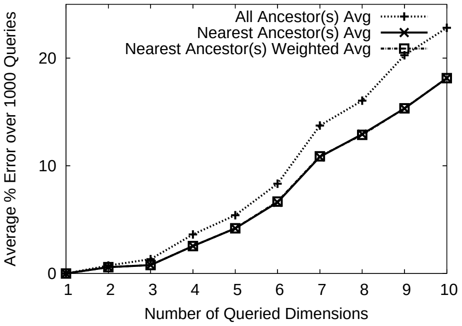


Figure 4.9: Query accuracy vs. query dimensionality for average household income dataset

Input: (1) Input table R; (2) *minsup*; (3) *minsd*

Output: Sampling cube shell S

Method:

```
$Candidates = \{$apex cuboid of $R\}$
while $Candidates \neq \emptyset$ or halting criteria not met
    $B$ = cuboid in $Candidates$ with largest CSDR
    remove $B$ from $Candidates$
    add $B$ to $S$
    add $B$'s descendant cuboids to $Candidates$
    update CSD values in $Candidates$
return $S$
```

Table 4.3: Query expansion experimental results

(a) Intra-Cuboid Expansion with `Age` dimension and `Average Number of Children` cube measure

Query		Average Query Answer Error			Sampling Sizes		
Gender	Marital	No Expand	Expand	% Imp.	Population	Sample	Exp.
FEMALE	MARRIED	0.48	0.32	33%	2473.0	2.2	28.3
FEMALE	SINGLE	0.31	0.21	30%	612.6	0.6	6.4
FEMALE	DIVORCED	0.49	0.43	11%	321.1	0.3	3.4
MALE	MARRIED	0.42	0.21	49%	4296.8	4.4	37.6
MALE	SINGLE	0.26	0.21	16%	571.8	0.5	3.6
MALE	DIVORCED	0.33	0.27	19%	224.7	0.2	1.2
Average		0.38	0.27	26%	1416.7	1.4	13.4

(b) Intra-Cuboid Expansion with `Age` dimension and `Average Household Income` cube measure

Query		Average Query Answer Error			Sampling Sizes		
Gender	Education	No Expand	Expand	% Imp.	Pop.	Sample	Exp.
FEMALE	HIGH SCHOOL	$55622	$31580	43%	641.4	0.3	3.9
FEMALE	SOME COLLEGE	$61526	$28822	53%	980.0	0.5	4.5
FEMALE	COLLEGE	$73309	$14504	80%	1132.8	0.5	6.8
FEMALE	POSTGRADUATE	$88658	$57907	34%	689.6	0.3	2.2
MALE	HIGH SCHOOL	$55671	$23503	57%	857.0	0.4	2.6
MALE	SOME COLLEGE	$63821	$34944	45%	1219.0	0.6	4.4
MALE	COLLEGE	$71120	$28913	59%	1511.0	0.9	7.4
MALE	POSTGRADUATE	$103619	$61191	40%	1191.8	0.6	4.2
Average		$71668	$35170	51%	1027.8	0.5	4.5

(c) Intra-Cuboid Expansion with `Number of Children` dimension and `Average Household Income` cube measure

	Average Query Answer Error				Sampling Sizes		
	No Expand	Expand	% Improve	CI Reduce	Population	Sample	Expanded
Average	$60873	$32458	51%	18%	713.9	3.6	12.3

Chapter 5

Moving Object Outliers

The previous chapter focused on abnormal behavior in the traffic of a road network. This chapter will focus on abnormal behavior in the individual moving objects. This was the basis for some of our recent work [49, 50]. Though outlier detection has been studied in many contexts [5, 38], the moving objects domain poses unique challenges. Some have tried to tackle this problem [54, 42]. However, they focus almost exclusively on the trajectories. In practice, trajectories are associated with non-spatiotemporal features and such associations are often more valuable for analysis. Imagine a group of objects having similar trajectories on record, but one of them does it in the middle of the night while the others occur during the day. The `time-of-the-day` feature would be helpful here for analysis. Additional information such as `speed` and `object type` play equally important roles. Further, anomalies may occur at arbitrary levels of abstraction and be associated with different time and location granularities. A speedboat may require analysis at the minute level while a freighter may only require the hour level. Hence, a systematic analysis of outliers must take into consideration of multiple features associated with different dimensions in each movement.

In addition, the treatment of trajectories have to be adjusted for the purpose of anomaly detection. Prior work in the area of trajectory prediction [54, 42] use Markov models or other sequential models to model a single trajectory and predict its future behavior. However, when used in the context of a large population with many different distributions, such approaches may not be effective.

There are in general two mechanisms for anomaly detection: *classification*, which relies on labeled training data sets, and *clustering*, which performs automated grouping without the aid of training data. Although both are interesting methods for mining moving object outliers, classification often leads to stronger mining results with the help of training data. Therefore, our focus will be on constructing a classification model.

The problem of anomaly detection in moving object data is defined as follows. The input data is a set of **labeled trajectories**: $\mathcal{D} = \{(t_1, c_1), (t_2, c_2), \ldots\}$, where t_i is a trajectory and c_i is the associated class label. A *trajectory*[1] is a sequence of spatiotemporal records of a moving object, *e.g.*, GPS records. Each record has the geographic location as well as a timestamp, and records can be made at arbitrary time intervals. The set of possible class labels is $\mathcal{C} = \{c^1, c^2, \ldots\}$. In simple anomaly detection, there could just be two classes: c^{normal} and $c^{abnormal}$.

The goal of the problem is to learn a function f which maps trajectories to class labels: $f(t) \rightarrow c \in \mathcal{C}$. f should be consistent with $\mathcal{D}$ as well as future trajectories not in $\mathcal{D}$. In other words, we want to learn a model which can classify trajectories as being normal or abnormal.

In [50], we propose a framework, called ROAM (<u>R</u>ule- and M<u>o</u>tif-based <u>A</u>nomaly Detection in <u>M</u>oving Objects), for the problem of anomaly detection. Compared to related work in classification or clustering of moving objects, ROAM incorporates a fuller feature space and examines more than just trajectories. At a high level, ROAM presents three novel features.

1. **Motif-based feature space**: Instead of modeling whole trajectories, we partition them into fragments (*motifs*) and construct a multi-dimensional feature space oriented on the motifs with associated attributes.

2. **Automated hierarchy extraction**: By examining the patterns in the trajectories, we automatically derive hierarchies in the feature space. This yields a multi-resolution view

---

[1] Trajectory in this paper is just data and does not imply path prediction.

of the data.

3. **Hierarchical rule-based classifier**: We develop a rule-based classifier which explores the hierarchical feature space and finds the effective regions for classification.

5.1 Key Insights

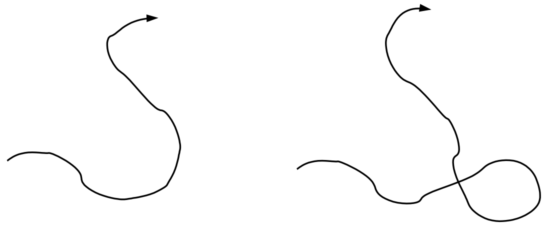
There have been some prior work in the area of trajectory prediction [54, 42]. Markov models or other sequential models can model a single trajectory and predict its future behavior. However, when used in the context of a large population with many different distributions, such approaches may not be effective.

Example 10 *Consider the two trajectories in Fig. 5.1(a). They have similar shapes except the one on the right has an extra loop. The impact of this additional loop depends on the task, but one would remark that the other portions are remarkably similar.*

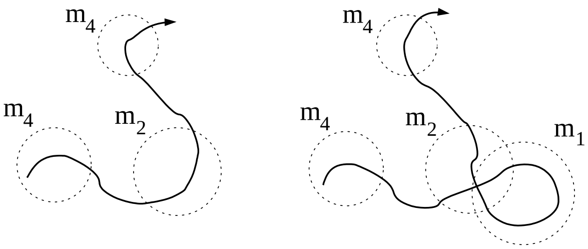
This example presents some problems for holistic models. It is difficult to represent the semantics of "mostly the same with the exception of an extra loop" using distance metrics between models. Local differences could either dominate the metric or be drowned out by the rest of trajectory. Furthermore, it is difficult to capture thousands or tens of thousands of trajectories in a single model. While a single object or a small set may have clear patterns, a large population (such as in real-world anomaly detection) presents a wide range of patterns across all granularities of time and space signals.

5.1.1 Motif-based Feature Space

In this paper, we propose that semantic analysis should be based on a rich feature space constructed using trajectory fragments. In ROAM, raw trajectories are partitioned into fragments. These fragments are overlaid on top of each other and the common patterns



(a) Two similar trajectories. The loop in the right trajectory is difficult to handle in holistic approaches.



(b) Same two trajectories after motif extraction. The right trajectory has an extra m_1.

Figure 5.1: Motif representation

become what we call **motifs**. Motifs are represented by a tuple (motif expression) which includes additional spatiotemporal attributes that may be helpful in analysis. The set of motif expressions observed then forms a feature space in which the original trajectories are placed. Using such a feature space, we can leverage algorithms in machine learning and data mining to learn complex associations between trajectory fragments and also other important information.

A *motif* is a prototypical movement pattern. Examples include *right turn*, *u-turn*, and *loop*. One could view them as parallels to gene expressions in DNA sequences or entity mentions in text. Fig. 5.1(b) shows the motifs in Fig. 5.1(a) as drawn by the dotted circles. In this form, the two trajectories now have much in common: They share one m_2 and two m_4's, and differ in one m_1 (*i.e.*, *loop* motif).

5.1.2 Multi-Resolution Feature Hierarchies

Another observation we make is raw recordings and semantic analysis often occur at different spatiotemporal granularities. While time recordings may be made at the minute or second level, analysis is usually more sensible on the hour level or even higher. The same scenario applies to the location measure. One might record at the meter level but analyze at the city block or district level.

By using a more general representation, fewer distinct measure values are used and the analysis task could become easier. In addition, it would improve human readability. If the final classification model produces human readable results (as ROAM does), having high level features not only reduces the size of the results but also increases their understandability.

Sometimes, these hierarchies are readily available, as in the case of time. With other features, however, it may not be obvious. In ROAM, we use a clustering-based technique to automatically extract hierarchies based on the behavior of the trajectories. Given such hierarchies, it is still hard to know a priori which levels will work the best for classification so we let ROAM adjust dynamically.

5.2 Framework

Figure 5.2 shows the ROAM (Rule- and Motif-based Anomaly Detection of Moving Objects) framework. Square boxes are computation modules, round boxes are data sources, and arrows show the flow of data. There are three computation modules in ROAM: Motif Extractor, Feature Generator, and Hierarchical Rule-based Classifier. Data flows through them in that sequence.

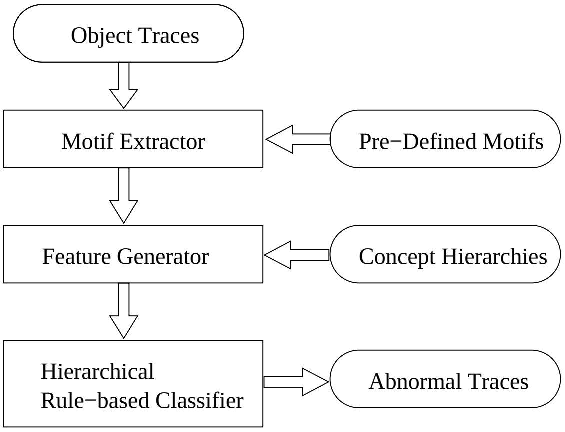


Figure 5.2: ROAM Framework

5.2.1 Motif Extractor

The first computation module in ROAM is the Motif Extractor. A motif is a prototypical movement pattern. ROAM use a sliding window technique to process the trajectories. All windows are overlaid on top of each other and clustering is used to group them into representative sets. These representative ones then form the set of interesting *motifs* in $\mathcal{D}$.

Given a trajectory, we slide a window of length ω across it. ω could be defined with respect to time or distance. If it is time, then different speeds (and thus distance traveled) would result in different motifs. If it is distance, then speed variances would be normalized. In our experiments, we used time since speed was relatively stable though more complex data might require more complex normalization methods. For each resultant window w, we compute the vector from the first point in w to the last point in w. The width of the vector is then expanded to accommodate all other points within w; this bounding box allow us to smooth over noises in the trajectory.

All bounding boxes are overlaid on top of each other. And using the Euclidean distance, we cluster them to find the representative patterns. The resultant cluster centers then define the set of motifs. In ROAM, a motif is represented just like a window: a vector with a bounding box around it. Depending on the task, other variables may be kept as well. Once the motifs are set, we then go through $\mathcal{D}$ again using the same sliding windows. This time, a window w in a trajectory is *similar* to a motif m if $||w - m|| \leq \epsilon$. And if a particular window is similar to a motif, we say that motif is "expressed" in the trajectory.

A natural question to raise is how to set ω. A too small of a value could miss motifs by dividing them into indistinguishable pieces and a too large of a value could bundle motifs together and lose discriminative power. Fortunately, it turns out that most reasonable values will perform just fine. As we will show empirically in Section 4.4, classification accuracy is fairly robust with regards to different ω values.

Given a trajectory, the motif extractor returns the sequence of **motif expressions** found in the trajectory. Each motif expression has the form

$$(m_i, t_{start}, t_{end}, l_{start}, l_{end}) \tag{5.1}$$

where m_i is the motif, t_{start} and t_{end} are the starting and ending times, and l_{start} and l_{end} are the starting and ending locations. The complete sequence is known as the **motif trajectory** of the original trajectory.

Motif Expression Attributes

The form of motif expression shown in Eq. (5.1) is only the first step in full motif expression extraction. Additional information on when, where, and how the motif was expressed is needed. Take Fig. 5.3 as an example. There are two objects moving in an area with the same trajectories; however, the left one is near an important landmark. This extra piece of information (*i.e.*, proximity to landmark) can be crucial in decision making. If we also knew that the left object was moving slowly during the middle of the night, the combination of all such information is telling in anomaly detection.

For each motif expression, we introduce a set of **attributes** in addition to the simple time

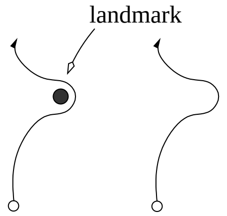


Figure 5.3: Two objects moving with the same trajectory.

and location ones in Eq. (5.1). Some examples include `duration`, `top_speed`, `avg_speed`, `radius`, and `general_location`. Some of these attributes can be derived easily from the time and location attributes, *e.g.*, `avg_speed` $= \textit{path-distance}(l_{start}, l_{end}) \div (t_{end} - t_{start})$. Others may require a more sophisticated motif extractor.

Let there be A such attributes: $\{a_1, a_2, \ldots, a_A\}$. We now represent each motif expression as follows,

$$(m_i, v_1, v_2, \ldots, v_A) \tag{5.2}$$

where m_i is the motif and v_i is the value of attribute a_i. Note that a_i may be continuous or even multi-dimensional.

5.2.2 Feature Generator

Once the motif expressions have been extracted, semantic analysis can begin. One could try the following naïve classification scheme. For each distinct (*motif, attribute, attribute value*) combination we see in the trajectory data, we map it to a feature. For example, (right-turn, `speed`, $11mph$) would map to a feature and (right-turn, `speed`, $12mph$) would map to another feature. Formally, $\forall\ i, j, k\ (m_i, a_j, v_k) \leftrightarrow f_x \in \mathcal{F}$ where $\mathcal{F}$ is the resulting feature space. We then use the following classifier.

This particular transformation from trajectories to a feature space is complete. Every motif attribute value is preserved as a feature and the frequencies of their expressions are preserved as the feature values. However, it is ineffective by the following observations. First,

```
1. Transform the motif-trajectories into vectors in the $\mathcal{F}$
   feature space. Suppose $f_x \leftrightarrow (m_i, a_j, v_k)$.
   Then the $x^{th}$ component of the vector
   has the value of the number of times
   motif $i$'s attribute $j$ expressed value $k$ in the trajectory.
2. Feed the feature space and the data points as
   input into a learning machine.
```

a large number of distinct motif attribute values leads directly to a high dimensional feature space. Specifically, suppose there are M motifs, A attributes, and each attribute has V distinct possible values. The instance space is then $\mathbb{N}^{MAV}$. Second, the high granularity or continuous motif attribute values make generalization difficult. Because these distinct values are transformed to distinct features, generalization becomes essentially impossible. Learning on a feature that is at `10:31am` will have no bearing on a feature that is at `10:32am`.

Feature Generalization

In order to overcome the difficulties in the Flat-Classifier, generalization in the feature space is needed. For example, (*right-turn*, `time`, $2am$), (*right-turn*, `time`, $3am$), and (*right-turn*, `time`, $4am$) features could be generalized into one feature: (*right-turn*, `time`, *early_morning*). This not only reduces the dimensionality of the feature space but also helps the learning machine through feature extraction.

Recall that each feature has the form (m_i, a_j, v_k), where each attribute a_j is either numerical (1D) or spatiotemporal (2D or 3D). We assume that each a_j has a distance metric defined on its values. Thus, features having the same m_i and a_j values (*i.e.*, $(m_i, a_j, *)$) can be compared with a formal distance metric. For example, (*right-turn*, `time`, $2am$) is more similar to (*right-turn*, `time`, $2:02am$) than (*right-turn*, `text`, $6pm$). But, it does not make sense to compare features with different m_i or a_j values (*e.g.*, (*right-turn*, `time`, $2am$) is not comparable to (*u-turn*, `speed`, $10mph$)).

We partition the features in $\mathcal{F}$ into sets with distinct $(m_i,\ a_j)$ values. If there are M motifs and A attributes, there are $M \times A$ disjoint sets. We propose to generalize the features in each (m_i, a_j) set into a smaller set. Further, this new set will be hierarchical where appropriate. This will be the task of the **Feature Generator** in the ROAM framework. Specifically, it will

1. discretize or cluster continuous or high granularity motif attribute values.

2. form a hierarchy over the attribute values, which in turn offers a multi-resolution view of the data.

We will treat each (m_i, a_j) space independently. Since the attribute values can have different forms (*e.g.*, numerical values, 2D spatial locations), we will use different methods where appropriate. We explain them in detail in the following two sections.

Spatial Attributes

Attributes such as `location` are spatial points in a 2D or 3D space. In such scenarios, we use a hierarchical "micro-clustering" technique similar to BIRCH [92] to discover prototypical patterns. Features are inserted into a tree-based data structure where nodes represent micro-clusters. A micro-cluster is a small, tightly grouped neighborhood, and features belong to the same micro-cluster only when they are closely related. A tree of these micro-clusters represents a concept hierarchy of the attribute.

Take the `location` attribute as an example. A micro-cluster may only include features which are within a few meters of each other. During insertion of features into the tree, each micro-cluster has a maximum radius parameter. If a feature cannot fit inside a micro-cluster, a new micro-cluster is created. The tree also had a maximum branching factor so insertions and rotations occur like a typical B-tree. After all features have been inserted into the tree, the leaf nodes form the set of micro-clusters. Each micro-cluster can be viewed as a meta data point that represents similar features. We then feed the set of micro-clusters into a hierarchical agglomerative clustering algorithm to construct the final hierarchy.

The final clustering tree is hierarchical in the following sense: any node in the tree contains summarized information for all data points in that node's subtree. For example, the root contains a summary of the entire tree. The summary information is sufficient for the calculation of the centroid and the radius of the points. The reason we choose a BIRCH-like algorithm in our system is two-fold. First, it performs micro-clustering, which fits our needs better. Second, building the CF tree is time and space efficient ($O(n)$). More properties are described in [92].

Numerical Attributes

Attributes such as `time` and `avg_speed` are numerical. Usually, in the presence of continuous attributes, *discretization* is performed. Doing so has many advantages. First, it makes the learning problem easier. A decision tree, for example, would have fewer splits to consider. A discrete feature allows better generalization. Second, it makes human readability easier. For instance, it is much easier to understand the feature value of `1pm-2pm` as opposed to reading all the distinct values between `1pm` and `2pm`.

Discretization techniques [56] can be split into two main groups: unsupervised and supervised. Since we have labeled data, supervised algorithms are more appropriate. There is an abundant number of methods available for this; most of them would function just fine here. An additional requirement we have is a hierarchy over the resultant discrete values. While most discretization methods do not have this property, we can easily add it by performing hierarchical agglomerative clustering as a post-processing step.

Since spatial attributes are a generalization of numerical attributes, we use the same clustering methods in our implementation of ROAM for both types of attributes. Clustering in one-dimensional data still provides meaningful groupings based on behavior.

Multi-Resolution View

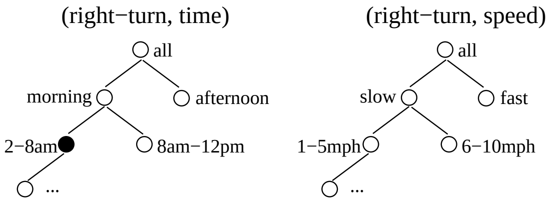


Figure 5.4: Two sample motif-attribute hierarchies

After building hierarchies in each of the (m_i, a_j) spaces, the overall feature space is now structured as a set of hierarchies. Fig. 5.4 shows a partial illustration. In it, there are two **motif-attribute hierarchies**: (*right-turn*, `time`) and (*right-turn*, `speed`). Each node corresponds to a micro-cluster feature discovered in $\mathcal{F}$. For example, the black node in Fig. 5.4 represents all right-turns taken between 2 and 8am. High level nodes in the hierarchies correspond to high level features and low level nodes correspond to low level features. By choosing different subsets of the nodes, a user can create distinctly different views of the data. For example, suppose one only used level one features in Fig. 5.4 (*i.e.*, "morning", "slow", etc). This generates a very rough view of the data and with only four features. On the other hand, choosing the leaf nodes in Fig. 5.4 generates a detailed view but with many more features.

As mentioned previously, concept hierarchies may already exist for some attributes (*e.g.*, time). In such cases, one may just choose to use them to construct the motif-attribute hierarchy. However, in other cases or sometimes in place of the existing hierarchies, one could use automated techniques in ROAM to construct the hierarchies. This has the distinct advantage that the hierarchies are built based on the behavior of the data. As a result, more features could be dedicated to the dense regions and fewer features to the sparse regions. Clustering and discretization techniques can adjust dynamically based on the data and could facilitate more effective analysis.

5.2.3 Classification

Let $\mathcal{F}'$ be the set of all nodes in the motif-attribute hierarchies. $\mathcal{F}'$ is the largest feature space where all resolutions are included. Though this feature space is "complete", it is unlikely to be the best one for classification. It creates a high dimensional feature space; this makes learning slow and possibly ineffective. On the other hand, suppose we choose only the root level nodes in all the motif-attribute hierarchies. That is, only the "all" nodes in Fig. 5.4. The problem with this feature space is that the features are too general to be useful. What we seek is something in between those two extremes.

To this end, we propose a rule-based classification method: CHIP (Classification using Hierarchical Prediction Rules). We chose a rule-based learning algorithm for several reasons. First and foremost, it produces human-readable results. This is useful in practice. Second, it is efficient. CHIP is $O(N)$ with respect to either the number of examples or the number of features. Other classifiers such as Naïve Bayes or SVM are $O(N^2)$ with respect to one. The problems we are dealing could be large and high dimensional. Lastly, the classification problem is unbalanced: the abnormal class has few training examples. In such contexts, rule-based learner have been shown to be effective [18].

Intuitions

Before formally describing CHIP, we give some intuitions. CHIP iteratively and greedily searches for the best available rule until all positive examples are covered. In addition, CHIP tries to use high-level features whenever possible. For example, suppose all ships that move at location X between the hours of *12pm* and *5pm* are normal. Then a single rule using the *afternoon* feature will suffice. Using this principle has several benefits. First, the feature space is kept to a small size. This speeds up learning and keeps the problem tractable. Second, features are kept high level whenever possible. This produces rules which are general and easily understood by a human. Third, it avoids the problem of over-fitting.

In machine learning research, the study of feature space simplification or generalization is known as *feature selection* [26]. Given a set features, choose a subset which will perform better (in terms of efficiency and/or accuracy) in the learning task. A typical approach scores each feature and iteratively inserts or removes them. In our setting, however, we have something that is different than the standard setting: there are hierarchical structures over the features. Thus, selection should be a little smarter.

With this in mind, we propose a top-down search in the feature hierarchies. We start with an initial high level feature space and try to describe the data (in the rules sense). If these features produce accurate rules, we are satisfied. But if at some point we find that a more specific feature will produce a better rule, we *expand* the existing feature space to include that specific feature. This process repeats until all the training data is sufficiently covered.

CHIP

We introduce some definitions first. CHIP learns a set of rules, much like FOIL [65] and CPAR [85]. A single rule r has the conjunctive form of:

$$l_1 \wedge l_2 \wedge \ldots \wedge l_n \rightarrow c$$

where each l_i is a literal (or predicate) of the form $(feature = value)$ and c is the class label. An example is "covered" by r if all the literals in r are satisfied in the example. Next, recall that $\mathcal{F}'$ is the complete set of features. For any feature f in $\mathcal{F}'$, let $Exp(f)$ return the set of f's children in $\mathcal{F}'$'s hierarchy. For example, $Exp(\texttt{morning}) = \{\texttt{2-8am}, \texttt{8am-12pm}\}$. At any time, CHIP uses a subset of the features in $\mathcal{F}'$. Let $\mathcal{F}_{\mathsf{C}}$ be this set.

A rule is learned one literal at a time. Literals are selected according to a weighted version of $Foil\_Gain$ [65], which is based on the positive and negative coverage of the rule before and after adding the literal. Let p_0 and n_0 be the number of positive and negative examples covered by rule r without literal l. Let p_1 and n_1 be the number of positive and negative examples covered by rule $r \wedge l$. $Foil\_Gain(l, r)$ is then defined as

$$p_1 \left(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0} \right)$$

The weighted version of *Foil_Gain* [85] allows previously covered positive examples to be used again but just weighs them down. This adjusts the p and n values appropriately in the above equation.

In the previous section, we gave the intuitive notion of discovering that a more specific feature will perform better than a current feature. Here, we formalize this notion in the function $Exp\_Gain(f, r)$ where f is a feature and r is a rule. It is defined as

$$Exp\_Gain(f, r) = \max_{(l, f_i) \forall l, f_i \in Exp(f)} Foil\_Gain(l, r)$$

The *Exp_Gain* (expansion gain) of a feature is the maximum *Foil_Gain* achieved by any literal in any of its child features. It is defined with respect to a non-empty rule similar to *Foil_Gain*. We chose this function because it allows sensible direct numerical comparisons between *Foil_Gain* and *Exp_Gain*.

Discussion CHIP starts with all examples uncovered and iteratively searches for the best rule to cover the positive examples. The search is greedy and halts when enough positive examples are covered. Rules are learned one literal at a time, choosing them based on *Foil_Gain* (line 4). In line 5, the *Exp_Gain* of each feature is calculated. If the better gain is *Foil_Gain*, the literal is added to the current rule (line 8). Otherwise, the feature space is expanded (line 10) and the process repeats.

Complexity CHIP has running time of $O(nSR)$ where n is the number of examples, S is the size of the used feature space, and R is the number of learned rules. In our implementation, we collapsed examples (trajectories) which appear the same into meta-examples. Thus, with a high initial feature space, n can be quite small if the data is skewed. S can also be small initially since there are only a few high level features. As the algorithm executes, both n and S will increase with feature expansion. This is another reason to avoid careless expansion.

5.3 Experiments

In this section, we show our framework's performance in a variety of settings. We conduct our experiments using both real and generated data to show efficiency and effectiveness. For data generation, we used GSTD [78] (which generates raw trajectories) and also our own data generator (which generates motif-trajectories). The latter allows us to test some parts of the framework independently of others. Efficiency experiments were run on an Intel Pentium 4 2.6GHz machine with 1.5GB of memory. The Motif Extractor was written in Python and the rest was written in C++ and compiled with GCC.

Each dataset consists of two classes: normal and abnormal. In GSTD, we achieve this by generating two datasets with slightly different parameters. In our own generator, the base data is motif-trajectories. A set of motif-expression seeds are initialized in the model and a Gaussian mixture model is used to create randomness. We generate the abnormal class by mixing "abnormal" motifs with a background model that is shared between both the normal and abnormal classes.

There are two parameters which controls CHIP. One is the starting level in the motif-attribute hierarchy. Level 0 denotes the root level. The other is β, the feature expansion weight. These two parameters are indicated as ROAM(*starting_level*, β). Finally, since the number of abnormal examples is small, we used the standard F1 metric instead of accuracy.

F1[2] is a harmonic mean of recall and precision and better reflects the effectiveness of the classifier. An F1 score of 100 indicates 100% recall and precision. F1 scores were the result of 10-fold cross validation. Experiments were run 5 times to get an average.

5.3.1 Real Data

We obtained real ship navigational data from the Monterey Bay Aquarium Research Institute (MBARI[3]). Under the MUSE project, several ships traveled in ocean waters near Northern California to conduct various aquatic experiments. The ships' navigational data, which includes time, longitude, and latitude, were recorded. Depending on the ship, the sampling rate varied from 10 seconds to a few minutes; the end result are fairly continuous paths. Figure 5.5 shows a typical path of a vessel named Point Sur.

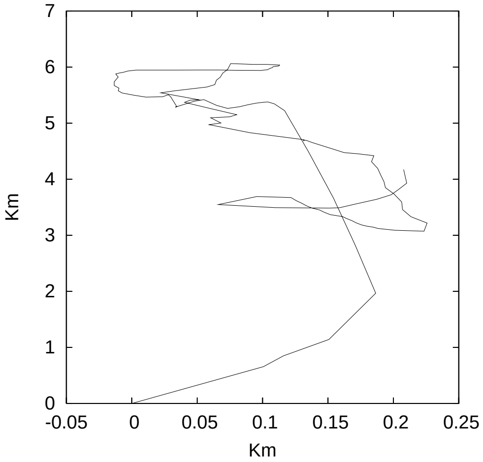


Figure 5.5: Path of ship Point Sur from 16:00 to 24:00 on 8/23/00 starting at point $(0, 0)$.

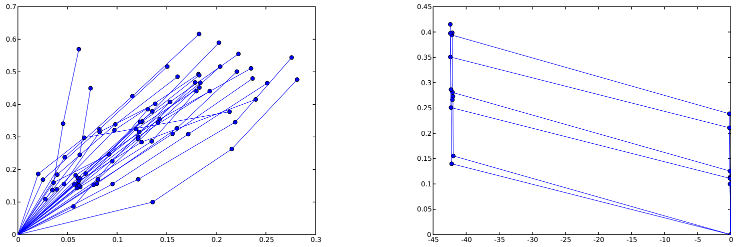
We collected data from two different ships (namely Point Sur and Los Lobos) and assigned different class labels to them. The two ships carried out different tasks and thus naturally had different movement patterns. There was a total of 23 paths (11 of one, 12 of another), each with 1500 to 4000 points. Using ROAM, we extracted 40 motifs, constructed features,

---

[2] $F1 = (2 \times recall \times precision) \div (recall + precision)$

[3] `http://www.mbari.org/MUSE/platforms/ships.htm`

and tried to recover the class labels using CHIP. Figure 5.6 shows two sets of trajectory segments that were marked as motifs 10 and 14. Motif 10 is a simple straight trajectory towards the northeastern corner. Motif 14 is a 3-part move of going north, northwest, and then north again.



(a) Extracted motif 10 (b) Extracted motif 14

Figure 5.6: Extracted motifs from MBARI data.

Motifs were extracted from a window of approximately 3 minutes, and had two additional attributes. One is the distance traveled, which indicates speed, and the other is the general Euclidean distance to the stored motif. We did not include the time-of-day attribute since the two ships had regular but different schedules and including them would make the problem too easy. Motif-attribute hierarchies (branching factor of 4) were also generated, which ranged from 2 levels deep to 7 levels deep.

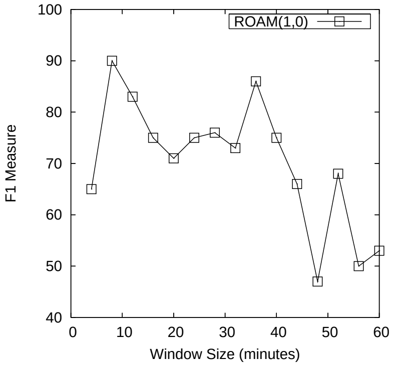


Figure 5.7: Effect of ω on classification accuracy.

An issue raised before was the setting of ω, the width of the window to extract motifs.

Figure 5.7 shows the effect on classification as ω increases from 4 minutes to 60 minutes on the MBARI data. As shown, accuracy with too small or too large of a window is poor, but in the intermediate, it is relatively stable. Thus, we believe that as long as the window is reasonable, performance should not be affected too much. Another issue is how *many* motifs to extract. This was set to 40 in Figure 5.7, and Figure 5.8 shows the effect as that number changes from 5 to 60. The curve shows that we were able to achieve 100% classification accuracy with 10 and 15 motifs. And as the number increases, accuracy decreases but not too drastically. In general, a reasonable number should not be too difficult to find.

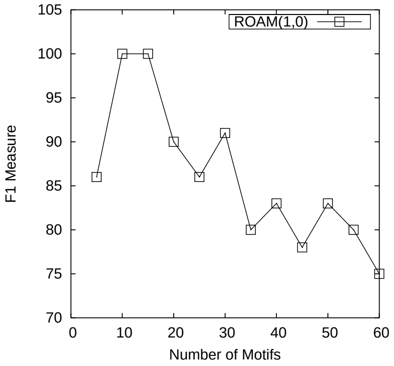


Figure 5.8: Effect of number of motifs on classification accuracy.

5.3.2 Synthetic Data

While the real data experiments provided some validation of our methods, we were unable to thoroughly test other aspects due to the small dataset size. To combat this, we experimented with synthetic data from GSTD and also our own data generator.

Notation

For our own data generator, we use the following notation to denote the parameters used in generation. Each data set's name is in the form of "$N\#B\#M\#A\#S\#L\#$", where N is the number of normal trajectories, B is the number of abnormal ones, M is the number of motifs, A is the number of attributes, S is the standard deviation in the Gaussian mixture distributions, and L is the average length of the trajectory.

Classification Accuracy

First, we tested accuracy using GSTD. In GSTD, we generate two different classes of data using Gaussian distributed movement centers. The two models shared the same parameters except the mean of the centers differed by 0.01 (0.50 vs. 0.51). Fig. 5.9 shows the results as we also varied the variance in the distributions. As expected, accuracy improves as the trajectories differ more from each other. But even at small differences, ROAM was able to distinguish the two classes.

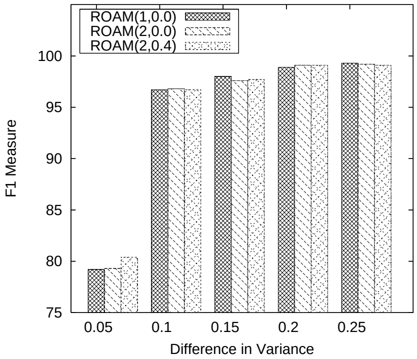


Figure 5.9: GSTD $N2000B200M30$: Accuracy with respect to difference in variance.

Next, we tested the accuracy using our own data generator. Fig. 5.10 shows F1 results as the number of motifs in the data increased from 10 to 100 on the y-axis. For comparison, we used SVM[4] (nu-SVC with radial kernel) with the Flat-Classifier as described before and also SVM with level 2 features. The first thing we notice is that SVM with Flat-Classifier is hopeless, as expected. We also observe that ROAM with level 1 features and a little bit expansion is almost as good as SVM with level 2 features. ROAM with level 2 and a little bit expansion is equal to or better than SVM.

---

[4] `http://www.csie.ntu.edu.tw/~cjlin/libsvm`

We note that the size of the classification feature space is much larger than the number of motifs. For example, when the number of motifs equals 100, the number of level 2 features equals nearly 1200.

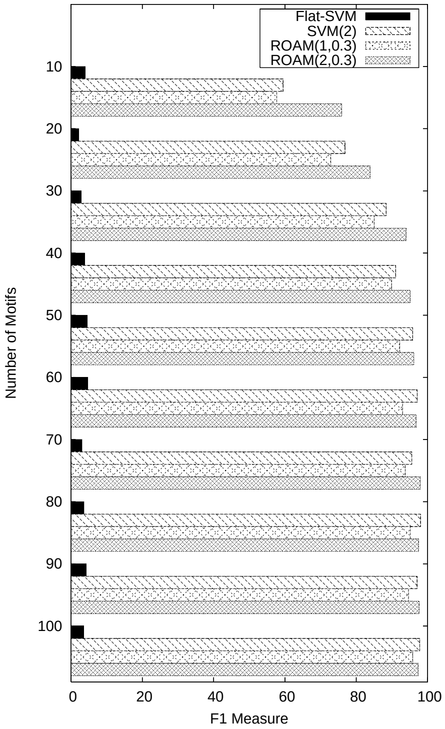


Figure 5.10: $N4kB200A3S5.0L20$: Accuracy with respect to number of motifs.

Fig. 5.11 shows F1 as the motif-trajectory length varies. As the length increases, the data gets denser and we observe that SVM's performance deteriorates. However, ROAM with its various configurations were fairly stable. Fig. 5.12 shows the effect as standard deviation is increases from 5 to 40. As expected, F1 decreases as the values get more spread out. One might have noticed that we have rather large standard deviation values. This is because the range of values is large ($\sim$1000).

Recall that a larger value of β, the expansion factor, increases the chances that CHIP will expand the feature space during learning. The effect of different β values vary from one dataset to another. Fig. 5.13 shows a typical result. In the ROAM(1,$*$) curve, ROAM starts

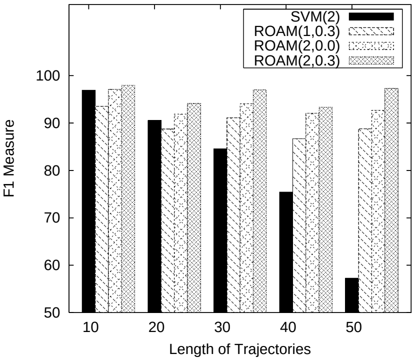


Figure 5.11: $N4kB200A3S5.0L20$: Accuracy with respect to length of motif-trajectories.

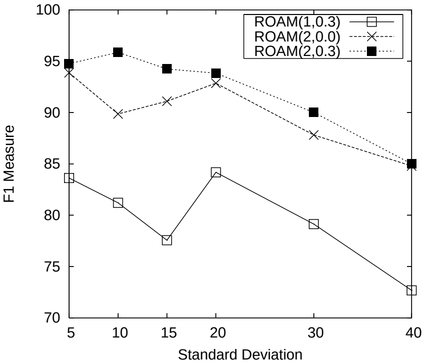


Figure 5.12: $N5kB100M20A3L20$: Accuracy with respect to standard deviation.

with level 1 features and improves significantly with expansion. In the ROAM(2,$*$) curve, F1 is high initially. It improves slightly with some expansion but eventually drops down. This is the effect of over-fitting. In other words, CHIP has expanded too greedily and the feature space has become too specific.

Finally, Fig. 5.14 compares a general feature space vs. a specific one. One is ROAM(2,0), which is level 2 features with no expansion. The other is ROAM(MAX), which is only the leaf features. We see that ROAM(2,0) is significantly better in accuracy. Furthermore, it is also faster. With 60 motifs, ROAM(2,0) took an average of 84 seconds with 705 features while ROAM(MAX) took 850 seconds with approximately 4350 features.

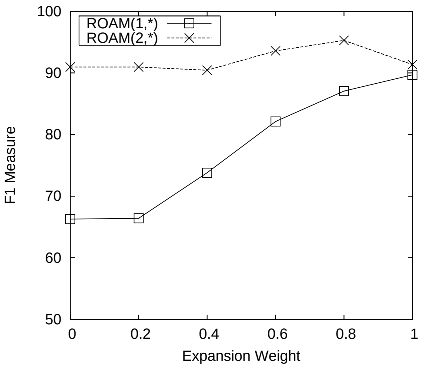


Figure 5.13: $N500B100M20A3S25L20$: Accuracy with respect to β.

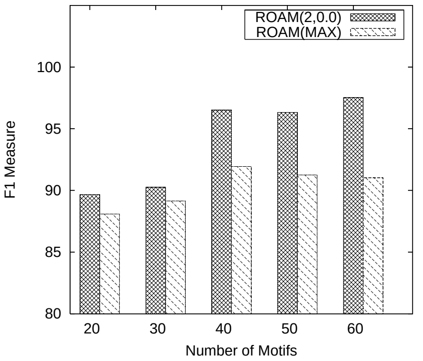


Figure 5.14: $N15kB500A3S25L20$: Accuracy with respect to number of motifs.

Efficiency

With regards to efficiency, we first check sensitivity to the number of trajectories. Fig 5.15 shows a plot broken down into ROAM's components, note the log scale. As we can see, all components scale nicely with respect to the number of trajectories. The Motif Extractor is the slowest, but it was implemented in Python (10–30 slower than C++) while the other components were in C++.

Another aspect of efficiency is sensitivity to the length of the trajectories. Fig. 5.16 shows the running time as the length was increased from 10 to 100 in our own data generator. Fig 5.17 shows a similar experiment using GSTD data. Again, we see a linear increase in running time as trajectory length increased. The reason is that with longer trajectories, there is a

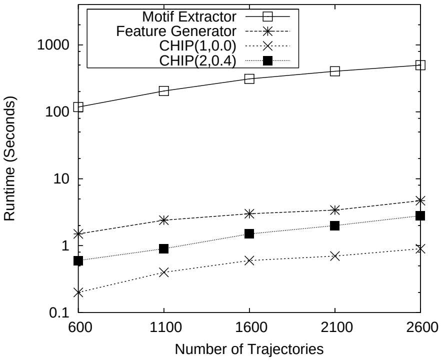


Figure 5.15: GSTD: $B200M20$: Efficiency with respect to number of trajectories.

linear increase in the number of motif expressions **ROAM** has to process.

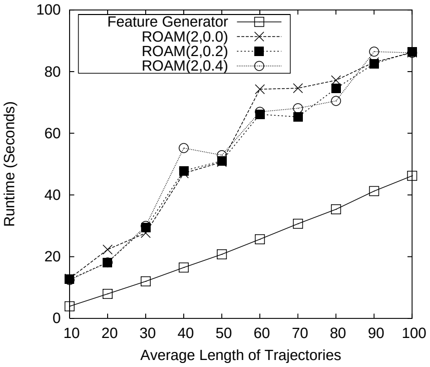


Figure 5.16: $N20000B1000M20A3S10.0$: Efficiency with respect to length of motif-trajectories.

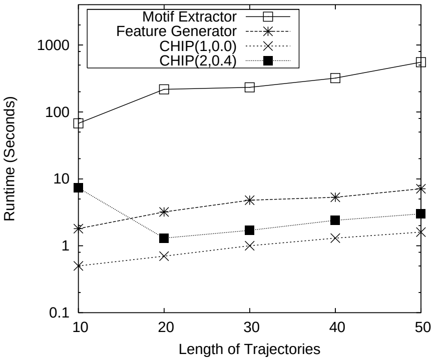


Figure 5.17: GSTD: $N2000B100M10$: Efficiency with respect to length of trajectories.

Input: (1) Training set $\mathcal{D} = P \cup N$, where P and N are the positive and negative examples. (2) Initial feature set $\mathcal{F}_{\mathsf{C}} \in \mathcal{F}'$.

Output: Set of classification rules R.

Method:

```
while not all of $P$ is covered
    initialize new rule $r$
    while true
        find literal $l$ with highest $Foil\_Gain(l, r)$
        find feature $f$ with highest $Exp\_Gain(f, r)$
        if both gains < min_gain then break
        if $Foil\_Gain(l, r) > \beta \cdot Exp\_Gain(f, r)$ then
            add $l$ to $r$
        else
            add feature $f$ to $\mathcal{F}_{\mathsf{C}}$
    add $r$ to $R$
return $R$
```

Chapter 6

Subspace Outliers in Multidimensional Data

As mentioned, one of the differences between the ROAM system and other traditional trajectory clustering or outlier detection algorithms is the combination of multi-dimensional information with the trajectory. This additional information can be incorporated nicely into a classification scheme [50]. However, for unsupervised learning (*i.e.*, clustering or anomaly detection without training), it is a different question. The reason is that there is an exponential number of subspaces within the possibly very high dimensional space and the anomaly could occur in any one of them.

We studied exactly this problem except with respect to time series [47], although the framework can be easily adapted to moving objects. The framework aims to discover non-trivial anomalies in a time-series data cube given some query. A brute force solution can be realized if the number of attributes (dimensions) is small. However, this would encounter major challenges in many real-world problems, where the data is high dimensional (such as $\sim$100). It is impossible to materialize a full cube in a high-dimensional space. Moreover, because anomaly in general does not have monotonic property [2], pruning in this large space is difficult.

We address this difficult problem via a divide-and-conquer approach [48, 47]. Because anomalies are rare by definition, many of the 2^n subspaces (generated from n dimensions) are not correlated with anomalies. We make two main contributions. First, the high-dimensional data is partitioned automatically to discover interesting subsets of dimensions *and* tuples. Each subset forms a small data cube in itself, and we further propose an efficient top-k cube anomaly mining algorithm. The combination of these two techniques leads to an efficient discovery of the global top-k cube anomalies.

More specifically, we propose an iterative subspace search algorithm named **SUITS** (<u>Su</u>bspace <u>I</u>terative <u>T</u>ime-<u>S</u>eries Anomaly Search) to mine top-k anomaly cells. Given a *query probe cell* in a data cube, one would expect that a descendant cell, which is a subset, should roughly follow a similar behavior. This leads to the computation of an *expected time series* and also the *anomaly measure*, which measures the difference between the expected and observed time series. Descendant cells of the probe cell are partitioned by their anomaly type and amount. For each partition, a correlated subspace data cube in the original high-dimensional space is extracted, and efficient top-k anomaly mining is performed on it. This process iterates for all partitions. Each partition produces a local top-k, and they merge to form an approximation of the global top-k. In experiments with real world sales data, this was shown to be both effective and efficient.

6.1 Problem Definition

6.1.1 Preliminaries

Time Series. A time series $s_i(t)$ is a sequence or function which maps time values, t, to numerical values. The range of t is usually restricted to some interval, and they are typically discrete values. A time series can represent any type of temporal data. For instance, the sequence $s(t) = \langle 5, 10, 13, 7, 2 \rangle$ could represent the daily sales of televisions at a store over a 5-day interval: $t = [0, 5]$.

Data Cube. Given a relation R, a *data cube* (denoted as C_R) is the set of aggregates from all possible group-by's on R. In an n-dimensional data cube, a cell $c = (a_1, a_2, \ldots, a_n : m)$ (where m is the cube measure) is called a k-dimensional group-by cell (*i.e.*, a cell in a k-

dimensional cuboid) if and only if there are k ($k \leq n$) values among $(a_1, a_2, \ldots, a_n)$ which are not $*$ (*i.e.*, all). Given two cells c_1 and c_2, let V_1 and V_2 represent the set of values among their respective $(a_1, a_2, \ldots, a_n)$ which are not $*$. c_1 is the *ancestor* of c_2 and c_2 is a *descendant* of c_1 if $V_1 \subset V_2$. c_1 is the *parent* of c_2 and c_2 is a *child* of c_1 if $V_1 \subset V_2$ and $|V_1| = |V_2| - 1$. These relationships also extend to cuboids and form a structure called the *cuboid lattice*. An example is shown in Figure 6.1. The "All" or *apex* cuboid holds a single cell where all its values among $(a_1, a_2, \ldots, a_n)$ are $*$. On the other extreme, the *base* cuboid at the top holds cells where none of its $(a_1, a_2, \ldots, a_n)$ values is $*$.

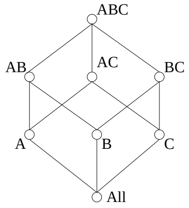


Figure 6.1: Cuboid lattice

Input Data. Consider a relation R with n attributes A_1, A_2, $\ldots$, A_n. Each attribute A_i contains discrete values. Let there be t tuples in this relation with transaction IDs of $tid_1, tid_2, \ldots, tid_t$. Let there also be a set of time series $S = \{s_1, s_2, \ldots, s_t\}$ where s_i is associated with tuple tid_i.

Gender	Education	Income	Product	Profit	Count
Female	Highschool	35k–45k	Food	s_1	u_1
Female	Highschool	45k–60k	Apparel	s_2	u_2
Female	College	35k–45k	Apparel	s_3	u_3
Female	College	35k–45k	Book	s_4	u_4
Female	College	45k–60k	Apparel	s_5	u_5
Female	Graduate	45k–60k	Apparel	s_6	u_6
Male	Highschool	35k–45k	Apparel	s_7	u_7
Male	College	35k–45k	Food	s_8	u_8

Table 6.1: Input market segment data

A market analysis sample data is shown in Table 6.1. Each of the four A_i's represents an attribute on either the customer or the product. Each tuple in this relation corresponds to a market segment, and the associated time series measures the "Profit" over time. For all s_i in S, the period and sampling rate are assumed to be the same (otherwise, preprocessing and normalization can be performed). In addition, count u_i is associated with each market segment. It records the number of records (*i.e.*, people) in that segment, and its value is initialized to 1 by default if the field was originally nonexistent.

6.1.2 The Anomaly Search Problem

Given a relation R and its associated time-series set S, a probe cell $p \in C_R$, and an anomaly function g, find the **anomaly cells among descendants of p in C_R as measured by g**. To search all data and perform a global analysis, one can simply set p to empty; the rest of the algorithm remains unchanged.

To make the algorithm more practical, we add three extra conditions: (1) each abnormal cell must satisfy a minimum count (support) threshold, which eliminates trivially small market segments, (2) anomalies do not have to hold for the entire time series, *i.e.*, g can be applied to sub-sequences, and (3) only the **top k anomaly cells as ranked by g** are returned. These conditions better match usage habits of analysts but are not critical to the core algorithm. For example, instead of condition (3), one may ask for all the anomalies larger than a fixed threshold.

Notice that although p is a cube cell, its description is like a selection query. This can be treated as a selection query, $\sigma_p(R)$, *i.e.*, select exactly the set of tuples in R which satisfy p in C_R.

For each cell c in C_R, there is an associated time series, denoted as s_c, and called an **observed time series**. In the context of a query probe p, s_c is computed by aggregating time series from S whose corresponding tid's are in $\sigma_p(R)$ and also c.

$$s_c = \sum_{tid_i \in \ c \ \cap \ \sigma_p(R)} s_i \tag{6.1}$$

In market analysis, S is typically a numerical measure, such as sales or profit. The aggregation function can be either SUM or AVG, depending on the application semantics. Here we take SUM by default. In general, any distributive or algebraic aggregation function may be substituted.

At each cell, we will also calculate an **expected time series**, denoted by $\hat{s}_c$. The measure of anomaly will be a function on the observed and expected time series, *i.e.*, $g(s_c, \hat{s}_c) \rightarrow R$. Since the topic of the paper [47] was time series, we developed several measures of g with respect to time series data. Because they are not pertinent to our discussion of moving objects, we will skip them. Suffice it to say, g measures the difference between s_c and $\hat{s}_c$ and returns an anomaly type and anomaly amount. To adopt g to moving objects would require taking one of the building blocks of similarity between trajectories and fitting it to g. We studied exactly in a recent work [45], but it has not been integrated into a multi-dimensional framework yet.

6.1.3 Ranking Anomalies in Data Cube

With g defined for the anomaly types, we can rank all descendant cells of p in descending order according to their absolute g values. In all four types, a larger absolute g value indicates a more substantial anomaly. The original query would then return the top-k market segments in this ranking. However, because the four types of g's are incompatible, it may not make sense to rank them together. Rather, it is more sensible to have a separate ranking for each distinct g. As a result, the top-k would be on individual types. With four types of anomalies defined, the final result would consist of $4k$ market segments.

6.2 Mining Top-K Anomalies in Data Cubes

With g defined, we return to the original problem of finding top-k anomaly cells among the descendants of p. A naïve solution to this problem is given in Algorithm 2. Its main observation is that C_R is unnecessary because the query only focuses on p. Thus, it only computes the data cube C_p using $\sigma_p(R)$ as the fact table. After C_p is constructed, the top-k anomaly cells within it are returned.

Algorithm 2 Naïve Top-k Anomalies

```
Input: Relation $R$, time-series data $S$, query probe cell $p$,
    anomaly function $g$, parameter $k$, minimum support $m$
Output: Top-$k$ scoring cells in $C_p$ as ranked by $g$ and
    satisfies $m$

1.  Retrieve data for $\sigma_p(R)$
2.  Compute the data cube $C_p$ with $\sigma_p(R)$ as the fact table
        with $m$ as the iceberg parameter
3.  Return top $k$ anomaly cells in $C_p$ for each $g$
```

The core difficulty with Algorithm 2 is how to deal with the high dimensional space; if there are n attributes in R, there are 2^n cuboids (subspaces) in C_p to examine in order to produce the final answer. This effectively prohibits full materialization of C_p for a medium n even if $\sigma_p(R)$ does not contain many tuples. To solve this problem, we propose a new algorithm SUITS, which iteratively select subspaces with the most potential of containing a top-k anomaly. Anomaly detection over a subspace tends to be very efficient since a subspace has typically a small number of attributes (dimensions). Fortunately, because anomalies are rare by definition, many of the 2^n subspaces are not correlated with anomalies. Figure 6.2 shows the general framework.

A natural question is then, "Which subspaces out of the 2^n should one examine?" SUITS chooses them based on the behaviors of the time series data (*i.e.*, t_i's). Roughly, abnormal time series in $\sigma_p(R)$ are separated into individual anomalies, and their correlated subspaces

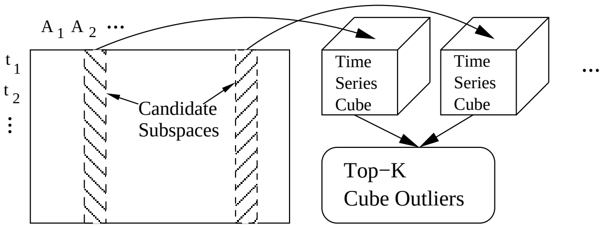


Figure 6.2: SUITS Framework

are chosen as **candidate subspaces**. These subspaces are then examined via exact cubing analysis. This approach avoids the curse of dimensionality in the original input data and turns it into a set of manageable sub-problems.

Additionally, during the computation of top-k's within a single subspace, the search space can be pruned if one detects that certain cuboids and their descendants does not have the potential to penetrate the top-k. This pruning method is developed in SUITS, and unpromising lattices in the data cube are avoided.

In summary, the algorithm proceeds iteratively as follows: (1) search for a group of anomalies, (2) find a subspace correlated with the group, and (3) compute the local top-k anomalies in the subspace data cube. The local top-k's of step (3) are merged together to form the global top-k. Though this merge is an approximation, we will show empirically that it usually matches the true top-k.

6.2.1 Retrieving $\sigma_p(R)$

Much like the naïve algorithm, the new algorithm also needs to first retrieve the set of data relevant to the query probe p, *i.e.*, $\sigma_p(R)$. Since there will be many different query probes posed to the same database, it is important to make this retrieval and its subsequent processing efficient. Thus, we perform preprocessing by pre-computing and storing C_R's shell-fragments [48] independent of the query and develop a shell fragment-based retrieval method.

A single *shell fragment* is a cuboid in C_R on a d-dimensional attribute group where d is a small number (*e.g.*, 1 to 3). For each cell in a fragment, the tid list of the associated tuples in R is recorded. For example, the shell fragment for the Gender dimension would contain two cells (*i.e.*, "Male" and "Female") and each would record essentially an inverted index on the tid's. A complete set of shell fragments (*i.e.*, where each dimension in R is represented in at least one shell fragment) is sufficient to compute any query on C_R. Shell fragments are efficient both in terms of speed and space.

Using these tid lists, retrieving $\sigma_p(R)$ at query time is simple. For each attribute-value restriction pair in p, we fetch its tid list from the most appropriate shell fragment. The intersection of all such tid lists is exactly $\sigma_p(R)$. This process is efficient no matter how many dimensions there are in R. Additionally, if p overlaps with some multi-dimensional shell fragments, efficiency will be vastly improved since those intersections are already pre-computed.

6.2.2 Selecting Candidate Subspaces

The idea of examining subspaces also exists in other problems. Subspace clustering [60] aims to find clusters in some of the 2^n subspaces. Principle component analysis and singular value decomposition also find more useful subspaces. In these problems, useful subspaces are discovered using signals such as density or class labels. In SUITS, the time series data are the signals. Intuitively, a significant anomaly at a cube cell should carry through to some of its descendants; for if all descendants are normal, their common ancestor would also be normal. Furthermore, descendants of common abnormal ancestors should also exhibit similar anomalies. Specifically, tuples in $\sigma_p(R)$ that share a common abnormal ancestor cell, which would also be a descendant cell of p, are likely to exhibit similar anomalies.

SUITS exploits this notion by *grouping* the tuples in $\sigma_p(R)$ based on their anomaly types and values. For a set of tuples in $\sigma_p(R)$ to be in the same group, they must have (1) the same anomaly type, (2) similar anomaly scores (*e.g.*, $\pm\delta$ range), and (3) same time span. Roughly, tuples within a group are base-level descendants of a single anomaly in a (possibly high level) descendant of the probe cell. At each iteration, the largest group is extracted and the most promising attribute values within it form the candidate subspace. This process starts with the construction of a **Time Anomaly Matrix**.

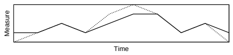
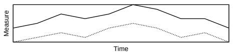
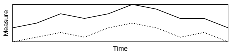
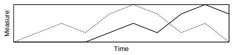
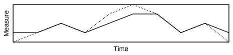
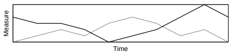
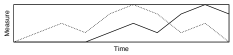
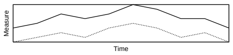
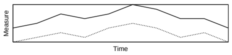
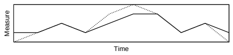
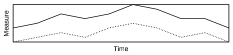
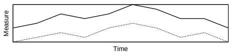
Education	Income	$S[1]$	$S[2]$	$S[3]$
Highschool	45k–60k	 None	 **Magnitude**	 **Magnitude**
College	35k–45k	 Phase	 None	 Misc
College	45k–60k	 Phase	 **Magnitude**	 **Magnitude**
Graduate	45k–60k	 None	 **Magnitude**	 **Magnitude**

Table 6.2: Time Anomaly Matrix

The Time Anomaly Matrix has size $T \times Q$, where T is the number of tuples in $\sigma_p(R)$ and Q is the number of sub-sequences SUITS will examine within S. Partitioning the time series satisfies the condition that anomalies will be found in sub-sequences. In business and many other applications, how to partition the s_i's is often natural, *e.g.*, every financial quarter.

Each entry (i, j) in the matrix corresponds to the jth time series sub-sequence of the ith tuple in $\sigma_p(R)$, and the value stored in the entry is the anomaly type and anomaly value of the corresponding sub-sequence. Formally, let $s[j]_{c_i}$ represent the jth sub-sequence of s_{c_i} where c_i is the ith cell or tuple in $\sigma_p(R)$. Then, the (i, j) entry in the matrix contains the output of $g(s[j]_{c_i}, \hat{s}[j]_{c_i})$.

Example 11 (Time Anomaly Matrix) *Using R from Table 6.1, let p be (Gender = "Female", Product = "Apparel"). For each tuple's s_i, we divide it into three pieces. Table 6.2* *shows the results. Under each piece is the anomaly type; we have omitted the score for simplicity.* ■

The Time Anomaly Matrix merely calculates anomalies in the base cuboid. Our overall goal is to find anomalies in possibly high-level cube cells. To enumerate all possible cube cells is cumbersome; instead we iteratively select potential subspaces. This process starts with the grouping of cells in the matrix. We use a simple method of hashing entries in the matrix into buckets by their anomaly type, anomaly score, and time. Each bucket will only hold one type, a small range of scores, and consecutive time spans. We then greedily select the largest such group by choosing the largest bucket. In Table 6.2, the six entries with Magnitude anomaly (in bold) form the largest group.

The next step is to find a useful subspace associated with this group. To exhaustively search for this is prohibitive. In many high-dimensional problems, greedy/heuristic methods are used to "bypass" the curse of dimensionality. For example, decision trees use an information-theoretic heuristic to greedily choose the decision nodes independently. CLIQUE [3] uses a coverage measure to select clusters in subspaces and greedily grow them to form bigger clusters. [2] uses an evolutionary algorithm to detect outliers in high-dimensional data. In SUITS, we take a similar approach by evaluating the attribute values individually in a statistical test to determine how well alone it correlates with the anomaly group. We term the score of this test the **Anomaly Likelihood** (AL) score. The few top-scoring values then form the correlated subspace. To measure the AL score of an attribute-value pair $(a_i = v_j)$, the purity of attribute a_i is calculated first via *entropy*.

$$Entropy(a_i) = -\sum_{v_j \in a_i} p(v_j) \log_2 p(v_j) \tag{6.2}$$

A "pure" attribute, that is an attribute whose values are homogeneous, would have low entropy; while an "impure" attribute whose values are uniformly distributed would have high entropy. If an attribute is pure, it is more likely to be correlated with the group than one that is impure. To give a trivial example, consider the Gender attribute for p in Example 11. It is 100% pure because all its values are "Female" and is trivially correlated with any anomaly. The equation below shows the AL score formula.

$$AL(a_i = v_j) = Frequency(a_i = v_j) \times Entropy(a_i)^{-1} \tag{6.3}$$

For each value v_j, it is also weighed by its frequency within the group. One can see that attribute values that occur very frequently and within a homogeneous attribute will have high AL scores.

Example 12 (AL Scores) *Table 6.3 shows results from the Magnitude anomaly group in Table 6.2. Within the Income attribute, the value "45k–60k" appears 3 times and no other value appears. The Income attribute is pure and thus scores an infinity for the AL score. Within the Education attribute, 3 different values appear uniformly, which maximizes entropy. In this case, Income = "45k–60k" is clearly correlated with the anomaly while Education is not. The AL score reflects this notion.*

Attribute Value	Frequency	AL Score
Income = 45k–60k	3	∞
Education = Highschool	1	1.58
Education = College	1	1.58
Education = Graduate	1	1.58

Table 6.3: Attribute value AL scores

In practice, Table 6.3 would be much bigger and the differences within it would not be as clear-cut. We select the top few scoring (5–7) attribute-value pairs to be **candidate attribute values**. The subspace formed by these values is the **candidate subspace**.

6.2.3 Discovering Top-K Anomaly Cells

The set of candidate attribute values describes a subspace within the original high-dimensional space. Its correlation to anomalies has only been suggested via simple entropy analysis. In this section, more exact cubing analysis is performed and the top-k cells in the subspace are found.

Let the set of candidate attribute values be B. A straight-forward solution is to materialize the data cube C_B, rank all cells by g, and return the top k. Note that the dimensionality of C_B is not necessarily equal to $|B|$. In Table 6.3 for example, even if all four attribute values are added to B, dimensionality is still just two. Second, to compute C_B, all tuples in $\sigma_p(R)$ are used, not just the candidate group. This ensures the detected top-k patterns apply globally. Though to ensure sub-sequences are searched, C_B only includes the time span of the found group.

This solution is definitely viable since C_B is relatively small. But we make two modifications in order to improve its efficiency. First, the number of times regression needs to be performed can be reduced through a property of the least-square error fitting. Second, branches in C_B can be pruned from the top-k search via an upper bound on g. The detail of these modifications can be found in [47]. Since they depend on properties of time series data, they will be skipped here.

6.2.4 Iterative Search

After discovering the local top-k cells in a subspace, they are merged into a global top-k. Entries from the original group are removed from the Time Anomaly Matrix. The whole process repeats until the Time Anomaly Matrix is empty. In the example of Table 6.2, the first iteration finds the rule: "Income = 45k–60k → Magnitude Anomaly : S[2–3]". In the next iteration, the entries with phase anomaly are selected and produce the following rule:

"Education = College → Phase Anomaly : S[1]". Algorithm 3 shows a high-level summary of SUITS.

Algorithm 3 SUITS

```
Input & Output: Same as Algorithm 2

1. Retrieve data for $\sigma_p(R)$
2. Repeat until global answer set contains global top-$k$
3.     $B \leftarrow$ candidate attribute values from $\{A_1, \ldots A_n\}$
4.     Retrieve top $k$ anomaly cells from $C_B$ using $g$ and $m$
5.     Add top $k$ cells to global answer set
6.     Remove discovered anomalies from input
7. Return top $k$ cells in global answer set
```

6.3 Experiments

To show the effectiveness of SUITS, we experimented with both synthetic and real world data. SUITS was implemented in C++ and compiled with GCC. All experiments were performed on a Linux machine with an Intel Core2 E6600 CPU and 2GB of memory.

6.3.1 Real World Data

We obtained real sales data from a Fortune 500 company. For confidentiality reasons, the name of the company, the names of products, or actual sales numbers cannot be revealed. The data include records from 1999 to 2005 and contains over 925,000 sales and nearly 600 dimensions. The measure in the cube is number of sales. g was computed for the entire time span and not sub-sequences.

In Table 6.4, we show efficiency results of many trend anomaly queries. For each query, we processed it in three different ways. First, we used the Naïve algorithm as described in Algorithm 2. Second, we used **SUITS**$_0$, which is SUITS without the top-k pruning. That is, it uses the iterative subspace search but local candidate cubes are fully materialized. And lastly, we used SUITS as described in the paper. Table 6.5 shows a similar experiment with magnitude anomaly queries except without SUITS_0. In both tables, $|R|$ shows the total number of dimensions in the data; we chose a relatively small set of low dimensionality because Naïve would often run out of memory with larger data sets of even medium dimensionality.

Probe	$\|R\|$	Naïve	SUITS_0		SUITS		Common
		Time	Time	% Imp.	Time	% Imp.	
Male, Single	10	14	5.9	58%	5.4	61%	9
Male, Married	10	299	95	68%	60	80%	10
Male, Divorced	10	3.6	2.8	22%	2.8	22%	10
Female, Single	10	15	8.2	46%	7.0	53%	9
Female, Married	10	114	31.0	73%	23.0	80%	8
Female, Divorced	10	5.5	3.8	31%	3.7	33%	10
Post-Boomer, Child=0	11	68.8	39.6	43%	32.1	53%	10
Post-Boomer, Child=1	11	16.8	5.4	68%	4.8	71%	10
Post-Boomer, Child=2	11	15.5	7.8	50%	6.7	57%	10
Boomer, Children=0	11	108.9	75.7	30%	52.4	52%	10
Boomer, Children=1	11	120.3	68.9	43%	58.0	52%	10
Boomer, Children=2	11	46.6	27.2	42%	23.6	49%	10
Average				48%		55%	9.6

Table 6.4: Run times of trend anomaly query with low dimensional data ($10 \leq |R| \leq 11$)

Probe	$\|R\|$	Naïve	SUITS		Common
Male, Single	10	13.4	8.2	38%	10
Male, Married	10	182.4	46.9	74%	10
Male, Divorced	10	4.1	3.1	24%	10
Female, Single	10	15.4	7.7	50%	10
Female, Married	10	85.5	17.4	80%	9
Female, Divorced	10	6.5	4.1	37%	10
High School	11	92.5	22.9	75%	10
College	11	382.4	35.5	91%	10
Post-Graduate	11	110.7	34.9	68%	10
Average				60%	9.9

Table 6.5: Magnitude anomaly query run times

In both tables, we notice that SUITS is, on average, over 50% faster than the Naïve algorithm. This is especially true for the large queries, either by a large number of dimensions or large number of tuples. This is not surprising because SUITS breaks a very large problem into more manageable parts. Figure 6.3 shows a closer look at a single query as the number of dimensions increase from 7 to 14. At 7, the Naïve algorithm is faster than SUITS, because the data cube is relatively small and SUITS has additional overhead. However, as dimensionality increases, the trends of Naïve and SUITS are very different. Naïve shows the expected curse of dimensionality; in fact, with $|R| = 12$, Naïve ran out of memory for full cube materialization. With SUITS, we observe a more or less linear or even sub-linear behavior. This is because SUITS is more dictated by the anomalies inside the data rather than the external size of the data.

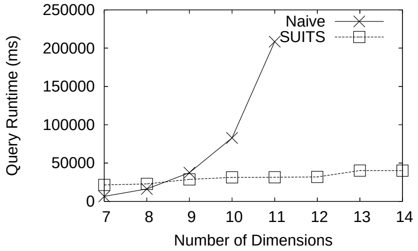


Figure 6.3: Running time vs. number of dimensions

As mentioned previously, the top-k produced by SUITS is not guaranteed to be the same as the true top-k. This could occur if particular attributes or combinations of attributes are not examined within a single iteration of SUITS. In practice, we noticed that this sometimes happens with dimensions of high cardinality (*e.g.*, zip code, state). The reason is that high-cardinality dimensions often have high entropy just by definition and thus low AL scores. And so they sometimes are not picked as candidates. An easy way to fix this would be normalize entropy based on the cardinality of the dimension. However, this scenario is usually the exception rather than the rule. The last columns of Tables 6.4 and 6.5 show the number of items in the top-10 that is common between the SUITS top-10 and the true top-10. As they show, SUITS usually produces the same top-10 as the true top-10.

6.3.2 Synthetic Data

To test SUITS in a more controlled environment, we also generated our own data. Each data set consisted of 95% normal, background "noise" and 5% abnormal patterns. For the normal portion, each value under each dimension was picked uniformly and independently between 1 and 5. Each value inside the count and time series measure was also picked uniformly and independently between 0 and 5. For the abnormal portion, 10 abnormal patterns were generated. Each pattern consists of 1–3 dimensions and a randomly chosen count/measure pattern. These 10 patterns are randomly inserted into the data 5% of the time. For all queries in the rest of this section, p was set to a random 2-dimension query. Each experiment was repeated 10 times to get an average.

Figure 6.4 shows running times in a 10-D data set as the number of tuples increases with everything else fixed. Both SUITS and Naïve exhibit linear complexity; even though SUITS is iterative. This is expected because the size of the Time Anomaly Matrix grows linearly with data size and so does the number of iterations needed to cover it.

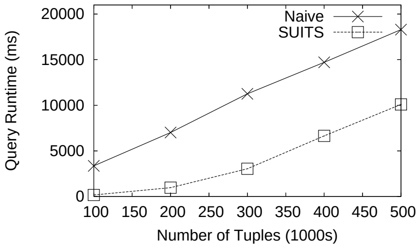


Figure 6.4: Running time vs. number of tuples

Curse of dimensionality is a well-known problem in data cubing and other multi-dimensional analysis. In SUITS, because bulk of the analysis is spent on subspaces, the overall dimensionality should not affect running time too much. Figure 6.5 shows running time as dimensionality increases from 6 to 16 but everything else remained the same. The number of tuples was 250,000. With the Naïve method, the running time quickly runs out of control. With SUITS however, we observe a relatively linear or flat curve. The flat portion is explained by the sparsity of the high dimensional data. In these cases, SUITS may only run a couple of iterations because anomalies are so easy to find.

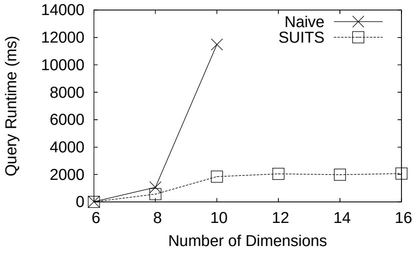


Figure 6.5: Running time vs. number of dimensions

One of the reasons we chose regression as the method of representation for time series data was that it allows aggregation from intermediate results as opposed to from scratch. Figure 6.6 corroborates this assertion. The data had 8 dimensions and 250,000 tuples. As the length of the time series increases from 20 to 100 with everything else fixed, one observes a linear increase for the Naïve method. This is expected because there are just more numbers to aggregate. For SUITS, the increase is much flatter since the length of time series does not affect aggregation of regression parameters. The minor increase is probably due to the increase in time to compute the least-square fit.

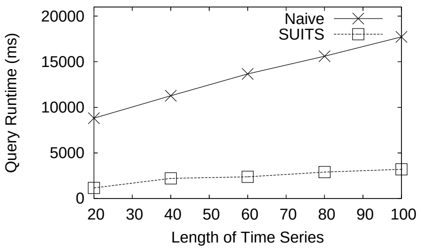


Figure 6.6: Running time vs. length of time series

Chapter 7

Temporal Traffic Outliers

One very practical and important problem in vehicle traffic analysis is outlier detection. A typical definition of an outlier is "*an observation (or a set of observations) which appears to be inconsistent with the remainder of that set of data* [5]." This rather vague definition can lead to many different outlier detection algorithms. This particular work focuses on the detection of *temporal outliers*. Previous studies have addressed temporal data [35, 84, 45]. But time has usually been represented as a set of dimensions and the outlier is measured for the entire timespan. In one of our recent studies [53], outliers are measured with respect to (1) other points in the dataset, (2) temporal history of itself, and (3) temporal history of other points in the dataset.

This work will primarily use the application scenario of *detecting temporal outliers in vehicle traffic data*, though other domains such as social networks or sensor networks are applicable as well. More specifically, it seeks to detect outlier behavior in the set of road segments of the traffic data and not individual moving objects. For example, consider Figure 7.1, which shows the load (count) of vehicles on road segment X over the course of several days.

From this figure alone, there does not seem to be any abnormal behavior. The average speed roughly follows a periodic function with one day being the period. But suppose additional information about other road segments in the city are given. Figure 7.2 shows the average speed of several other road segments in the city.

Road segment X remains the same in Figure 7.2. But in light dashed lines are many

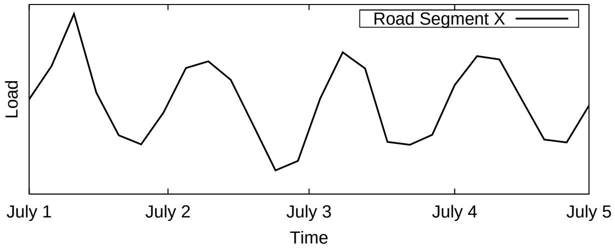


Figure 7.1: Historical Speed on Road Segment X

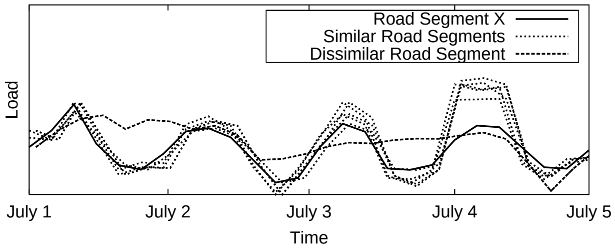


Figure 7.2: Historical Speed on Many Road Segments

other road segments that have similar loads as road segment X from July 1st to July 4th. On July 4th, however, they all show an increase in load while X remains the same. The question of outliers has now become much trickier. If there are thousands of these *historically similar* road segments and they *all* show the same increase on July 4th (a national holiday in the United States) and yet X remains low, X has now just become an outlier. Recall the outlier definition regarding "inconsistency." This example and the rest of this work use historically similar neighbors as the basis for consistency comparisons. In Figure 7.2, the darker dashed line does not show historical similarity to X so it is not considered in X's analysis.

This work presents the **Temporal Outlier Discovery** or TOD framework for detecting *temporal outliers*. In contrast to other algorithms, this method utilizes agglomerated temporal information of the entire dataset to detect outliers. At each time step, each road segment checks its similarity (based on load, speed, or any other measure) vs. other road segments in the traffic data, and the historical similarity values are recorded in a *temporal neighborhood vector* at each road segment. Outliers are calculated from drastic changes in these vectors.

By using the historical similarities between road segments, the method is robust to global changes such as weekend or holiday traffic in the measure itself. Further, instead of using a clustering method to record whether two road segments belong to the same cluster over time, the temporal neighborhood vectors allow temporal flexibility since traffic is very dynamic and a binary answer is often impossible to reach. Likewise, the final outliers are not binary either. An *outlier score* is given to each road segment in the dataset where a higher score indicates a larger likelihood of a road segment being an outlier.

The rest of the chapter is organized as follows. Section 7.1 discusses the representation of road segments. Section 7.2 formally defines outliers and the detection algorithm. Section 7.3 describes the overall TOD framework. Experiments are shown in Section 7.4.

7.1 Road Segment Representation

The first step in the TOD framework is to represent the vehicle traffic data in a formal feature space. This section discusses the form of the raw input data and feature vector construction of road segments.

7.1.1 Input Data

At the raw level, input data contains timestamped locations in a road network. This can be gathered via GPS sensors in the moving vehicle or external sensors installed throughout the city (*e.g.*, RFID). Because the vehicles are physically restricted in the road network, this naturally leads to a graph representation of the road network. More formally, let the road network be represented with a planar graph $G = (V, E)$ where the set of vertices V represent street intersections and the set of edges E represent road segments. Then, the most basic input data is a set of timestamp and edge tuples: $\{(t, i), \ldots\}$ where t is some time value and i is some edge in E. Note that the IDs of the vehicles are not needed since the outlier analysis is focused on behaviors of road segments (*i.e.*, E). In many real world scenarios, the IDs are unavailable anyway due to privacy concerns. Also, because the analysis operates on the edge level, the exact position along edge i is ignored. Only the binary edge traversal is recorded. Furthermore, depending on the analysis goal, other attributes can be added to the basic tuple of (t, i). For instance, if speed and weather are of interest, each tuple could take the form of (t, i, s, w) where s is the speed and w is the weather.

7.1.2 Feature Space

In order to formally compare edges in terms of temporal behavior, every edge in E is mapped to a point in some feature space. The TOD framework allows for any number of feature dimensions. Depending on the application, the features can be spatial or not. For instance, if the goal were simple speed outlier detection, then the spatial location of the road segments could be excluded. Let the set of used features be $\mathcal{F}$, then every road segment is mapped to a point in $R^{|\mathcal{F}|}$.

Table 7.1 shows an example of four road segments with two features values indicating the average speed on them during the morning and afternoon hours of a day. This coarse granularity point-of-view is probably going to be ineffective in the real world, but it is used here to conserve space. Because the data changes temporally and temporal outlier is the topic at hand, these two feature values are updated every day (or any other time period) using the traffic data. That is, every input tuple as described in the previous section is processed to update the appropriate feature(s) in the corresponding road segment. Table 7.1 shows 3 consecutive days of feature values.

With the feature space constructed and the road segments mapped to points, one can now formally discuss a similarity measure or distance metric between edges. Some examples include the Euclidean distance, cosine similarity, Manhattan distance, and L-∞ distance. The specifics of the metric depend heavily on the final application. More will be discussed

	Day 1		Day 2		Day 3	
	AM Speed	PM Speed	AM	PM	AM	PM
1	3	20	3	19	10	20
2	5	21	3	21	4	22
3	2	20	3	23	3	21
4	15	32	4	23	20	20

Table 7.1: Sample Feature Space

later about how these distances will be used.

7.2 Temporal Outlier Detection

Outlier analysis in TOD depends heavily on the time dimension. A prerequisite for this is the existence of some temporal patterns in the data. The next few paragraphs will show that this is the case in real world data both in terms of the measure (*e.g.*, speed) and neighborhood relationships. Figure 7.3 shows the average percent deviation in average daily speed and average daily load (*i.e.*, count) of taxicabs on over 20,000 road segments in the San Francisco area during the month of July 2006; the road segments are chosen for having at least 50 vehicle traversals per day. The average speed and average load on each road segment are calculated every day, and the graph shows the average percent change from the previous day for all road segments. There are over 33 million recorded vehicle road segment movements in the data set.

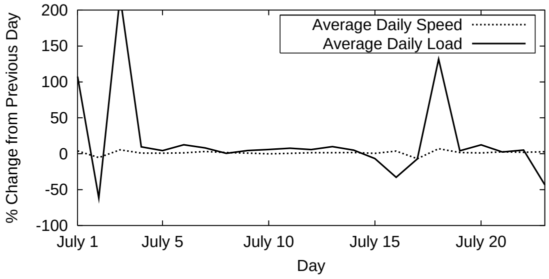


Figure 7.3: Stability of Average Daily Speed and Load

As the figure shows, the average daily speed is very stable (modulo the spikes for now) day-to-day. Though not particularly surprising, this is good news for temporal outlier detection. It shows that there are stable trends and they can be used as a basis for outlier detection.

There are several big spikes in the average load deviation in Figure 7.3. They occur around July 3rd and July 18th. While one may attribute the first spike due to July 4th, a national holiday in the United States, a closer look into the data reveals the true cause. Due to either sensor error or data collection error, no traffic was collected for a 12-hour period on July 2nd, an 18-hour period starting on the night of the 16th until noon on the 17th, and a 12-hour period on the 24th. If the outlier detection system used simple historical trends based solely on the measure itself, these spikes would generate many outliers. But they would all be erroneous because the change was global. On a lesser scale but not due to data error, weekend/weekday/holiday shifts will also produce erroneous outliers.

The TOD framework uses a different method of outlier detection. Instead of evaluating the trends of the measure(s) directly and *singularly*, trends of similar neighbors based on the measure(s) are evaluated. Figure 7.4 shows a confirmation of this observation from real world data. In it, neighbor road segments according to speed (± 5MPH) and load (± 50 cars) of each road segment is calculated on every day. Then, the average percent change in the size of a continual intersection of this neighbor set of all road segments is plotted against time. The plots are erratic at the beginning because stable neighborhoods have not been established yet. But as time goes on, the neighborhoods become very stable with an average deviation of less than 5%. More interestingly, around July 16th and July 17th where data loss caused huge spikes in Figure 7.3, Figure 7.4 barely shows any effect. This is because the data loss was global and entire neighborhoods shifted together. As a result, locality of road segments in the feature space is preserved.

The outlier detection scheme in TOD is motivated by the stable neighborhoods. Given a

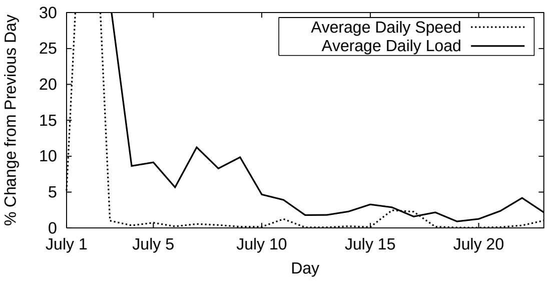


Figure 7.4: Speed and Load Neighborhood Stability

road segment with a historically stable set of neighbors (in feature space, not physical space), an outlier is loosely defined as a drastic change in the membership of this set. The power of this method vs. a method that measures only the singular road segment is that it is robust to population shifts. For example, on a weekend, the behavior of many road segments are likely to change. But because these changes are likely to be similar, the uniform *global* shift will not affect the set of stable neighbors.

7.2.1 Temporal Neighborhood Vector

A natural method to record temporal neighborhoods is clustering. One could cluster road segments at each time step and link them temporally. The problem with this approach is that it is very rigid. It is difficult to say definitively whether two edges belong to the same cluster over a long period of time. Not only would that make the final answer extremely sensitive to the clustering method and associated parameters, it would also not allow any latitude in slight temporal changes. In contrast, TOD uses a more local and flexible method to maintain stable neighborhoods: Temporal Neighborhood Vectors.

Definition 10 (Temporal Neighborhood Vector) *Every edge i maintains a vector $\vec{v_i}$ to record historical similarities to other edges. The length of this vector is $N = |E|$. Every j^{th} value, $v_{i,j}$, in the vector records the historical similarity between edge i and edge j. A large $v_{i,j}$ indicates high historical similarity and vice-versa. Initially, all vector values are set to*

0.

The temporal neighborhood vectors indicate the "probability", in a non-technical sense, that edges i and j are similar to each other over time. The usage of a real value to indicate historical similarity is much more flexible than a binary flag in clustering. The *degree* of similarity over time can be easily expressed.

At every time step, each value $v_{i,j}$ is adjusted up or down depending on the similarity between edge i and j at that instant. TOD uses a threshold to determine similarity.

Definition 11 (Instantaneous Similarity) *Let $d(i,j)$ be a distance function defined on the feature space. Then, edges i and j are similar with respect to an instant in time and the particular feature values at that instant (i.e., instantaneously similar) if $d(i,j) \leq \theta$.*

The temporal neighborhood vector values are updated differently based on whether instantaneous similarity holds true or false. As a simple example, consider the Day 1 data from Table 7.1. Let $d(i,j)$ be the L-∞ distance (*i.e.*, maximum absolute difference between feature values). Table 7.2 shows the L-∞ distance between pairwise edges on Day 1.

Edge Pairs	L-∞ Distance
L-∞(1, 2)	2
L-∞(1, 3)	1
L-∞(1, 4)	12
L-∞(2, 3)	3
L-∞(2, 4)	12
L-∞(3, 4)	13

Table 7.2: Edge Similarity Values for Day 1

Let $\theta = 5$. In this case, edges 1, 2, and 3 are instantaneously similar to each other while edge 4 is not to any other edge. Assume, for now, that each $v_{i,j}$ gets incremented by 1 if edges i and j are similar and decremented by 1 otherwise (with a minimum of 0). The temporal neighborhood vector of edge 1 is then $\langle 1, 1, 1, 0 \rangle$. The first value $v_{1,1}$ indicates edge

1 is similar to itself, which is always true. The second value $v_{1,2}$ indicates edge 1 is similar to edge 2 and so forth.

7.2.2 Temporal Vector Update Rules

With instantaneous similarity defined, the next step is then updating the neighborhood vectors temporally, where time is essentially a sequence of instants. As mentioned briefly before, the amount of change in $\vec{v_i}$ at each time step is the measure of anomaly or "outlier-ness" for edge i. Thus, it is critically important to have proper update rules. Before presenting the exact formulas, which are formally simple, some motivation is needed to justify them.

Recall that a large $v_{i,j}$ value indicates high historical similarity between edges i and j. This naturally leads to the conclusion that if edges i and j are instantaneously similar, then $v_{i,j}$ should increase. Likewise, if edges i and j are instantaneously dissimilar, then $v_{i,j}$ should decrease. A natural inclination might be to simply add 1 to $v_{i,j}$ every time $d(i,j) \leq \theta$ and subtract 1 otherwise. The total absolute change in $\vec{v_i}$ is the outlier score for edge i at each instant.

This system is simple but flawed. The amount of reward or penalty for two edges being instantaneously similar or dissimilar is the same regardless of their previous history. This seems unfair since a previously strong or weak historical relationship should have a different impact on the temporal neighborhood vector and the outlier score. TOD uses the following intuitions based on the existing similarity values.

1. If two edges are historically similar, then a new instantaneous similarity can be noted lightly.

2. If two edges are historically similar, then a new instantaneous dissimilarity should be noted heavily.

3. If two edges are not historically similar, then a new instantaneous similarity should be noted heavily.

4. If two edges are not historically similar, then a new instantaneous dissimilarity can be noted lightly.

Compared to the simple fixed increment/decrement system, this set of intuitions provides a *varying degree of reward or penalty based on the existing similarity value.* For example, if two historically similar neighbors all-of-a-sudden break their similarity, the penalty is large and in turn causes a large outlier score. Figure 7.5 shows a graphical representation that summarizes the four intuitions listed above. In the figure, the x-axis shows the existing similarity value, and the y-axis shows the amount of penalty or reward in a generic sense. Exact values notwithstanding, the two curves portray the intuitions accurately. In the reward curve, at low levels of similarity (left side), the reward is larger than at high levels of similarity (right side). The opposite is true in the penalty curve.

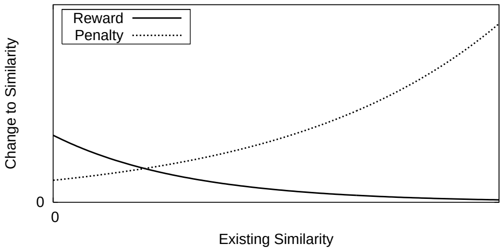


Figure 7.5: Similarity-based Reward/Penalty to $v_{i,j}$

The two curves in Figure 7.5 are easily identifiable as exponential functions. This turns out to be the case in TOD. Let the period of update be daily and consider the similarity value $v_{i,j}^{d-1}$ between edges i and j on day $d-1$. If these two edges are instantaneously similar on the next day d, the reward is defined as

$$reward(i,j,d) = \alpha_1^{v_{i,j}^{d-1} - \alpha_2} \qquad \alpha_1 < 1.0,\ \alpha_2 \geq 0 \tag{7.1}$$

For progressively larger $v_{i,j}^{d-1}$ values, the reward naturally becomes smaller. The update function for $v_{i,j}^{d}$ is then

$$v_{i,j}^{d} = v_{i,j}^{d-1} + reward(i, j, d) \tag{7.2}$$

The same equation applies to penalties as well. If edges i and j are instantaneously dissimilar on day d, the penalty is defined as

$$penalty(i, j, d) = \beta^{v_{i,j}^{d-1}} \qquad \beta > 1.0 \tag{7.3}$$

For progressively larger $v_{i,j}^{d-1}$ values, the penalty becomes larger exponentially. The update function for $v_{i,j}^{d}$ is then

$$v_{i,j}^{d} = v_{i,j}^{d-1} - penalty(i, j, d) \tag{7.4}$$

The question then becomes how to set the α_1, α_2 and β values. A small α_1 quickly reduces the reward as existing similarity increases, and a large β quickly increases the penalty as existing similarity increases. Section 5.2 will discuss them in more detail.

Continuing the running example, the updated temporal vectors from Day 1 until Day 3 from Table 7.1 are shown in Table 7.3. α_1 is set to 0.9, α_2 is set to 0, β is set to 1.1, θ is set to 5, and $d(i, j)$ is set to L-∞. There is a minimum of 0 to any $v_{i,j}$ value.

Edge	Temporal Neighborhood Vectors		
	Day 1	Day 2	Day 3
1	$\langle 1, 1, 1, 0\rangle$	$\langle 1.9, 1.9, 1.9, 1\rangle$	$\langle 2.7, 0.7, 0.7, 0\rangle$
2	$\langle 1, 1, 1, 0\rangle$	$\langle 1.9, 1.9, 1.9, 1\rangle$	$\langle 0.7, 2.7, 2.7, 0\rangle$
3	$\langle 1, 1, 1, 0\rangle$	$\langle 1.9, 1.9, 1.9, 1\rangle$	$\langle 0.7, 2.7, 2.7, 0\rangle$
4	$\langle 0, 0, 0, 1\rangle$	$\langle 1, 1, 1, 1.9\rangle$	$\langle 0, 0, 0, 2.7\rangle$

Table 7.3: Day 1–3 Temporal Neighborhood Vectors

7.2.3 Temporal Outlier Scoring

Outliers in TOD are measured from *drastic* changes in the temporal neighborhood vectors. Intuitively, the most drastic or abnormal changes are ones that differ the most from historical values. Furthermore, the more stable the historical values are previously, the more the change should contribute to the overall outlier score. Fortunately, these intuitions are easily captured in the reward and penalty equations in the previous section. Because $v_{i,j}^{d-1}$ is in the exponent of Equations (7.1) and (7.3), the big rewards and penalties will come from previously stable trends (either similar or dissimilar). The outlier score of edge i on a particular day is then equal to the sum of rewards and penalties. Let $v_{i,j}^{d}$ represent the value of $v_{i,j}$ on day d. Then, the *outlier score* of edge i on day d, $OS(i,d)$, is defined as

$$OS(i,d) = \sum_{j=1, j \neq i}^{N} \left| v_{i,j}^{d} - v_{i,j}^{d-1} \right| \tag{7.5}$$

With N fixed, large changes in $v_{i,j}$'s are necessary in order to produce a large outlier score. Consider the various types and semantics of changes from $v_{i,j}^{d-1}$ to $v_{i,j}^{d}$. There are two situations when $v_{i,j}^{d-1}$ increases. First is when a historically similar neighbor (*i.e.*, large $v_{i,j}^{d-1}$) is instantaneously similar yet again. This is not particularly surprising and Equation (7.1) treats it lightly. Second is when a historically dissimilar neighbor (*i.e.*, small $v_{i,j}^{d-1}$) becomes similar. This is somewhat surprising and Equation (7.1) gives a large reward accordingly. Conversely, there are also two situations where $v_{s,j}^{d-1}$ decreases. First is when a historically similar neighbor becomes dissimilar. This is very surprising and Equation (7.3) gives a large penalty. Second is when a historically dissimilar neighbor remains dissimilar. This is expected and Equation (7.3) gives a small value accordingly.

Continuing the previous examples, consider the outlier scores of Table 7.3. $OS(1,2) = |1.9 - 1| + |1.9 - 1| + |1.0 - 0| = 2.8$. Table 7.4 shows the outlier scores of edges on all days. On Day 2, edge 4 has the highest outlier score. Intuitively, this is sensible because it became similar to the other edges while it was not on the previous day.

Edge	Outlier Scores		
	Day 1	Day 2	Day 3
1	2.0	2.8	3.4
2	2.0	2.8	3.0
3	2.0	2.8	3.0
4	0.0	3.0	3.0

Table 7.4: Outlier Scores

7.3 The TOD Framework

The discussion so far has been limited to only road segments as the target of outlier detection. However, analysis can also be on higher level concepts such as sequences of segments or entire moving object trajectories. Figure 7.6 shows the overall flow of data in TOD. Clear boxes show data and shaded boxes show computation modules. The framework is general in the sense that it can operate on any type of spatiotemporal data, feature space, and distance function.

For example, suppose one created "virtual edges" between the starting and ending locations of all trajectories. Semantically, these edges would represent the general flow of people in a city. There are, of course, temporal patterns in such data. Consider the number of people who drive to work in downtown from the various suburbs everyday. These flows are very stable day-to-day because people tend to goto work during the same time everyday and use the same routes. A slight outlier in such data is indicative of some anomaly in the city traffic.

On a lesser scale, one could also analyze sub-trajectories. Instead of single road segments or whole trajectories, sequences of road segments could be analyzed for anomalies in popular sub-routes. The key point is that no matter at which level of abstraction the original

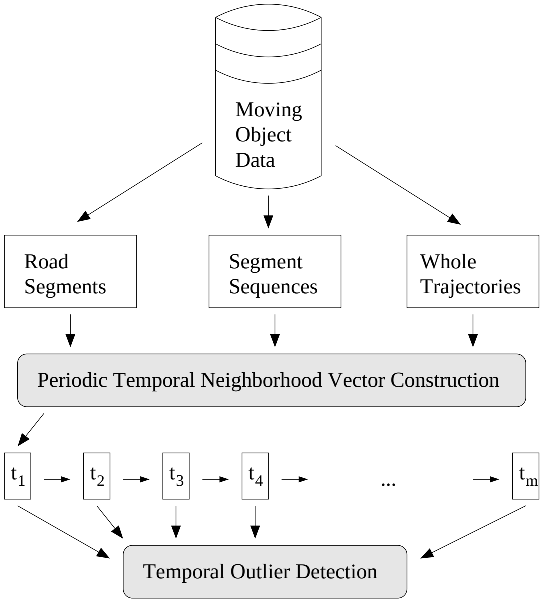


Figure 7.6: TOD Data Flow

input data is processed, the overall TOD framework remains the same. The output will automatically match the abstraction level of the input.

7.3.1 Complexity

The runtime of TOD is evenly divided per time step; thus complexity with respect to time is linear. At each time step, two processing steps are needed. First is to generate the feature values for each point (*i.e.*, road segment, path, etc.) in the dataset. This entails processing all the traffic data once and is a linear time operation. Second is to compute the pairwise similarities between all points in order to update the temporal neighborhood vectors. If there are N points, the runtime cost at each time step is then $O(N^2)$.

7.3.2 Setting Parameters

There are four parameters to set in TOD. First is θ, which dictates the threshold of instantaneous similarity. For many features, this is easy to set. For example, if the feature is speed and the similarity measure is L-∞, ± 5MPH could be reasonable choice. Because θ is directly related to the real world measure, it is quite intuitive to set. Further, the θ parameter could actually be eliminated if it is incorporated into Equations (7.1) and (7.3) directly. That is, the reward and penalty equations could operate on the distance between edges directly. This seems attractive because it eliminates a parameter, but it can be very tricky to alter the equations. Because θ is so intuitive to set directly, the choice was made to retain it.

The other three parameters are related to the temporal neighborhood vector updating rules, specifically α_1, α_2 and β. A small α_1 decreases reward quickly as existing similarity grows. Semantically, this reduces the impact of history because a long period of instantaneous similarity does not equal a big reward as time goes on. A large β increases penalty quickly as existing similarity grows. This increases the impact of history because a long period of instantaneous similarity equals a large penalty. It also serves the following function. Suppose two edges have been historically similar for a long time. Then, due to some structural change, the similarity is broken permanently. If the penalty for this is small, it would take a long time to "undo" the existing similarity. This would cause the system to unnecessarily report the outlier for a long time. With a reasonable β, however, it would be quickly erased. The key is to set β such that history is erased gracefully and not too quickly. Figure 7.7 shows a sample timeline of similarity with $\alpha_1 = 0.95$, $\alpha_2 = 3$, and $\beta = 1.05$. In the figure, it takes roughly 10 time steps to undo 40 time steps of continuous instantaneous similarity. In experiments, this was found to be reasonable, but the exact settings depend on the application and also the importance of history.

7.3.3 Other Applications

One possible extension of the TOD framework is application to possibly non-spatiotemporal data. At its core, temporal relationships between objects form the basis for outlier detection.

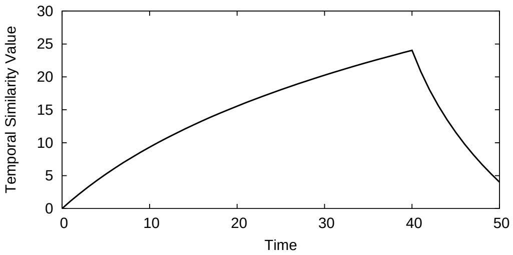


Figure 7.7: Temporal Similarity

Social networks that evolve temporally is a possible application for this data. Instead of measuring the relevant measures on the actors, temporal relationships between the actors can be studied to discover outliers that suddenly change their ties. Sensor networks is another good application. In many cases, events will cause entire groups of sensor readings to shift. This maybe understood as normal because the event is expected. Thus, it maybe more appropriate to detect temporal outlier behavior within groups of sensors.

Another attractive feature of the TOD framework is its extensibility to streaming data. Every road segment maintains its own temporal neighborhood vector, and they can be updated on-the-fly with new incoming data. Because the update process is relatively efficient at $O(N^2)$ per time step, running the outlier detection algorithm periodically is reasonable. Furthermore, there is a slight optimization possible if only the top-k outliers are seeked at each time step. Because the update formulas for reward and penalty are known a priori, it is possible to bound them without looking at the data. Then, if only the top k outliers are seeked at a new time step, the bounds can be used to reduce the number of computations. However, these bounds get looser over time if no update takes place. At some point in time, they will become too big to require an update, which entails processing all the data again if the true answer is desired. As a result, the amortized runtime at each time period will still approach the original.

7.4 Experiments

Real world moving object data was used to test the effectiveness of TOD. 24 days of moving taxicab data in the San Francisco area were collected during the month of July in 2006. In all, there were over 800,000 separate trips, 33 million road segment traversals, and 100,000 distinct road segments. Data was recorded at the second level for time and also includes the speed on each road segment. In all experiments, α_1 is set to 1.1, α_2 is set to 0, and β is set to 0.95.

7.4.1 Outliers

To test the effectiveness of TOD, a variety of feature spaces and parameter settings are used to detect outliers in the real world data. A few examples are shown here. Figures 7.8 and 7.9 show two outliers with respect to speed. The feature space includes eight features, one for every 3-hour span of the day (*i.e.*, 12am–3am, 3am–6am, etc.). Figure 7.8 shows the average speed of vehicles on one segment of 14th St. in San Francisco during 6am–9am over the course of several days. Also shown is the average speed of vehicles on neighboring road segments, which are road segments in the temporal neighborhood vector of 14th St. with the top 25% largest absolute changes (counting both reward and penalty) on the day of the outlier. In Figure 7.8, 14th St. has 876 such neighbors. Their average is very stable in time: 16MPH constantly and has very small deviations on each day: less than 0.2. This stability is not surprising due to the large number of road segments. On July 11th, the average speed on 14th St. rises significantly compared to its similar neighbors and is caught as an outlier. Figure 7.9 shows another similar example with a different road segment.

Figure 7.10 shows an outlier with respect to load (*i.e.*, count) of vehicles. The neighboring segments have a stable trend (427 neighbors with daily standard deviation under 1.0) from July 3rd to July 13th while the outlier road segment's load increases significantly on July

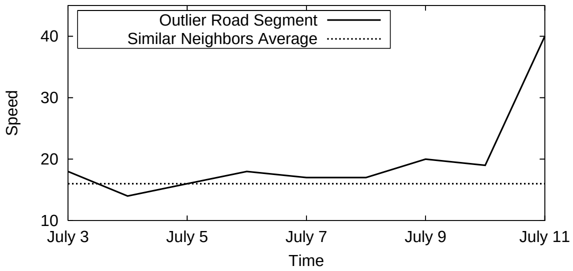


Figure 7.8: Speed Outlier (14th St.)

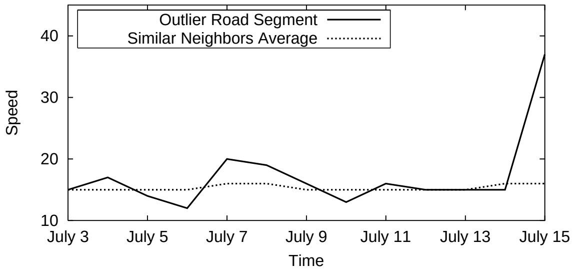


Figure 7.9: Speed Outlier (Webster Ave.)

9th and 13th.

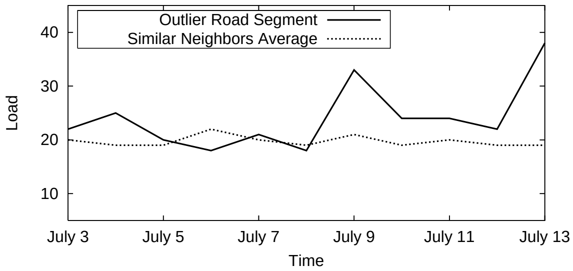


Figure 7.10: Load Outlier

Figure 7.11 shows another outlier with respect to load. In this case, the cause is opposite than that of Figure 7.10. The load of the neighbors (107 of them) increases on July 6th and 7th (weekend) while the road segment in question remains stable. This is rather strange because the road segment is very similar to its neighbors on weekdays. And yet, it did not receive the weekend bump in load due to some unknown reason.

The basis for consistency comparisons in TOD are historical similarities between road

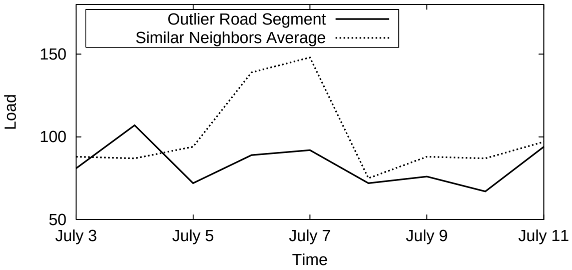


Figure 7.11: Load Outlier

segments. In contrast to using direct historical trends in the measure itself, this is more powerful since sometimes trends may not exist in the measure but do in the similarities. Figure 7.12 shows a non-outlier in TOD. On July 7th and 8th, the load increases significantly when compared to previous data. A naïve method is likely to report these outliers. But when compared to its historically similar neighbors, the increase is global and rather normal. Those two days are actually weekend days (Friday and Saturday) and the universal increase in load is reasonable.

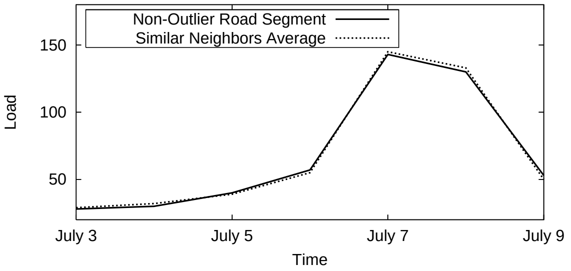


Figure 7.12: Non-Outlier

7.4.2 Naïve Outlier Detection

One of the advantages of TOD is its robustness to global changes, which could fool outlier detection systems that only rely on the historical data of the single road segment. In this section, such a system is implemented and tested on the real world data. For every road segment, a running average of daily average speed is maintained. Then, on each day, the absolute difference between that day's average speed and the running average is the outlier score.

Figure 7.13 shows one particular outlier found through this algorithm. The first graph shows the running average of the road segment in question as a function of time. On each day, the running average is the average of all previous day's average speeds excluding the current day. The second graph shows the average speed on every day, ignoring all previous history. The naïve algorithm simply compares the current day's average vs. the running average. In this case, it is obvious from this comparison that the speed on July 7th and 8th deviate significantly from the running average, and thus an outlier is reported. But, this turns out to be incorrect when historically similar neighbors are considered. The third graph in Figure 7.13 shows the average daily speeds of historically similar neighbors (within 5MPH on every day before July 8th). From this graph, it can be seen that the entire set of road segments has an increase in speed on July 7th and 8th. A closer examination reveals these are actually weekend days. Thus, the more plausible explanation for the rise is that there are less cars on certain roads (interstate highway I-880 in this example) on the weekends and speed naturally rises on all of them. As a result, this particular road segment is not an outlier. This conclusion would not have been possible had similar neighbors not been included in the analysis.

Figure 7.14 shows another example of a false outlier reported by the naïve method. Again, there is an increase in speed (July 14th, 2006, another weekend) when compared to the running average of the road segment itself. However, when the neighbors are taken into consideration, it seems rather normal.

7.4.3 Efficiency

Finally, the efficiency of `TOD` is tested. Figure 7.15 shows the running times of `TOD` as a function of the number of days processed. There was a total of 33 million road segment

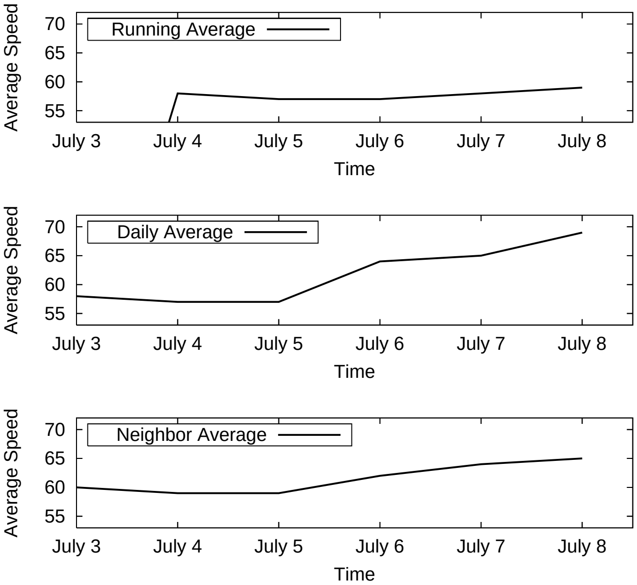


Figure 7.13: Average Speeds of I-880

traversals spread throughout 24 days, which makes an average of roughly 1.4 million road segment traversals per day. As the figure shows, processing these traversals and updating the temporal neighborhood vectors is a linear time operation with respect to the number of days. There are two curves shown for two different settings of the parameter θ, the threshold for instantaneous similarity. In both curves, the features on the road segments represent speed and θ sets the maximum difference in the L-∞ distance. As expected, when θ is equal to 10, running time increases because there are more neighbors to process at each time step.

Figure 7.16 shows a more extensive experiment comparing running time vs. θ. Features related to speed are used in the experiment, and θ is the maximum miles-per-hour difference in the L-∞ distance. All 24 days of traffic was processed. The same trend in Figure 7.15 presents itself. As θ increases, the number of instantaneously similar neighbors to be processed at each time step increases and thus causes a longer running time.

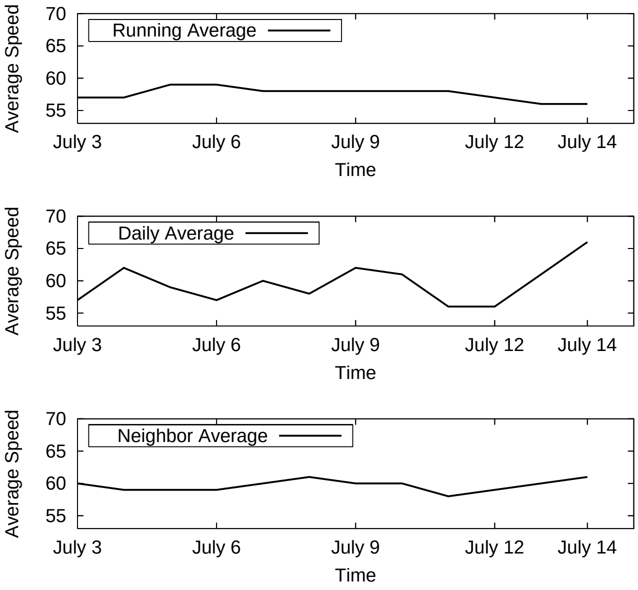


Figure 7.14: Average Speeds of San Jose Ave.

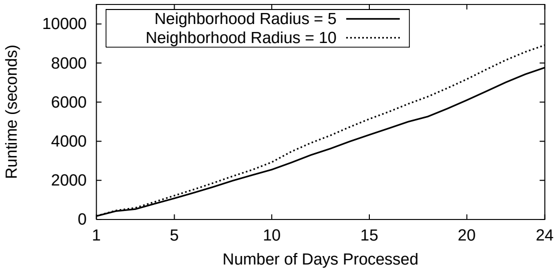


Figure 7.15: Efficiency vs. Number of Days

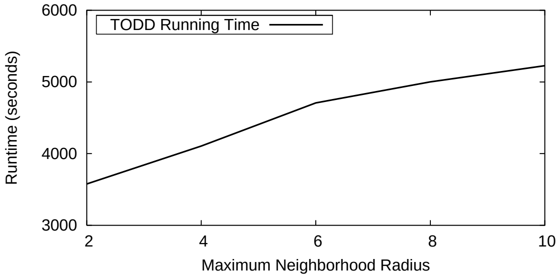


Figure 7.16: Efficiency vs. Neighborhood Radius

Chapter 8

Conclusion

In this thesis, we have explored the possibility of data mining in spatiotemporal data, specifically moving objects. We reviewed current work in the area of indexing, query processing, clustering, data mining and more. Though there has been a good amount of work in this area, some in depth, there is still a dearth of attention on high-level problems. Most work focus on traditional spatiotemporal data, not moving objects, where trajectories are formed. More importantly, most of the efforts have been put into fundamental problems such as indexing or query processing. Such problems are very important and their solutions form very important building blocks to higher level problems. At this point in research, the field of moving objects research has accumulated enough robust building blocks such that one can finally look towards higher level problems and solutions.

Figure 8.1 shows again a summary of our work. In the middle layer, we present two studies in the pre-processing stage. One of them summarizes trajectories for aggregate analysis and the other studies sampled trajectories in a multidimensional space. Next, we present studies in the analysis of moving objects, specifically towards the goal of outlier detection. We explored three types of outliers. The first addresses outlier objects in free moving form. The second addresses subspace outliers in a multidimensional space. And the third addresses traffic outliers in a road network.

In the figure, we also show two major directions for future work. First is the combination of moving object data with other data sources. Most current studies only focus on the moving objects themselves. This makes sense given the relatively young age of the field. But

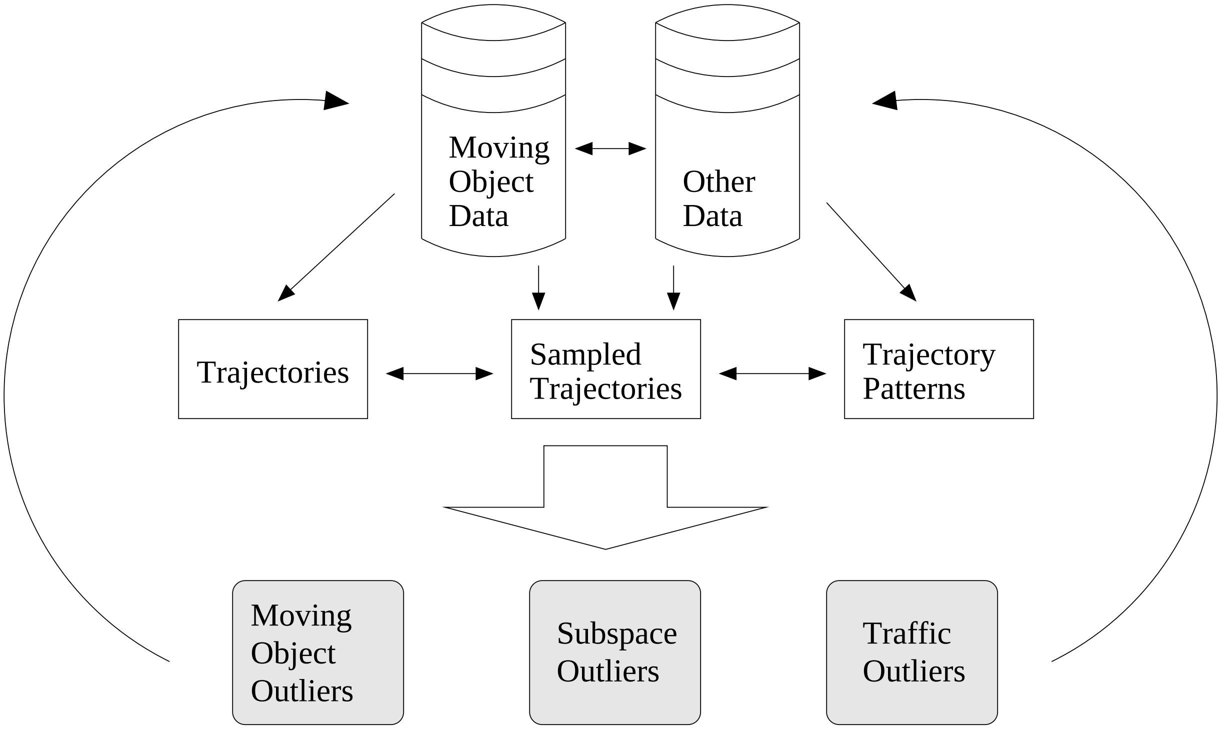


Figure 8.1: Thesis Framework & Future Direction

as better technologies develop and algorithms mature, one will soon realize that the world includes many additional things in addition to the moving objects. The linkage between them can often be more interesting than the objects alone. For example, as people (being tracked by their GPS-embedded cellphones) move throughout a shopping mall, the tracking of their movements can be linked to sales records. This linkage can be used to predict future sales numbers based on people movements. On a small time scale, *e.g.*, daily, this might seem trivial and useless. However, on a larger scale, this offers tremendous analytical information. Every quarter, a massive number of stock analysts descend on sales data in order to predict quarterly earning results. Being accurate can be the difference in millions or billions of dollars in stock sales. Often, these analysts are rather blind in their observation because the corporations do not release enough information. The analysts can only rely on small-sampled information to predict trends. However, given the tracking of people in stores, they can possibly build very accurate predictive models for quarterly earnings and stock purchasing decisions.

Another direction for future research is the feedback loop between the outputs of analysis modules and the moving objects themselves. Recently, there has been much development in two-way GPS devices for the mass public. That is, in addition to receiving information about the location and possibly traffic information, the GPS in a vehicle can also send back information about itself. More specifically, it can report back its location, its historical moving records, and its current speed. With many users reporting back such information, a collaborative filtering type of model emerges. Vehicles on the road receive, in real time, information about other vehicles and can adjust accordingly, in real time. For example, they could learn that a traffic accident has just occurred at some road segment and will then use a detour to avoid it. One can quickly see another problem that comes out of this. Road capacities are not infinite, if many vehicles all move to the same alternative location to avoid the traffic jam, they will just create another traffic jam! In these scenarios, one quickly realizes that game theory is the name of the game. That is, one has to consider the effect of a single player's actions on other players as well as the reverse. In contrast to traditional game theory, this game is much more complicated because actions are not instanteneous. They are implicit in the movements. This also gives some delay to the calculation of the game's output, which can lead to many interesting problems and solutions. With more and more moving objects being tracked and analyzed in the real world, we believe this is a field sure to receive much attention in both academia and industry in the near future, if not already!

References

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *Proc. 2000 ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS'00)*, pages 175–186, Dallas, TX, May 2001.

[2] C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'01)*, pages 37–46, Santa Barbara, CA, May 2001.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'98)*, pages 94–105, Seattle, WA, June 1998.

[4] V. T. Almeida and R. H. Guting. Indexing the trajectories of moving objects in networks. In *Proc. 2004 Int. Conf. on Scientific and Statistical Database Management (SSDBM'04)*, Santorini Island, Greece, June 2004.

[5] V. Barnett and T. Lewis. *Outliers in Statistical Data.* John Wiley & Sons, 1994.

[6] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. 1990 ACM SIGMOD Int. Conf. Management of Data (SIGMOD'90)*, pages 322–331, Atlantic City, NJ, June 1990.

[7] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis. Nearest neighbor and reverse nearest neighbor queries for moving objects. In *Proc. IDEAS*, pages 44–53, 2002.

[8] S. Berchtold, D. Keim, and H.-P. Kriegel. The X-tree: An efficient and robust access method for points and rectangles. In *Proc. 1996 Int. Conf. Very Large Data Bases (VLDB'96)*, pages 28–39, Bombay, India, Sept. 1996.

[9] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proc. 1999 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'99)*, pages 359–370, Philadelphia, PA, June 1999.

[10] T. Brinkhoff. A framework for generating network-based moving objects. In *GeoInformatica*, 2002.

[11] I. V. Cadez, S. Gaffney, and P. Smyth. A general probabilistic framework for clustering individuals and objects. In *Proc. 2000 ACM SIGKDD Int. Conf. Knowledge Discovery in Databases (KDD'00)*, pages 140 – 149, Boston, MA, Aug. 2000.

[12] Huiping Cao, Nikos Mamoulis, and David W. Cheung. Mining frequent spatio-temporal sequential patterns. In *Proc. 2005 Int. Conf. on Data Mining (ICDM'05)*, Houston, TX, Nov. 2005.

[13] Jidong Chen, Xiaofeng Meng, Yanyan Guo, and Zhen Xiao. Update-efficient indexing of moving objects in road networks. In *Third Workshop on Spatio-Temporal Database Management (STDBM'06)*, 2006.

[14] L. Chen, M. T. Ozsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proc. 2005 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'05)*, pages 491–502, Baltimore, Maryland, June 2005.

[15] Yun Chen and Jignesh M. Patel. Efficient evaluation of all-nearest-neighbor queries. In *Proc. 2007 Int. Conf. Data Engineering (ICDE'07)*, Istanbul, Turkey, April 2007.

[16] R. Cheng, Y. Xia, S. Prabhakar, and R. Shah. Change tolerant indexing for constantly evolving data. In *Proc. 2005 Int. Conf. Data Mining (ICDE'05)*, Tokyo, Japan, April 2005.

[17] H. D. Chon, D. Agrawal, and A. E. Abbadi. Range and knn query processing for moving objects in grid model. In *ACM/Kluwer MONET*, pages 401–412, 2003.

[18] François Denis. Pac learning from positive statistical queries. In *Algorithmic Learning Theory: 9th International Conference, ALT'98, Otzenhausen, Germany, October 1998*, Otzenhausen, Germany, 1998.

[19] Yang Du, Donghui Zhang, and Tian Xia. The optimal-location query. In *Proc. 2005 Int. Symp. Spatial and Temporal Databases (SSTD'05)*, 2005.

[20] E. Frentzos. Indexing objects moving on fixed networks. In *Proc. 2003 Int. Symp. Spatial and Temporal Databases (SSTD'03)*, pages 289–305, Santorini Island, Greece, July 2003.

[21] S. Gaffney and P. Smyth. Trajectory clustering with mixtures of regression models. In *Proc. 1999 Int. Conf. Knowledge Discovery and Data Mining (KDD'99)*, pages 63–72, 1999.

[22] Fosca Giannotti, Mirco Nanni, Dino Pedreschi, and Fabio Pinelli. Trajectory pattern mining. In *Proc. 2007 Int. Conf. Knowledge Discovery and Data Mining (KDD'07)*, San Jose, CA, Aug. 2007.

[23] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational operator generalizing group-by, cross-tab and sub-totals. In *Proc. 1996 Int. Conf. Data Engineering (ICDE'96)*, pages 152–159, New Orleans, Louisiana, Feb. 1996.

[24] G. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann, 2005.

[25] A. Guttman. R-tree: A dynamic index structure for spatial searching. In *Proc. 1984 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'84)*, pages 47–57, Boston, MA, June 1984.

[26] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. In *Journal of Machine Learning Research*, volume 3, pages 1157–1182, 2003.

[27] W. L. Hays. *Statistics*. CBS College Publishing, New York, NY, 1981.

[28] H. Hu, J. Xu, and D. L. Lee. A generic framework for monitoring continuous spatial queries over moving objects. In *Proc. 2005 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'05)*, pages 479–490, Baltimore, Maryland, June 2005.

[29] S. Hwang, K. Kwon, S. K. Cha, and B. S. Lee. Performance evaluation of main-memory r-tree variants. In *Proc. 2003 Int. Symp. Spatial and Temporal Databases (SSTD'03)*, pages 10–27, Santorini Island, Greece, July 2003.

[30] G. S. Iwerks, H. Samet, and K. Smith. Continuous k-nearest neighbor queries for continuously moving points with updates. In *Proc. 2003 Int. Conf. Very Large Data Bases (VLDB'03)*, Berlin, Germany, Sept. 2003.

[31] G. S. Iwerks, H. Samet, and K. Smith. Maintenance of spatial semijoin queries on moving points. In *Proc. 2004 Int. Conf. Very Large Data Bases (VLDB'04)*, Toronto, Canada, Aug. 2004.

[32] C. S. Jensen, D. Lin, and B. C. Ooi. Query and update efficient b+-tree based indexing of moving objects. In *Proc. 2004 Int. Conf. Very Large Data Bases (VLDB'04)*, pages 768–779, Toronto, Canada, Aug. 2004.

[33] Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. On discovering moving clusters in spatio-temporal data. In *Proc. 2005 Int. Symp. Spatial and Temporal Databases (SSTD'05)*, pages 364–381, 2005.

[34] I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proc. 1994 Int. Conf. Very Large Data Bases (VLDB'94)*, pages 500–509, Santiago, Chile, Sept. 1994.

[35] E. Keogh, J. Lin, and A. Fu. Hot sax: Efficiently finding the most unusual time series subsequence. In *Proc. 2005 Int. Conf. on Data Mining (ICDM'05)*, pages 226–233, Houston, TX, Nov. 2005.

[36] E. J. Keogh and M. J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Proc. 1998 Int. Conf. Knowledge Discovery and Data Mining (KDD'98)*, pages 239–243, New York, NY, Aug. 1998.

[37] K. Kim, S. K. Cha, and K. Kwon. Optimizing multidimensional index trees for main memory access. In *Proc. 2001 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'01)*, pages 139–150, Santa Barbara, CA, May 2001.

[38] E. Knorr and R. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proc. 1998 Int. Conf. Very Large Data Bases (VLDB'98)*, pages 392–403, New York, NY, Aug. 1998.

[39] G. Kollios, D. Gunopulos, and V.J. Tsotras. On indexing mobile objects. In *Proc. 18th ACM Symp. Principles of Database Systems (PODS'99)*, Philadelphia, PA, May 1999.

[40] G. Kollios, D. Papadopoulos, D. Gunopulos, and V.J. Tsotras. Indexing mobile objects using dual transformations. In *The VLDB Journal*, 2005.

[41] K. Koperski and J. Han. Discovery of spatial association rules in geographic information databases. In *Proc. 1995 Int. Symp. Large Spatial Databases (SSD'95)*, pages 47–66, Portland, Maine, Aug. 1995.

[42] Vlaho Kostov, Jun Ozawa, Mototaka Yoshioka, and Takahiro Kudoh. Travel destination prediction using frequent crossing pattern from driving history. In *Proc. 8th Int. IEEE Conf. Intelligent Transportation Systems*, pages 970–977, Vienna, Austria, Sept. 2005.

[43] Iosif Lazaridis, Kriengkrai Porkaew, and Sharad Mehrotra. Dynamic queries over mobile objects. In *Proc. 2002 Int. Conf. Extending Database Technology (EDBT'02)*, Prague, Czech, March 2002.

[44] J. Lee, J. Han, and K. Whang. Trajectory clustering: A partition-and-group framework. In *Proc. 2007 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'07)*, Beijing, China, June 2007.

[45] Jae-Gil Lee, Jiawei Han, and Xiaolei Li. Trajectory outlier detection: A partition-and-detect framework. In *Proc. 2008 Int. Conf. Data Mining (ICDE'08)*, Cancun, Mexico, April 2008.

[46] H. Li, D. Agrawal, A. E. Abbadi, and M. Riedewald. Exploiting the multi-append-only-trend property of historical data in date warehouses. In *Proc. 2003 Int. Symp. Spatial and Temporal Databases (SSTD'03)*, Santorini Island, Greece, July 2003.

[47] Xiaolei Li and Jiawei Han. Mining approximate top-k subspace anomalies in multi-dimensional time-series data. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07)*, Vienna, Austria, Sept. 2007.

[48] Xiaolei Li, Jiawei Han, and Hector Gonzalez. High-dimensional OLAP: A minimal cubing approach. In *Proc. 2004 Int. Conf. Very Large Data Bases (VLDB'04)*, pages 528–539, Toronto, Canada, Aug. 2004.

[49] Xiaolei Li, Jiawei Han, and Sangkyum Kim. Motion-alert: Automatic anomaly detection in massive moving objects. In *Proceedings of the 2006 IEEE Intelligence and Security Informatics Conference (ISI'06)*, San Diego, CA, May 2006.

[50] Xiaolei Li, Jiawei Han, Sangkyum Kim, and Hector Gonzalez. Roam: Rule- and motif-based anomaly detection in massive moving object data sets. In *Proceedings of the Seventh SIAM International Conference on Data Mining (SDM'07)*, Minneapolis, MN, April 2007.

[51] Xiaolei Li, Jiawei Han, Jae-Gil Lee, and Hector Gonzalez. Traffic density-based discovery of hot routes in road networks. In *Proc. 2007 Int. Symp. Spatial and Temporal Databases (SSTD'07)*, pages 441–459, Boston, MA, July 2007.

[52] Xiaolei Li, Jiawei Han, Zhijun Yin, Jae-Gil Lee, and Yizhou Sun. Sampling cube: A framework for statistical olap over sampling data. In *Proc. 2008 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'08)*, Vancouver, Canada, June 2008.

[53] Xiaolei Li, Zhenhui Li, Jiawei Han, and Jae-Gil Lee. Temporal outlier detection in vehicle traffic data. In *submission*, 2008.

[54] Lin Liao, Dieter Fox, and Henry Kautz. Learning and inferring transportation routines. In *Proc. 2004 Nat. Conf. Artificial Intelligence (AAAI'04)*, 2004.

[55] D. Lin, C.S. Jensen, S. Saltenis, and B.C. Ooi. Efficient indexing of the historical, present, and future positions of moving objects. In *6th International Conference on Mobile Data Management (MDM'05)*, 2005.

[56] H. Liu, F. Hussain, C. L. Tan, and M. Dash. Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 6:393–423, 2002.

[57] M. F. Mokbel, X. Xiong, and W. G. Aref. Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *Proc. 2004 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'04)*, Paris, France, June 2004.

[58] M. F. Mokbel, X. Xiong, W. G. Aref, S. E. Hambrusch, S. Prabhakar, and M. A. Hammad. Place: A query processor for handling real-time spatio-temporal data streams. In *Proc. 2004 Int. Conf. Very Large Data Bases (VLDB'04)*, Toronto, Canada, Aug. 2004.

[59] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient olap operations in spatial data warehouses. In *Proc. 2001 Int. Symp. Spatial and Temporal Databases (SSTD'01)*, Redondo Beach, CA, July 2001.

[60] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: A review. *SIGKDD Explorations*, 6:90–105, 2004.

[61] J. M. Patel, Y. Chen, and V. P. Chakka. Stripes: An efficient index for predicted trajectories. In *Proc. 2004 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'04)*, Paris, France, June 2004.

[62] T. B. Pedersen and N. Tryfona. Pre-aggregation in spatial data warehousess. In *Proc. 2001 Int. Symp. Spatial and Temporal Databases (SSTD'01)*, Redondo Beach, CA, July 2001.

[63] D. Pfoser and C. S. Jensen. Indexing of network constrained moving objects. In *GIS '03: Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pages 25–32, New York, NY, USA, 2003. ACM Press.

[64] K. Porkaew, I. Lazaridis, and S. Mehrotra. Querying mobile objects in spatio-temporal databasess. In *Proc. 2001 Int. Symp. Spatial and Temporal Databases (SSTD'01)*, Redondo Beach, CA, July 2001.

[65] J. R. Quinlan and R. M. Cameron-Jones. FOIL: A midterm report. In *Proc. 1993 European Conf. Machine Learning*, pages 3–20, Vienna, Austria, 1993.

[66] K. Raptopoulou, A. Papadopoulos, and Y. Manolopoulos. Fast nearest-neighbor query processing in moving object databases. In *GeoInfomatica*, pages 113–137, 2003.

[67] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'95)*, pages 71–79, San Jose, CA, May 1995.

[68] S. Saltenis and C. Jensen. Indexing of moving objects for location-based services. In *Proc. 2002 Int. Conf. Data Engineering (ICDE'02)*, pages 463–472, San Fransisco, CA, April 2002.

[69] S. Saltenis, C. Jensen, S. Leutenegger, and M. Lopez. Indexing the positions of continuously moving objects. In *Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, pages 331–342, Dallas, TX, May 2000.

[70] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A dynamic index for multi-dimensional objects. In *Proc. 1987 Int. Conf. Very Large Data Bases (VLDB'87)*, pages 3–11, Brighton, England, 1987.

[71] S. Shekhar and Y. Huang. Discovering spatial co-location patterns: A summary of results. In *Proc. 2001 Int. Symp. Spatial and Temporal Databases (SSTD'01)*, Redondo Beach, CA, July 2001.

[72] C. Sun, D. Agrawal, and A. E. Abbadi. Hardware acceleration for spatial selections and joins. In *Proc. 2003 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'03)*, San Diego, CA, June 2003.

[73] J. Sun, D. Papadias, Y. Tao, and B. Liu. Querying about the past, the present, and the future in spatio-temporal databases. In *Proc. 2004 Int. Conf. Data Engineering (ICDE'04)*, pages 202–213, Boston, MA, March 2004.

[74] Y. Tao, G. Kollios, J. Considine, F. Li, and D. Papadias. Spatio-temporal aggregation using sketches. In *Proc. 2004 Int. Conf. Data Engineering (ICDE'04)*, Boston, MA, March 2004.

[75] Y. Tao, D. Papadias, and C. Faloutsos. Approxiate temporal aggregation. In *Proc. 2004 Int. Conf. Data Engineering (ICDE'04)*, Boston, MA, March 2004.

[76] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *Proc. 2002 Int. Conf. Very Large Data Bases (VLDB'02)*, pages 287–298, Hong Kong, China, Aug. 2002.

[77] Y. Tao, D. Papadias, and J. Sun. The tpr*-tree: An optimized spatio-temporal access method for predictive queries. In *Proc. 2003 Int. Conf. Very Large Data Bases (VLDB'03)*, pages 790–801, Berlin, Germany, Sept. 2003.

[78] Y. Theodoridis, J.R.O. Silva, and M.A. Nascimento. On the generation of spatiotemporal datasets. In *6th Int. Symposium on Spatial Databases (SSD'99)*, Hong Kong, China, July 1999.

[79] I. Tsoukatos and D. Gunopulos. Efficient mining of spatiotemporal patterns. In *Proc. 2001 Int. Symp. Spatial and Temporal Databases (SSTD'01)*, pages 425–442, Redondo Beach, CA, July 2001.

[80] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *Proc. 2002 Int. Conf. Data Engineering (ICDE'02)*, pages 673–684, San Fransisco, CA, April 2002.

[81] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *Proc. 2005 Int. Conf. Very Large Data Bases (VLDB'05)*, Trondheim, Norway, Aug. 2005.

[82] D. Xin, J. Han, X. Li, and B. W. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In *Proc. 2003 Int. Conf. Very Large Data Bases (VLDB'03)*, Berlin, Germany, Sept. 2003.

[83] X. Xiong, M. F. Mokbel, and W. G. Aref. Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases. In *Proc. 2005 Int. Conf. Data Mining (ICDE'05)*, Tokyo, Japan, April 2005.

[84] Dragomir Yankov, Eamonn Keogh, and Umaa Rebbapragada. Disk aware discord discovery: Finding unusual time series in terabyte sized datasets. In *Proc. 2007 Int. Conf. on Data Mining (ICDM'07)*, Omaha, NE, Sept. 2007.

[85] X. Yin and J. Han. CPAR: Classification based on predictive association rules. In *Proc. 2003 SIAM Int. Conf. Data Mining (SDM'03)*, pages 331–335, San Fransisco, CA, May 2003.

[86] M. L. Yiu, Y. Tao, and N. Mamoulis. The b^{dual}-tree: Indexing moving objects by space filling curves in the dual space. In *The VLDB Journal*, 2006.

[87] Man Lung Yiu, Xiangyuan Dai, Nikos Mamoulis, and Michail Vaitis. Top-k spatial preference queries. In *Proc. 2007 Int. Conf. Data Engineering (ICDE'07)*, Istanbul, Turkey, April 2007.

[88] J. S. Yoo and S. Shekhar. A partial join approach to mining co-location patterns. In *GIS'04*, Washington, D.C., Nov. 2004.

[89] X. Yu, K. Q. Pu, and N. Koudas. Monitoring k-nearest neighbor queries over moving objects. In *Proc. 2005 Int. Conf. Data Mining (ICDE'05)*, Tokyo, Japan, April 2005.

[90] H. Zen, K. Tokuda, and T. Kitamura. A viterbi algorithm for a trajectory model derived from hmm with explicit relationship between static and dynamic features. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04)*, volume 1, pages 837–40, 2004.

[91] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. Lee. Location-based spatial queries. In *Proc. 2003 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'03)*, pages 443–454, San Diego, CA, June 2003.

[92] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'96)*, pages 103–114, Montreal, Canada, June 1996.

[93] X. Zhang, N. Mamoulis, D. W. Cheung, and Y. Shou. Fast mining of spatial collocations. In *Proc. 2004 Int. Conf. Knowledge Discovery and Data Mining (KDD'04)*, pages 384–393, Seattle, WA, Aug. 2004.

Author's Biography

Xiaolei Li was born on December 2nd, 1980 in the city of Shanghai, China. At the age of 11, he moved to Omaha, NE in the United States. In August of 1999, Xiaolei started his undergraduate studies in Computer Science at the University of Illinois at Urbana-Champaign. He graduated in May of 2002 and continued with his Masters studies starting in August of 2002. After receiving his Masters in Computer Science in May of 2004, Xiaolei continued at University of Illinois at Urbana-Champaign for his Doctorate of Philosophy in Computer Science under the guidance of Prof. Jiawei Han. Following his completion of the Ph.D., Xiaolei will join Microsoft to further his research.