Spring 2011

Dynamic-parinet (D-parinet) : indexing present and future trajectories in networks

Mou Nandi
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

DYNAMIC-PARINET (D-PARINET): INDEXING PRESENT AND FUTURE TRAJECTORIES IN NETWORKS

By
Mou Nandi

While indexing historical trajectories is a hot topic in the field of moving objects (MO) databases for many years, only a few of them consider that the objects movements are constrained. DYNAMIC-PARINET (D-PATINET) is designed for capturing of trajectory data flow in multiple discrete small time interval efficiently and to predict a MO's movement or the underlying network state at a future time.

The cornerstone of D-PARINET is PARINET, an efficient index for historical trajectory data. The structure of PARINET is based on a combination of graph partitioning and a set of composite B+-tree local indexes tuned for a given query load and a given data distribution in the network space. D-PARINET studies continuous update of trajectory data and use interpolation to predict future MO movement in the network. PARINET and D-PARINET can easily be integrated into any RDBMS, which is an essential asset particularly for industrial or commercial applications. The experimental evaluation under an off-the-shelf DBMS using simulated traffic data shows that D-PARINET is robust and significantly outperforms the R-tree based access methods.

DYNAMIC-PARINET (D-PARINET):

INDEXING PRESENT AND FUTURE TRAJECTORIES IN NETWORKS

by
Mou Nandi

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer Science

April 2011

APPROVAL PAGE

DYNAMIC-PARINET (D-PARINET):
INDEXING PRESENT AND FUTURE TRAJECTORIES IN NETWORKS

Mou Nandi

Dr. Vincent Oria, Thesis Advisor — Date
Associate Professor of Computer Science, NJIT

Dr. Cristian M Borcea, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. Dimitrios Theodoratos, Committee Member — Date
Associate Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Mou Nandi

Degree: Master of Science

Date: May 2011

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, USA, April, 2011

- Bachelor of Engineering in Production Engineering,
  Jadavpur University, Kolkata, India, MAY, 2000

Major: Computer Science

ACKNOWLEDGMENT

I would like to express my gratitude to my advisor Professor Vincent Oria for accepting me as his student, his guidance, time, contribution and most importantly keeping his patience for getting this work done. I am greatly thankful to Dr. Iulian Sandu Popa from PRiSM Laboratory, University of Versailles, France, one of the author of PARINET work for his detail guidance on the experimental set up, historical data and pre-configured queries, figures and tables used in original PARINET work, which would have cost me lot more time to create from scratch. I also want to express my gratitude to my co-advisors, Dr. Cristian M Borcea and Dr. Dimitrios Theodoratos for their time and valuable guidance during the course of this work. All credits for this work go to them.

I also want to thank my fellow student, Mr. Chris Zizac, who was kind enough to extract the New Jersey highway's loop sensor data from NJDOT database in a custom format and send to me whenever I wanted, without that I would not have to do this work.

I am grateful to my husband, Mr. Krishanu Banerjee for being very supportive and enthusiastic for last two years. I would not be able to complete the degree without him. I want to extend my thanks to my mother-in-law, Ms. Swapna Banerjee, who took care of my son, Ujan Ray, when I travelled cross-country every week for the whole semester and of course, my husband paid for my travel.

At the end, it will be an injustice if I do not mention my four year's old son for sacrificing his park-time in the weekends and (happily) watching Netflix movies for hours to let me do this work.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

DA_Q	Total number of disk accesses for a query
IA_p	Number of index accesses in a partition
IA_p^f	Number of fixed index accesses in a partition
IA_p^v	Number of variable index accesses in a partition
PA_p	Number of page accesses in a partition
N_p	Number of units (tuples) in partition p

CHAPTER 1

INTRODUCTION

1.1 Objective

With the proliferation of mobile devices capable of accurately reporting their positions in time, it has become possible to accumulate large amounts of trajectory data. Moreover, the data acquisition can be made in real-time by using the ubiquitous wireless communication systems. A wide range of applications in areas like transportation planning, traffic management, location-aware services, rely on these data. Subsequently, an important research effort went into the general field of moving objects databases (MOD). Most of these works can fit in one of the following two complementary classes: modeling spatio-temporal databases; and indexing techniques to efficiently process spatio-temporal queries.

The performance issue has become critical in spatio-temporal applications due to the large amount of data and the computation cost of geometric operators. An impressive number of access methods have been proposed for efficient processing of moving objects (MO) queries. These index methods are classified from a temporal or a spatial point of view. From the temporal perspective, some techniques aim at indexing real-time application data with the objective of minimizing the update and retrieval costs. Examples include TPR*-tree proposed by Tao, Y., Papadias, D., Sun, J. in "The TPR*-tree: an optimized spatio-temporal access method for predictive queries", STRIPES by Patel, J.M., Arbor, A., Chen, Y., Chakka, V.P. in "STRIPES: an efficient index for predicted trajectories" and ST2B-tree by Chen, S., Ooi, B.C., Tan, K.L., Nascimento,

M.A. in "The ST2B-tree: A Self-Tunable Spatio-Temporal B^+-tree Index for Moving Objects", to name a few. Some other techniques focus on indexing complete (past) trajectories of MOs and aim at reducing the retrieval costs in large datasets. Several access methods such as the MV3R-Tree by Hadjieleftheriou, M., Kollios, G., Tsotras, J., Gunopulos, D. in "Indexing spatiotemporal archives" and by Tao, Y., Papadias, D. in "MV3R-Tree: A spatio-temporal access method for timestamp and interval queries" have been proposed in this context.

From the spatial perspective, most MO access methods consider that the objects are moving freely in the space. However, in several real-life applications, the object movements are constrained (e.g., trains moving along a railroad network or vehicles moving along a road network). Taking into account the network can lead to specific models that are optimal for the data representation. A few works have proposed access methods for objects moving in networks.

The existing indexing techniques for objects moving in networks decompose the network into roads, and then index the spatio-temporal location of the MOs on each road with a specific index, e.g., a 2D R-tree. One of the shortcomings of this approach is the way the space is decomposed: it is solely determined by the road network, and takes neither into account the distribution of the trajectory data, nor the queries on the data. Hence, more recent access methods for non-constrained MOs have proposed to partition the 2D space according to the data distribution. Moreover, indexing both the spatial and temporal dimensions for a given road is not always useful, since the spatial dimension (i.e., relative positions) tends to be less selective than the temporal one in most cases.

Another important observation in the case of MO trajectory data is that the datasets can be very dynamic and can span over very long periods of time, which are expanding continuously. For example, it is not uncommon nowadays to continuously monitor the traffic in certain road networks or highway networks, and also to record all these data for future use. Indeed, numerous applications are based on analyzing the historic (trajectory) data, e.g., for location-based services, for traffic planning, for measuring the traffic impact on the environment, for infrastructure developments, etc.

A more general problem suggested by the above mentioned applications is to efficiently manage trajectory data flows. The existing techniques for indexing the current and near-future movements of MO focus on tracking the positions of a set of MO. The challenge for an index in this case is the ability to continuously adapt to the spatio-temporal distribution of the data and to find a balance within the update and query cost tradeoff. These methods discard the historical data. On the other hand, the methods that index past trajectory data consider mostly static datasets that are known in advance (since the data is historical) and that are subject to little or no changes. The main issue in this case is to optimize the retrieval cost of spatio-temporal queries. More recently, Pelanis et al. in the paper "Indexing the past, present, and anticipated future positions of moving objects", proposed an indexing technique for capturing the positions of moving objects at all points in time (past, present, and anticipated future). Nonetheless, the focus of this work is on indexing the transition from present states to recent-past states of the data, while indexing the whole past is not a concern.

Therefore, in a more general context, it would be interesting to have an access method that efficiently processes the spatio-temporal queries over the recorded history,

while continuously recording the history of trajectory data up to the current time. Note that in the general context, the focus is still on the snapshot (spatio-temporal) queries, i.e., which are evaluated only once.

This thesis work is based on PARINET, i.e., a PARtitionned Index for in-NEtwork Trajectories. Thus, we first present PARINET, an access method to efficiently retrieve the (past) trajectories of objects moving in networks. Given a data set of trajectories, PARINET proceeds by partitioning the data and by indexing the partitions with composite B+-trees. This allows exploiting the built-in B+-tree, a robust and efficient index structure that exists in every database system. Instead of using a 2D grid as in the previous methods, the partitioning of the data is based on graph partitioning theory in order to integrate the network topology. In addition, we proposed a cost model that allows tuning correctly the index structure for a given query load. The part of the work dealing with indexing constrained trajectory data has already been published by Sandu Popa, I., Zeitouni, K, this thesis advisor , Prof Oria, V., Barth, D., Vial, S. in the paper "PARINET: A tunable access method for in-network trajectories" last year. Similar to the existing approaches, the focus of that work was on indexing historical trajectory datasets, i.e., where the data are known in advance and are subject to little or no changes.

Then, the same team extended PARINET to solve the more general problem, i.e., indexing trajectory data flows. The Temporal PARINET (T-PARINET) provides an optimized handling of trajectory data flows. T-PARINET is configurable in a dynamic environment and to fulfill its goal, T-PARINET uses an on-line tuning process that creates periodically a new PARINET to index the trajectory data from the current

moment to a future moment in time. The on-line tuning process is based on monitoring a set of parameters indicating the quality of the last built index in the structure of the T-PARINET. PARINET and T-PARINET is based on graph partitioning and time interval indexing. It also presents a cost model that combines the statistics on the data and the query workload to estimate the number of disk accesses for a given index configuration. PARINET can automatically choose a good index configuration, based on the provided cost model, the data distribution and the query workload using well-known graph partitioning algorithms.

In this thesis Dynamic PARINET (D-PARINET) is proposed for indexing continuously and efficiently in-network trajectory data flows following the same principle as T-PARINET and extend the T-PARINET implementation by introducing trajectory prediction by interpolating data generated over short discrete time intervals. D-PARINET uses an on-line tuning process to automatically determine the index evolution in time. The tuning process is based on the cost model of PARINET and T-PARINET adapted to the context of indexing trajectory data flows. We characterize the query types in the network constrained MO context and provide different test scenarios.

We have implemented PARINET and D-PARINET using an off-the-shelf DBMS and validated our approach using an extensive experimentation that shows their efficiency and their scalability properties.

1.2 Background Information

1.2.1 Indexing Moving Objects in Networks

As pointed earlier, considerable attention has been paid to indexing methods for moving objects. Most of these works deal with indexing past, present or near-future positions of MOs that move freely in a two-dimensional space. There are only a few methods for indexing (past) trajectories of MOs in networks, which is the focus of this paper. The trajectories are represented with reference to a network, i.e., with the relative positions of the MOs on network edges. The main idea in the previous works is to decompose a three-dimensional problem in two sub-problems in lower dimensions and then use a combination of two-level R-trees to index the trajectories.

The approach by Pfoser and Jensen uses two 2D R-trees, one for indexing road edges and the other for accessing 2D transformed trajectory segments. The 3D (x, y, t) coordinates of a trajectory are mapped into a 2D (p, t) coordinate space using a Hilbert curve to linearize the network line segments. The same mapping is performed for queries. However, this generally leads to multiple sub-queries and may decrease the performances. For simplicity we will refer to this approach as PJ-tree in the rest of the paper.

The FNR-tree utilizes a 2D R-tree to index road segments. For every leaf node in the 2D R-tree, there is a 1D R-tree to index the objects whose trajectories cross the segments included in the leaf node at a certain period of time. A major disadvantage of the FNR-tree is its limitation in trajectory modeling. Since only the time intervals are

stored in the 1D R-tree, it is assumed that the objects cannot stop, change speed or direction in the middle of a road segment.

This limitation is addressed by the MON-tree. The MON-tree is composed of a 2D R-tree (the top R-tree) that indexes the network edges and a set of 2D R-trees (the bottom R-trees) that index the object movements along the edges. An additional hash structure used to map each edge to its corresponding tree helps speed up insertions. Given a 3D spatio-temporal query, the top R-tree is used to find the precise intersection between the spatial part of the query and the network. Based on this intersection, a set of sub-queries is generated for each intersected part of each edge involved. Then, the corresponding bottom R-trees are accessed in order to respond to the sub-queries. MON-tree can handle two network models: an edge oriented model and a route oriented model. The experimental evaluation of the MON-tree against the FNR-tree shows that the first method always outperforms the second. The MON-tree on a route oriented network model shows better results.

1.2.2 Indexing MO Trajectory Data Flows

To the best of the authors' knowledge, no work in the MOD area considers the problem of continuously indexing data flows of trajectories (for constrained or non-constrained MOs) to optimize spatio-temporal queries. The closest works we found are the ones that focus on continuously tracking a set of non-constrained moving objects.

These works mainly index two types of queries. A first group of methods, such as TPR-tree, TPR*-tree, Bx-tree or ST2-B-tree have been proposed to optimize snapshot spatio-temporal queries that refer to present or near-future times. Typical examples of

such queries are: “Which MOs are within 1 km of my location right now?” or “What will be the number of MOs in the city center ten minutes from now?”

A second group of methods, such as SINA, Q-index and CNN, optimize continuous spatio-temporal queries. An example query in this context is: “Continuously report the hotels within 5 km of my location”. Both the first group and the second group of works mainly focus on spatio-temporal range queries, but these approaches remain applicable to a broader class of spatio-temporal queries, e.g., nearest-neighbor or aggregate queries.

In the above mentioned works, only the current positions (along with, in some cases, the current velocity vector used to predict near-future positions) of the tracked MOs are indexed. The past states representing the MOs’ trajectories up to the current time are discarded. A few works deal with indexing the movements of non-constrained MOs at all points in time. These works include the proposal of Sun et al., the BBx-tree and the RPPF-tree. A common feature in these approaches is the focus on indexing the transition from present states to recent-past states of the data, while indexing the whole past is not a concern.

Sun et al. proposed a method to approximately answer aggregate spatio-temporal queries at all moments in time. The method is based on a multidimensional histogram representing the spatio-temporal evolution of the distribution of the moving objects. A main memory structure is used to keep the distributions corresponding to the current time and to the recent past. Older states of the histogram are migrated to the secondary storage and are simply indexed using a single packed B-tree or a 3D R-tree.

BBx-tree is an extension of the Bx-tree and consists in an array of indexes that store the old phases of a rotating Bx-tree. For the past states, each index covers a time interval equal to $1.5T_{max}$, where T_{max} represents the anticipated maximum duration between two consecutive updates of any moving object. Also, the lifespans of two consecutive indexes overlap on an interval of length T_{max}. Since T_{max} can be very small in comparison with the length of the recorded history of the movements of moving objects, it is expected to have a large number of indexes in the structure of a BBx-tree even for relatively short periods of time (e.g., of a few weeks). Moreover, the short lifespan of an index and the index overlap, make that normal range queries intersect several indexes, which will increase the query processing overhead. Another disadvantage of BBx-tree is that it does not record the real position of the MOs, but only estimations of the MOs' location.

RPPF-tree is another proposal to index the positions of non-constrained moving objects at all points in time for time slice queries. RPPF-tree is based on the TPR-tree . Pelanis et al. applied the partial persistence paradigm to the TPR-tree to enable it to retain and query past states of the indexed data. The focus here is on the application of partial persistence to the TPR-tree, which is not a trivial task. Once more, a single R-tree-like structure is used to index all the data spanning from the recording start instant in the past to the current time. This may lead to an important degradation of the index structure due to the high number of index entries.

CHAPTER 2

THE CONTEXT OF PARINET

2.1 Network Model

PARINET uses two representations for the road network: a geometric view and a topologic view. The geometric view (or 2D view) captures the approximate geographic locations of the road network components. This is the base view of the road network. The topologic view uses a graph in order to represent the road sections and the intersections. It is useful in the partitioning of the network.

The geometric representation of a road network is given by a tuple $RN^{2D} = (S,C)$, where S is a set of segments and C is a set of connections. A road segment $s \in S$ is a 2D line segment defined by (p_s, p_e), where $p_s = (x_s, y_s)$, $p_e = (x_e, y_e)$ and $p_s \neq p_e$; p_s and p_e are respectively the start and end points of the segment. A connection $c \in C$ is a tuple (p, S^c), where p is a geographical point that represents the location in the 2D space of the connection and S^c is a set of segments that meet at the connection. The list of segments in Sc should have p as one of their end points. Figure 2.1 gives a simple example of a geometric representation of a road network.

2.1.1 Definition 2.1

Given a road network RN^{2D} as described above, we define a road in RN^{2D} as $Road = (rid, S^c, start)$, where rid is a unique identifier, S^c is a set of connected segments that form a non self-intersecting polyline in RN^{2D} (which may be open or closed (a

cycle)) and start is one of the two endpoints of the polyline. Each segment belongs to one road only.

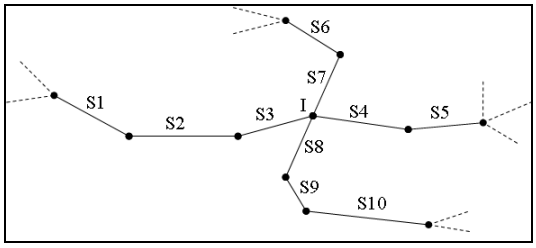

Figure 2.1 Example of a geometric representation of a network

2.1.2 Definition 2.2

Given a road network RN^{2D} as described above, we define the set of junctions in RN^{2D} as $\text{Junctions} = \left\{ j \middle| j \in C \wedge \text{card}(c(S^j)) \geq 3 \right\}$.

Different granularities can be superimposed to a road resulting in different network models.

2.1.3 Definition 2.3

For a given road network RN^{2D}, we define three possible network models:

1. Segment oriented network model: each segment corresponds to a road.
2. Edge oriented network model: each road is defined as the polyline between two junctions.
3. Route oriented network model: the complete roads are considered without split. They can extend over the junctions. Notice that several configurations are possible for the route model on the same road network.

In the example in Figure 2.1,

1. 10 roads (S1,…,S10) in the segment oriented model;

2. 4 roads: (S1, S2, S3), (S4, S5), (S6, S7) and (S8, S9, S10) using the edge oriented model;
3. 2 roads: (S1, S2, S3, S4, S5) and (S6, S7, S8, S9, S10) in the route oriented model.

In the sequel, the general term road network is used to denote a road network modeled as one of the three possible network models.

2.1.4 Definition 2.4

Given a road network RN^{2D}, we define a position in the network space as a pair (rid, pos), where rid is a road identifier and $pos \in [0,1]$ is the relative position on the road measured from the start end point of the road.

This is closely related to the concept of linear referencing widely used in GIS for transportation and available in DBMSs as Oracle Spatial or GIS tools as ArcGIS.

2.1.5 Definition 2.5

Given a road network RN^{2D}, we define a road connection rc as a tuple (p, R^c) where p is the geographical point location in 2D space of the road connection and R^c is the set of roads that meet at the connection. p has the same coordinates as one of the two end points of each road in the given connection.

Based on the 2D representation of a road network RN^{2D}, we construct the topologic representation of the network. In this representation, a network is defined as an undirected weighted graph $G = (V, E)$ with V a set of vertices and $E \subseteq V \times V \times N$ a set of edges, where N is the set of natural numbers. Each $v \in V$ corresponds to one road connection in RN^{2D}. Given $v_1, v_2 \in V$, there is an edge $e = (v_1, v_2, w)$ in G if there is a

road in RN^{2D} between the corresponding road connections. The weight w is given by the function W, which depends on the data distribution. Notice that in our network model, the roads are non-oriented. But taking into account the traffic orientation is a straightforward extension that can be achieved by splitting the two-way roads into two edges.

2.2 Data Model

As mentioned earlier, we intend to index the trajectories of the MOs in a network. An object moving on a road network reports its position at different moments in time. We assume that such an update is issued each time the MO changes its speed or passes on a different road in the network. An update contains the identifier of the MO, the network position (as given in Definition 3.4) and the associated time instant: $(\text{moid}, \text{rid}, \text{pos}, \text{t})$. We define the trajectory of a moving object as a non-regulated sequence of units (i.e., the time intervals are not of equal size). Each unit is a tuple defined by two consecutive updates: $(\text{moid}, \text{rid}, [\text{pos}_1, pos_2], [\text{t}_1, t_2])$; t indicates a time instant, while pos gives the relative position on the road at the beginning and the end of the time interval. For each unit, it is assumed that the MO moves at constant speed, i.e., a linear interpolation is considered over each interval. Given a road, the relative position on the road and the time can be viewed as the two orthogonal axes of a 2D space. In this space, we denote by unit segment the 2D line segment bounded by the points $(\text{pos}_1, \text{t}_1)$ and $(\text{pos}_2, \text{t}_2)$.

2.3 Query Types

There are several types of queries that have been studied in the field of MOD, such as range queries, spatio-temporal join, nearest neighbors, within distance (or e-distance join) or skyline queries to name but a few. Among these query types, the range and the nearest neighbor queries are, probably, the most studied in the context of MOD. In this paper, we consider these two types of queries and focus on the range queries since the nearest neighbor queries can be brought down to a succession of range queries.

The range queries are composed of a spatial part and a temporal interval, $Q = (Q_s, Q_t)$. The queries return either all the MOs that have lied within the area of Q_s, at a certain time interval Q_t, or only the pieces of the trajectories that overlap the query. We consider two types of range queries: 2D queries and path queries. The difference between the two types of queries lies in the spatial part Q_s.

The spatial component of the first type of queries is a 2D region. Hence, the 2D queries represent "standard" range queries .Thus, Q_s is a 2D region (usually a rectangle). In the rest of the paper, we will refer to this type of queries as 2D queries. To support 2D queries, a transformation of Q_s is performed first. The exact intersection between the 2D region and the network is computed. Then the initial region in Q_s is replaced with the intersected network region.

Formally, the new Q_s is a set of road sections: $Q_s = \{rs_1, rs_2, \ldots rs_n\}$ where $rs_i = (rid_i, [pos^i_{11}, pos^i_{12}], [pos^i_{21}, pos^i_{22}], \ldots [pos^i_{k1}, pos^i_{k2}])$ and $\{rs_i \neq rs_j\}$ and $pos^i_{m1} \leq pos^i_{m2} \wedge pos^i_{m2} < pos^i_{(m+1)1}$. Each rs_i represents a set of disjoint and ordered intervals on one road [1]. Multiple intersection intervals with the query region are

possible when the road is a polyline, which is the case for an edge or route network model. Usually, one can use a 2D R-tree over the network to speed up the computation of the mapping between a 2D region and a network region.

The constrained movement suggests another type of useful query. For example, "find in a database all the MOs whose trajectories intersect a given MO trajectory", or "find the number of MOs that traverse a given road section at a certain time (interval)" are path queries that need to refer to the network. Path queries represent a new type of range queries that we introduce. In a path query, the spatial part, Q_s, represents a path in the network, i.e., a sequence of connected road sections. For this type of queries, no mapping is needed from the 2D space to the network space and Q_s has the same formalization as above.

Beside the range queries, the nearest neighbor (NN) queries are another popular type of query in MOD. Moreover, there are several types of NN queries, e.g., reverse NN, aggregate NN or continuous NN. The interested reader can refer to [27] for a discussion about the processing of conventional NN queries by PARINET. That is, given a (static) position in the network space and a time interval Q_t, the query returns the k MOs that were closer (w.r.t. the network distance, i.e., the shortest path between two network positions) to the network location during Q_t.

2.4 Observations

This subsection gives a short informal intuition of the PARINET index structure. PARINET uses a filtering and refinement approach. The main idea of our proposal is that an approximate index search could deliver very good performances in terms of

computation time, while offering at the same time good results in terms of physical accesses. The overall performance of such an access method can surpass the "exact" index search used in the existing methods.

Actually, in a network space, the spatial dimension is composed of a discrete component (the road identifier) and a continuous component (the relative position on the road). T-PARINET is based on the four following observations.

2.4.1 Observation 1

The relative position dimension is usually less selective than the temporal dimension. Using an index on time for filtering candidates followed by a refinement step should be more efficient than using an R-tree on the two dimensions.

The MON-tree and the PJ-tree fully index the bi-dimensional space (relative positions and time) with a 2D R-tree. Nevertheless, it is expected to have an important amount of overlapping of the indexed units in the spatial dimension, because in general, trajectories traverse entirely the road segments in their path. Moreover, except for queries on very small regions, the usual queries cover many road segments. Therefore, indexing only the temporal dimension might be more efficient, since time is more selective in this case. For this reason, a B+-tree combined with sorted data is used on the time components. This offers an efficient sequential range scan of the tuples that intersect the temporal query interval Q_t.

2.4.2 Observation 2

The partitioning of the network space should not be made only on a road identifier basis, as it is the case for the existing methods. It should be based on the data distribution and the network topology.

Indeed, while the alternative of one index per road offers the advantage of an exact filtering on one component of the spatial dimension, it nevertheless has a few shortcomings. The partitioning is strictly related to the static road view of the network and does not consider the data statistics (distribution of MOs over the network). This is an important aspect and is even more relevant in a historical context. Moreover, the performance of the existing methods, e.g., MON-tree depends on the granularity of the employed network (section, edge, or route based model). Another argument is that a network can contain several thousand roads and having a separated index for each one could degrade the system performance even for small datasets.

Instead, an index structure is proposed, that takes into account the data distribution over the network and the network topology. The network will be partitioned in network regions that will be balanced with respect to the amount of data in each region. Therefore, the parts of the network with less traffic (e.g., the peripheral ones) will have larger extents than the busy zones (e.g., the central ones). Queries are most of the time defined on regions where road segments are close or connected. A general rule is to group together the objects that are close, which will help return more results in a few page accesses (for instance, R-trees are based on this rule). Because a network is being considered, the grouping should take into account the connectivity of road segments, i.e., the network topology, in addition to the data distribution.

2.4.3 Observation 3

The access method should be supplemented with a good quality cost model that will allow (self) tuning the structure for better performances.

2.4.4 Observation 4

In the context of continuous indexing of trajectory data flows, the access method should be able to adapt to the variation in time of the data distribution and density. Also, the access method should be efficient w.r.t. insertions and robust to massive index updating.

Trajectory data are inherently divers in space and time. For example, the central part of a road network is more circulated than the peripheral parts. Also, the traffic can be denser during peak times on working days than during the week-end. In the case of indexing trajectory data flows, the access method should be able to adapt to these variations, since different index configurations are near-optimal for different periods of time. Moreover, the update efficiency of the index and its robustness to massive updates are essential in this context. Once again, having an access method that is based on the B+-tree index appears to be the right choice due to the efficiency of the B+-tree in performing update operations.

CHAPTER 3

PARINET AND D-PARINET INDEX STRUCTURE

3.1 Index Structure

In this section, PARINET is introduced for indexing datasets of in-network trajectories. PARINET and T-PARINET constitute the foundation for D-PARINET.PARINET is capable of answering two kinds of queries on historical constrained trajectories, namely range and nearest neighbor queries. First the index structure and its operations is presented followed by a cost model based on query and data sizes, and formalizes PARINET tuning in terms of a graph partitioning problem. Finally, it is shown that how one can automatically tune PARINET for a better performance, given a road network, the distribution of the data to be indexed, and an expected query workload.

Based on the above-mentioned observations, the approach is to create a B+-tree index on time intervals for the set of roads in each partition (retuned by the partitioning phase) rather than creating an index for each isolated road. The partitioning is based on both the data distribution and the network topology, i.e., the partitions are balanced in terms of the amount of data and the partitions separate the network into regions (i.e., connected or close roads are grouped in the same partition).

The discussion on how one can choose a good number of partitions and how the partitioning is obtained is presented later. For now, it is considered that this aspect is solved and a good partitioning can be obtained for a given network and a given data distribution. As a result of this operation, each road will be assigned to a certain partition (cluster).

Given a dataset D containing trajectories of MOs in a network as a set of trajectory units: $D = \{(moid, rid, [pos_1, pos_2], [t_1, t_2])\}$, the index is built in three steps: partitioning the trajectory units based on their road identifiers, sorting the partitions on the time intervals and indexing each partition using a composite B+-tree on (t_1, t_2) interval. Note that an interval-based B-tree such as the RI-tree [10] could be used for indexing time intervals, but we chose a simple B+-tree to allow an easier implementation. The index structure is quite simple. An example is given in Figure 2. A table RP (Road Partitioning) that contains one entry for each cluster keeps some basic information on the partitioning: the list of road identifiers for a cluster and a pointer to the B+-tree index over the unit segments in the cluster. As we partition the data according to the spatial dimension, the time (t_{min}, t_{max}) represents the entire spanning time of the indexed trajectories. Therefore, only one RP table is necessary to report the relationship between the partition attribute (i.e., the rid) and the partition index.

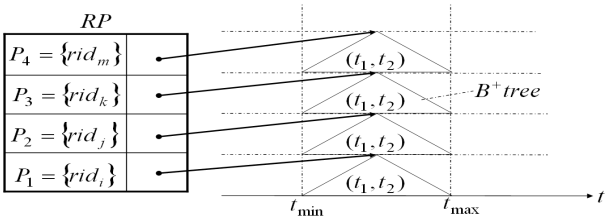

Figure 3.1 Example of PARINET index structure

3.2 Query Search Processing

Given a (2D or path) spatio-temporal range query $Q = (Q_s, Q_t)$ where $Q_s = \{rs_1, rs_2, \ldots rs_n\}$ and $Q_t = [t_s, t_e]$, PARINET can find all the objects that have traversed the road sections in Q_s during the time interval of Q_t, or simply return the

trajectory units that intersect Q. Data retrieval is performed in three steps. First, we identify the partitions that contain the road identifiers in the query, i.e., the spatial filtering step. Then, we use the B+-tree indexes of the selected partitions and look up candidate data, i.e., the temporal filtering step. Finally, we perform an exact match search among the candidates, i.e., the refinement step.

Based on the set of road identifiers $\{rid_{q1}, rid_{q2}, \ldots rid_{qn}\}$ in Q_s and on the distribution table RP, we determine the set of partitions $\{P_{p1}, P_{p2}, \ldots P_{pm}\}$ that include all the roads in a given query. Note that $m \leq n$, but in general $m < n$ and we might also have $m << n$ depending on partition and query sizes. This means that the total number of searched partitions is smaller than the total number of accessed roads in general, as it is the case in a road oriented partitioning.

Then, for each accessed partition we perform a range scan by using the B+-tree index in order to find the data pages that temporally overlap Q_t (see Figure 3). It can be noted that this may lead to false positives, because the filtering is based only on time and does not consider the road identifiers or the relative positions on the road. However, the capability of accessing groups of roads that are likely to appear together in a query will lower the number of false positives.

Finally, at the refinement step, for each candidate data we determine if it truly intersects Q, i.e., the unit segment intersects one of the sub-query windows. The actual intersection between the unit segment and the sub-query window is computed only if the unit MBR is not completely covered by the window.

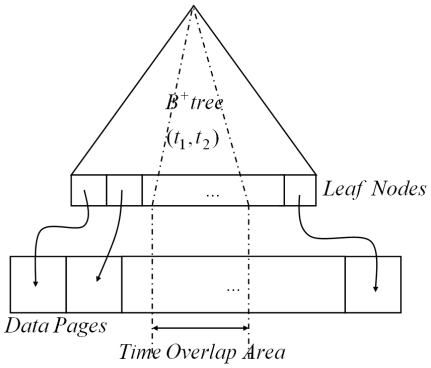

Figure 3.2 Example of index range search

3.3 Data Partitioning

3.3.1 Problem Statement

PARINET is based on the partitioning of the road network. Moreover, the partitioning must take into account the data distribution over the network (e.g., total number of unit segments for each road) and the network topology. It is clear that, for a given query load, different partitioning of the same data will lead to different performances. The goal is to automatically find the best partitioning scenario for a given query load. This is possible as the network and the data to be indexed are known in advance.

This section presents a cost model that estimates the number of disk accesses necessary to answer a query load, given a certain configuration of the PARINET index. Then, using the cost model, the partitioning problem can be re-written as an optimization problem and use a graph partitioning algorithm to resolve it. Overall performance (mainly the response time) of the index is considered to be directly related to the number of disk accesses.

3.3.2 PARINET Cost Model

This section presents a cost model that estimates the number of physical disk accesses for a given query and index configuration. The total number of disk accesses for a given query is the sum of the physical accesses in each accessed partition. The table RP , which gives the distribution of road identifiers in the partitions, is sufficiently small to fit in main memory. For each accessed partition, disk accesses for the range scan in that partition is known. A range scan comprises the index search and the data page scan. Formula (3.1) for the total number of disk accesses can be read as,

$$DA_Q = \sum_{p \cap Q_S} (IA_p + PA_p) \tag{3.1}$$

The data access cost is the number of pages containing the data that overlap with Q_t. Given the distribution of the data in time ρ_p^t, the number of pages read is:

$$PA_p = Pages_p \times \int_{Q_t + T_{\max}} \rho_p^t \cdot dt = \frac{N_p}{BS_d} \times \int_{Q_t + T_{\max}} \rho_p^t \cdot dt \tag{3.2}$$

For simplicity, a uniform temporal distribution is considered, such as $\rho_p^t = \rho_p = const$. In this case Formula (3.2) becomes,

$$PA_p = \frac{N_p}{BS_d} \times (|Q_t| + T_{\max}) \times \rho_p \tag{3.3}$$

Where $|Q_t| = t_e - t_s$

T_{max} decreases the temporal selectivity of the query by enlarging the query time interval. The problem of long time intervals is well-known when indexing time related data. The usual solution is to decompose long time intervals into several smaller intervals. The drawback is that this will increase the data set size. However, this is not a problem with

trajectory data sets because only a small percentage of the time intervals are long, i.e., there are few MO that are moving very slowly and that issue very rare updates. In general, constrained MOs such as vehicles moving in a road network need to report their location frequently to have an accurate view of their trajectories. Hence, T_{max} is expected to be much smaller than Q_t and to have a limited impact on the query cost.

Table 3.1 Nomenclature

DA_Q	Total number of disk accesses for a query
IA_p	Number of index accesses in a partition
IA_p^f	Number of fixed index accesses in a partition
IA_p^v	Number of variable index accesses in a partition
PA_p	Number of page accesses in a partition
N_p	Number of units (tuples) in partition p
$Pages_p$	Number of data pages in a partition
ρ_p^t	Temporal data distribution in a partition (percentage of $Pages_p$ per time unit)
T_{max}	Maximum length of the unit time intervals in the dataset
BS_i	Index block size (number of entries) per index page
BS_d	Data block size (number of entries) per data page

The number of index accesses is composed of a fixed cost and a variable cost. The fix cost comprises the accesses performed to reach the leaf nodes from the index tree root, which is equal to the tree height. The height of a B+-tree is equal to the number of levels in the tree including the root level. This can be computed based on the number of index entries and the tree fanout,

$$IA_p^f = \lceil \log_{fan} N_p \rceil \tag{3.4}$$

A typical value for IA_p^f is 3 when $fan \approx 100$ and the number of index entries is in the millions of tuples. The variable index cost reflects the number of pages with leaf nodes that overlap with Q_t. Similar to PA_p, it can be shown that ,

$$IA_p^v = \frac{N_p}{BS_i} \times (|Q_t| + T_{\max}) \times \rho_p \tag{3.5}$$

From Formulas (3.1), (3.3), (3.4) and (3.5) the following can be obtained:

$$DA_Q = \sum_{p \cap Q_s} \left[\lceil \log N_p \rceil + N_p (|Q_t| + T_{\max}) \rho_p (\frac{1}{BS_i} + \frac{1}{BS_d}) \right]$$

For the sake of simplicity, we consider that Q_t is implicitly enlarged with T_{max} in the following. The final formula for the number of disk accesses is,

$$DA_Q = \sum_{p \cap Q_s} \left[\lceil \log N_p \rceil + N_p \cdot |Q_t| \cdot \rho_p (\frac{1}{BS_i} + \frac{1}{BS_d}) \right] \tag{3.6}$$

One advantage of PARINET is that it allows a simple estimation of the disk accesses for a given query load, based on some statistics on the indexed data. This estimation can be used to automatically tune the index for a better performance. In short, the average area of network partitions can be modified by changing the total number of partitions n. Intuitively, given a query of a certain size, the number of disk accesses needed to answer the query will decrease with the partition size, because less false positives will be examined. However, increasing the number of partitions after a certain point will result in a performance loss. This is due to the fact that more partitions need to be considered, which increases the fixed index physical accesses and query overhead.

3.3.3 Using Graph Partitioning

Assuming that the above cost model is accurate, the performance of the PARINET for a given configuration can be estimated without effectively constructing the index. Therefore, some of the possible configurations can be tested and the best one can be materialize with respect to the cost model. A possible index configuration corresponds to a network partitioning into a given number of parts that respects some given constraints.

Graph partitioning is an important problem that has been extensively studied in the last decades. The problem is to partition the vertices of a graph in n roughly equal parts, such that the number of edges connecting vertices in different parts is minimized. The problem was extended to graphs where each node and each edge can have weights. Therefore, the resulting partitions can be balanced in term of node weights instead of number of nodes, for example. The graph partitioning problem is NP-complete. However, many algorithms have been developed to find high quality partitions extremely fast based on specific heuristics. Public implementations are also available, e.g., METIS.

As formulated in Section 3.1, the constraints imposed by PARINET on the network partitioning, can be entirely satisfied by the graph partitioning algorithms. The formalization of the approach is the following: given an undirected network graph $G = (V, E)$ and a dataset D (as described in Section 3.2), we compute the weight function of the graph roads $W : E \rightarrow N$. W associates for each road in G the number of units from D on that road. Let $L(G)$ be the line graph of G. W is a node weight function of $L(G)$. Let $P = \{P_1, P_2, \ldots P_n\}$ be the partitioning of $L(G)$ in n parts, such that the partitions are contiguous and balanced in terms of total weight. Let

$Q_L = \{Q_1, Q_2, \ldots Q_k\}$ be a query load. We define the quality indicator of P over L(G) as, $QI^n_{Q_L} = \sum_{i=1}^{k} DA_{Q_i}$, where DA_{Q_i} is computed by formula (3.6).

The goal is to find the partitioning such that QI_{Q_L} is minimal (Algorithm 1 in Appendix C). The idea is to implement a program that is based on METIS and that returns the partitions with the best QI_{Q_L} by iterating through the possible index configurations. METIS takes as input a weighted node graph and a number m of parts (line 5 in Algorithm 1). It partitions the input graph in m parts such that the partitions are fairly balanced and contiguous (although this is not guaranteed, non-contiguous portions are exceptions), which is conform to our demands for the partitioning of the road network (cf. Observation 2). By iterating with m from 1 to $card(E)$, we choose the partitioning with the best QI for the materialization of the index structure. Notice that our experimental results showed that a step of 100 for m in the iteration is sufficient because usually $QI^m_{Q_L}$ has small variations with m. Thus, the computation time for the optimal partitioning takes about one minute on our testing machine, which is negligible compared to the time necessary for testing several index configurations. For example, it takes several minutes to index about one million trajectory units. The time required to test the index performance needs also to be considered. Notice also that the partitioning algorithm can work with any network granularity, i.e., segment, edge or route (cf. Definition 3.3), since a graph representation can be built for the road network for each of the possible network granularities.

3.4 T-PARINET and D-PARINET

Trajectory datasets are very dynamic, i.e., characterized by frequent updates. In general, the updates do not concern the exiting data (although this type of update is possible, it is less probable), but rather about inserting new (parts of) trajectories to the dataset as time goes. New data can be added continually either as periodical large batches of updates or by continuously logging the individual updates coming from tracking a group of moving objects. Typically, the newer data are also the most recent from a temporal point of view.

In this context, a good access method should not only offer a good performance in term of querying the trajectories dataset, but it should also perform efficiently the updates and should have a robust performance in time. Ideally, an index should be capable of integrating at a low cost the continuous incoming updates, while allowing to process queries over the complete dataset. Moreover, the query and update performances of the index should not degrade in time.

Clearly, using a single index structure is not an appropriate solution for two main reasons. First, the performance of most indexes degrades with the size of the dataset. The query performance of the R-tree (frequently used in this context) degrades with the number of indexed entries due to an increase in the number of overlapping MBBs. Also, any tree-like index will continue to grow with the number of index entries. Although the increase in the tree height is logarithmic this aspect can not be neglected in the case of virtually infinite datasets such as trajectory datasets. Second, as indicated in the previous section, trajectory data are inherently divers in space and time. An index structure such as PARINET can balance the spatio-temporal diversity by choosing the best configuration for a given dataset, i.e., for a dataset corresponding to a time interval. However, the

spatial distribution of the data can change between the observed time periods. Therefore, different index configurations are near-optimal for different periods of time.

In order to achieve near-optimal index performance for different periods of time Temporal PARINET (T-PARINET) is designed to continuously index the trajectories of in-network moving objects. Given a road network, T-PARINET periodically creates a new PARINET index that will span over a certain temporal window. The structure of the new index is determined based on an expected spatial distribution of the data and an expected query size by using an extended PARINET cost model. The construction of a new index is triggered based on two parameters that are continuously monitored, i.e., the current index degradation (due to the difference between the expected and the real data distribution and query size) and the expected degradation (due to an increase in the index height as the data accumulate).

D-PARINET follows the same strategy as T-PARINET for indexing continuous trajectory updates but while T-PARINET manages past and future data, D-PARINET takes a different approach to index present and future data and hence supports predictive queries for near-future time period.

3.5 D-PARINET Structure and Operations

D-PARINET uses the same structure as T-PARINET index with a component index for a current to near-future time interval. T-PARINET indices are associated with different time intervals covering the index lifespan from t_0, corresponding to the oldest data in the trajectory dataset, to the current time (t_c).Each component index $PARINET_i$ is associated to a time interval $T_i = [t_{i-1}, t_i)$. The time intervals partition the lifespan $[t_0, t_c)$ of the

global index. The time intervals of the component indexes are disjoint. A Time

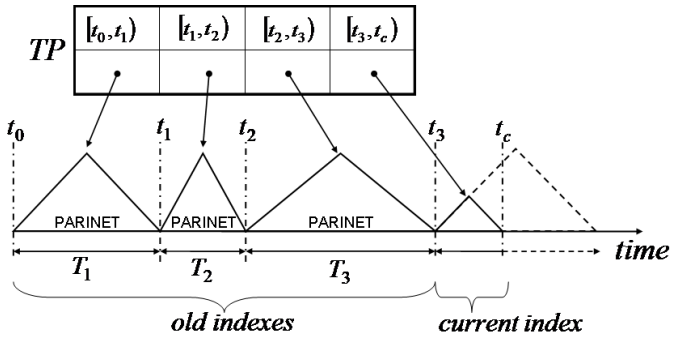

Figure 3.2 Example of a T-PARINET Index Structure

Partitioning table (TP) has one entry for each component index, which contains the corresponding lifespan and a pointer to the index.

D-PARINET uses the same current index component for managing current and interpolated future trajectory data for prediction purpose with a lifespan $[t_c, t_{c+\delta})$. Where the δ represents the time window in future over which the current trajectory data is interpolated.

3.5.1 Search Algorithm

Given a spatio-temporal range query $Q = (Q_s, Q_t)$, where $Q_s = \{rs_1, rs_2, \ldots rs_n\}$ and $Q_t = [t_s, t_e]$, the query processing for D-PARINET will be same as T-PARINET.

3.5.2 Index Evolution in Time

Given a dataflow of moving object updates $(moid, rid, pos, t)$ in a road network, we can build a D-PARINET to index the current and future trajectory data following the same structure used in T-PARINET current index component. At time t_c the structure of the index is initialized with an empty PARINET. The structure of a PARINET index is

determined based on the distribution (i.e., the temporal density of trajectories on each road) of the trajectory dataset that needs to be indexed and expected query load. In the case of continuous indexing a dataflow of trajectories the data (distribution) is not exactly known in advance. Hence, the structure of a new component index in D-PARINET will be computed based on an expected distribution of the data.

There are several ways of anticipating the spatio-temporal distribution of the data. For example, this can be based on statistics of the traffic from previous observations in the road network. For the I-80 highway loop detector data, it fits the Poisson model. If this type of information is not available, one could consider, for instance, a uniform trajectory distribution (although more elaborated models could be easily devised) for the index. Then, as the time passes and new component indexes are instantiated, the past distribution of the data from old indices can be used to foresee a possible future trajectory distribution in D-PARINET. Nonetheless, the index should be robust w.r.t. both the data distribution and the query size.

3.5.3 Optimizing Update Operations

Given a D-PARINET structure that continuously stores and indexes a flow of trajectory updates, only the current index in the structure is modified by the updates. The current index, which is a PARINET index, consists in a forest of B^+-trees over clusters of trajectory units. The trajectory units in each cluster need to be kept sorted on the units' time interval $[t_1,t_2]$ to ensure an optimal query performance of the index. This constraint is easy to preserve since the updates are inherently chronologically ordered. However, an in-memory buffer that stores all the updates in the time interval $[t_c - T_{max},t_c)$ is needed to be certain that the new trajectory units are inserted sorted on the units' time interval. The

reason is that the time intervals can have different lengths, but insertions occur once t_2 is known.

In the context of adding intensively and continuously new trajectory units to the indexed dataset, the cost of the insert operation becomes crucial for the index throughput. Moreover, the robustness of the index with regard to the insert operation is also important. PARINET is based on the B^+-tree index. This type of approach has an important advantage over the existing methods that are based on the R-tree index. The index operations in a B^+-tree (e.g., search, insertion, and deletion) can be performed more efficiently than in an R-tree. The difference in performance between the two structures is even more significant in a concurrent environment. This represents an essential aspect for real-time applications, where frequent queries and updates arrive simultaneously. Moreover, in our context the insert operation can be performed even more efficiently, since the new data can be directly appended to the existing data, i.e., the index is expanding only to the right as seen on the time axis.

Although it is expected that PARINET would offer a good performance w.r.t. the cost of the insert operation, some simple optimizations, yet having potentially an important impact on the update cost, can still be considered in this context. Recall that PARINET partitions the dataset over a number of clusters (corresponding to network regions) that are each indexed with a B^+-tree. In the case of a non-partitioned index, the buffered data page is copied to disk at the cost of one I/O operation and then the index is updated at the cost of one or several I/O operations. On the other hand, since the index is partitioned in our case, the buffered updates will generally fall in different partitions. Therefore, one page of buffered updates will require several disk accesses to copy to disk,

i.e., equal to the number of involved partitions. Moreover, all the local indexes in the affected partitions need to be updated, increasing even more the insertion cost.

Hence, the partitioning used by PARINET, which greatly improves the query performance in a static environment, can have a reverse effect on the insertion operations. This shortcoming can be easily avoided. Instead of buffering the updates page-wise, i.e., gathering one page of updates before copying to secondary storage, one could use a partition-wise buffering, i.e., one in-memory data page for each partition. A simple in-memory hash structure having one package (e.g., of a disk page size) for each partition can be used. The updates are committed to disk only when a package overflow occurs. The insertion of the trajectory units in a package will only affect one partition, thus minimizing the operation cost.

3.5.4 D-PARINET Cost Model

In this section the cost model of PARINET is revisited in the context of D-PARINET. We define the constraints and the parameters needed by D-PARINET for an on-line tuning of the evolution of the index structure in time. The main idea is that the construction of a new component index is triggered whenever the degradation of the current index exceeds a certain defined limit. T-PARINET cost model considers this and we adopt that model for D-PARINET. However, in case of D-PARINET the time interval covered by each component index should also be able to provide a good query results in terms of prediction quality.

Given a spatio-temporal query $Q=(Q_s,Q_t)$ over a D-PARINET index and assuming that the query time interval $Q_t=[t_s,t_e]$ is inside the lifespan of a component index PARINET_i, the number of disk accesses needed to answer this query is:

$$DA_Q = \sum_{p \cap Q_s} \left[\left\lceil \log N_p^i \right\rceil + N_p^i |Q_t| \left(\frac{1}{BS_i} + \frac{1}{BS_d}\right) \int_{t_s}^{t_e} \rho_p^i(t) \cdot dt \right] \quad (3.7)$$

N_p^i and ρ_p^i have the same significance as N_p and ρ_p are measured relatively to the component index PARINET$_i$.

Assuming that the temporal interval of a query can be situated with equal probability at any instant within the lifespan of the D-PARINET, we can use an average temporal data distribution instead of the local temporal distribution $\rho_p^i(t)$. The average temporal distribution is $\tilde{\rho}_p^i = \frac{\int_{t_{i-1}}^{t_i} \rho_p^i(t) \cdot dt}{|T_i|} = \frac{1}{|T_i|}$, where $|T_i| = t_i - t_{i-1}$ is the lifespan of the local index. This assumption does not limit the generality of the cost model. In the case of non-uniform temporal distribution of the queries, $\tilde{\rho}_p^i$ can be estimated based on the specific distributions. Then, Formula (3.7) can be rewritten as follows

$$DA_Q = \sum_{p \cap Q_s} \left[\left\lceil \log N_p^i \right\rceil + N_p^i \left(\frac{1}{BS_i} + \frac{1}{BS_d}\right) \frac{|Q_t|}{|T_i|} \right] \quad (3.8)$$

where $|Q_t| = t_e - t_s$

To simplify the formulas, we use the notation $\sigma_p^{Q_t} = N_p^i \left(\frac{1}{BS_i} + \frac{1}{BS_d}\right) \frac{|Q_t|}{|T_i|}$ representing the (average) temporal selectivity of a query in a partition. Using this notation the number of disk accesses for given a query is $DA_Q = \sum_{p \cap Q_s} \left(\left\lceil \log N_p^i \right\rceil + \sigma_p^{Q_t} \right)$. Also, given a query load $Q_L = \{Q_1, Q_2, \ldots Q_k\}$ the quality indicator of PARINET$_i$ is $QI_{Q_L}^i = \sum_{j=1}^{k} DA_{Q_j}^i$.

D-PARINET uses the following parameter definitions that measure the quality of a component index PARINET_i in a T-PARINET.

Definition 5.1 The *global cumulated degradation* of a component index PARINET_i w.r.t. a query load Q_L is defined as: $GCD^i_{Q_L} = \frac{QI^i_{Q_L} - QI^{i\text{-optimal}}_{Q_L}}{QI^{i\text{-optimal}}_{Q_L}}$, where $QI^{i\text{-optimal}}_{Q_L}$ represents the quality indicator of the optimal configuration of PARINET_i w.r.t. the cost model.

Definition 5.2 The *local cumulated degradation* of a component index PARINET_i w.r.t. a query load Q_L is defined as: $LCD^i_{Q_L} = \frac{stdev(DA^i_{Q_L}) - stdev(DA^{i\text{-optimal}}_{Q_L})}{stdev(DA^{i\text{-optimal}}_{Q_L})}$, where

$stdev(DA^i_{Q_L}) = \sqrt{\frac{1}{k} \cdot \sum_{j=1}^{k} (DA^i_{Q_j} - \overline{DA}^i_{Q_L})^2}$ and $\overline{DA}^i_{Q_L} = \frac{\sum_{j=1}^{k} DA^i_{Q_j}}{k}$.

Definition 5.3 The *load factor* of a component index PARINET_i having m partitions is defined as: $LF^i = \frac{\sum_{j=1}^{m} N^i_j}{m \cdot fan^{\lceil \tilde{h} \rceil}}$, where fan is the fanout of the B^+-trees indexes and $\tilde{h} = \frac{\sum_{j=1}^{m} \lceil \log N^i_j \rceil}{m}$ is the average height of the trees.

The defined parameters indicate the expected degradation of an index (e.g., the index of the future trajectories or a set of future indices). GCD measures the percentage of query performance loss of an index compared with the optimal index configuration w.r.t. the cost model. This parameter offers a global view of the performance loss, since it considers the aggregated cost over the query load. LCD is a parameter intended to measure the unbalance in the query cost across the indexed data space. Recall that PARINET partitions the data into several clusters that are balanced w.r.t. the amount of

data in each cluster. Due to the difference between the expected and the real distribution of the data, the partition weights can be unbalanced. LCD indicates a local degradation, since it measures the unbalance in the query cost.

The first two parameters measure a degradation that is already presented in the index. LF indicates an imminent degradation of the current index. A drop in the query performance is also caused by an increase in the tree height of the B^+-trees since the data continuously accumulate. When LF approaches 1, the average height of the B^+-trees is expected to augment to include the new entries. This will lead to an increase in the query cost. However, the overhead can be avoided by an earlier "closing" of the index and triggering the creation of a new index.

In addition to monitoring these parameters, the construction of a new component index in a D-PARINET can be automatically triggered whenever they exceed certain predefined threshold values of prediction window. Assuming the maximum time interval of the queries $\left|Q_t^{\max}\right|$, the lifespan of a D-PARINET index should verify the following inequality in order to have searchable temporal predicates in the queries.

$$\left|T_i\right| > \left|Q_t^{\max}\right| \cdot \frac{BS_i + BS_d}{BS_i}$$

Each partition in a PARINET index uses a B^+-tree to index the temporal dimension of the data. The query optimizer will make use of these indexes only if the estimated number of disk accesses for an index based search is lower than the number of disk accesses in a full partition data scan. Therefore, building a B^+-tree index in each partition is useful only if, $\lceil \log N_p^i \rceil + N_p^i (\frac{1}{BS_i} + \frac{1}{BS_d}) \frac{|Q_t|}{|T_i|} < \frac{N_p^i}{BS_d}$. Since $\lceil \log N_p^i \rceil << \frac{N_p^i}{BS_d}$, the above inequality leads to above formula.

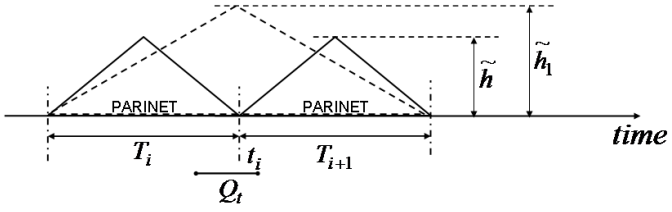

Figure 3.4 Component Index PARINET$_i$

Assuming the maximum time interval of the queries $|Q_t^{\max}|$, the creation of a new component index indicated by the *load factor* of the current index is beneficial only if the lifespan of the current index verifies the following inequality: $|T_i| > |Q_t^{\max}| \cdot \tilde{h}$, where $\tilde{h}$ is computed by $\tilde{h} = \frac{\sum_{j=1}^{m} \lceil \log N_j^i \rceil}{m}$.

Considering the component index PARINET$_i$ in Figure 3.4, which has the lifespan T_i, at time t_i, LF^i is close to 1 indicating thus an imminent increase in the average index height. Therefore, a new component index is built for the data arriving after t_i. To keep the formulas tractable, we consider that PARINET$_{i+1}$ has the same configuration as PARINET$_i$. Creating a new index will keep the cost of the queries situated in the interval T_i from augmenting, except for the queries having Q_t overlapped with t_i. For these queries the cost will increase since we need to visit two PARINET indices to evaluate them. Globally, this is not too penalizing if the percentage of queries in T_i that intersect t_i is low. This percentage can be estimated as $\frac{|Q_t|}{|T_i|}$, conform with the temporal uniformity assumption considered earlier. Hence, the average cost of a query in T_i is equal to

$DA_Q = \sum_{p \cap Q_s} \left[(1 + \frac{|Q_t|}{|T_i|})\tilde{h} + \sigma_p^{Q_t} \right]$. This is the cost of traversing the index is increased with the probability that a query overlaps two indexes.

Another option would be to continue indexing the data arriving after T_i with the same index regardless of the increase of the average index height. In this case the average cost of a query is $DA_Q^1 = \sum_{p \cap Q_s} (\tilde{h}_1 + \sigma_p^{Q_t})$, where $\tilde{h}_1 = \tilde{h} + 1$. It is then beneficial to create a new index at t_i only if $\frac{|Q_t|}{|T_i|}\tilde{h} < 1$.

3.5.5 Temporal Partitioning Algorithm

In the previous section, the cost model of PARINET and T-PARINET is adopted in the context of D-PARINET, i.e., for continuously indexing future trajectory datasets. The minimum lifespan of a index in D-PARINET based on the maximum expected query time interval and the maximum lifespan of the index depends on the prediction quality degradation factor. In addition, we used some quality factors, i.e., GCD, LCD and LF, to measure the performance of a component index. By simply monitoring the evolution in time of these parameters in the current index of a D-PARINET, an on-line tuning process can automatically decide when is an appropriate moment to trigger the construction of a new current index. Note that these parameters only take into account the query cost since the updates can be performed at approximately constant cost regardless of the index configuration. Also, the instantiation of a new component index is considered to have a small cost and is neglected.

The on-line tuning process of T-PARINET is based on a simple algorithm (Algorithm 2 in APENDIX C) which can be extended for D-PARINET. The process computes and continuously updates a few global statistics on the current index. Then, it verifies based on these statistics if the index quality indicators are situated within some predefined limits. A new (component) index is created when one of the parameters exceeds a threshold value. The configuration of the new index is determined based on an expected data distribution and density for that timespan. Note that the creation of a new component index can be triggered only if the lifespan of the current index is greater than a predefined value based on prediction quality factor. Afterwards, the tuning process continues monitoring the new index in D-PARINET.

Given a component index $PARINET_i$, $Stat_i$ includes the following information on the index structure: the current number of trajectory units for each road and the total current number of trajectory units in each partition of $PARINET_i$. This is sufficient to compute GCD^i, LCD^i and LF^i.

Note that the tuning process used here, can be improved in certain cases. For instance, if good prediction models for the traffic are available, an important change in the distribution and the density of the data flow can be foreseen and the construction of a new current index can be triggered in advance, i.e., without having to wait for a degradation of the current index.

CHAPTER 4

MODELING VEHICLE DISTRIBUTION ON ROUTE

4.1 Data Gathering

We have collected data from State of New Jersey Department of Transportation. NJDOT monitors more than 3000 sites for continuous traffic count data recorded at hourly for 24 hours. For our work, we have selected Route I-80 as it is a busy urban freeway and data were available for most of the stations. We can safely assume that any change between two stations on a freeway can only happen at the exits between the stations. We use available freeway map and station's longitude, latitude to identify the exits. Whenever there is more than one exit between two stations, we consider them as one and assume them to be equally likely for a vehicle to exit and enter the freeway.

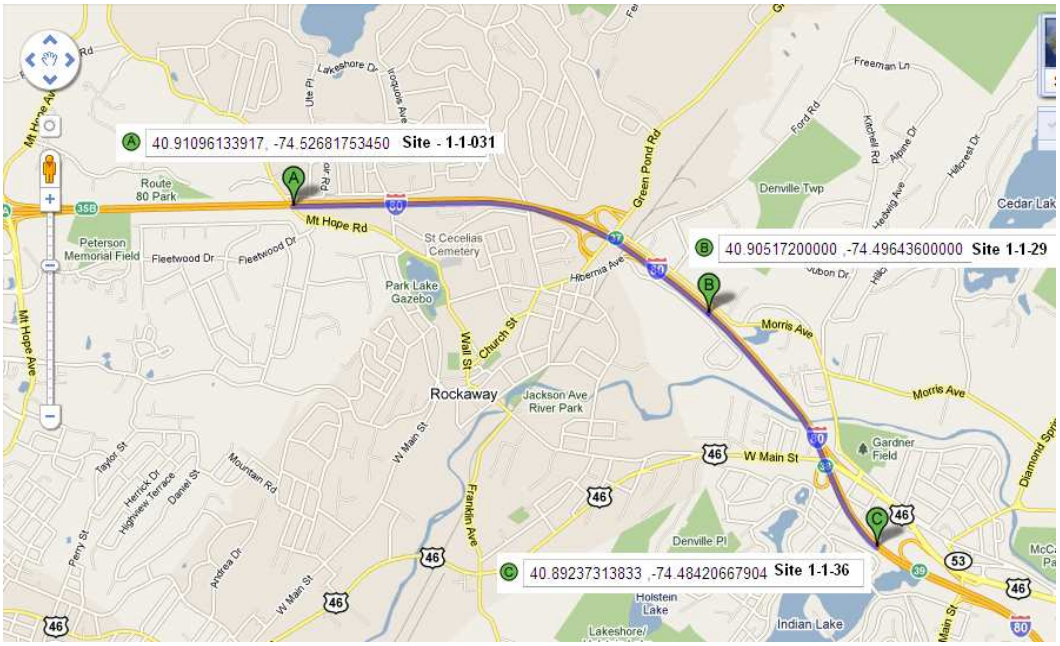

Figure 4.1 Position of selected loop detector stations on Google map

We have selected stations A(1-1-31e), B(1-1-29),C(1-1-36) for primary analysis as these stations are located on I-80 East and have only one exit between two locations and we can model the change of traffic at that exit.

4.2 Pre-processing of data

The data is recorded hourly and saved in a daily file. Data was processed for the year 2009 and inserted the data into an oracle database for the ease of analysis. This was a time consuming process as the format of data was not 100% consistent for each station. For learning purpose, weekday's morning from 5:00 AM to evening 8:00PM is considered because this is the time window when the roads are busy and the traffic reflects changes due to daily commute to work. Hourly count data is processed for the three selected stations and plotted the average hourly count.

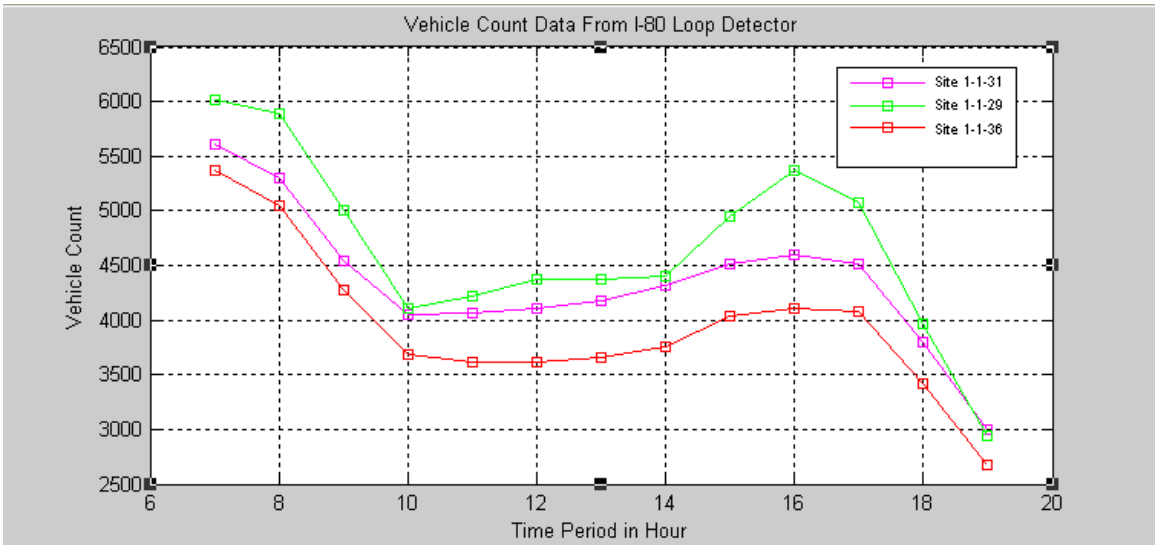

Figure 4.2 Vehicle count data at three sites on I-80

Sometimes the sensors at these stations were not in working conditions with zero counts which is very unlikely at the selected time of the day. Available data is interpolated in order to avoid undesired skew in the data.

Table 4.1 Change between two sites during the day

Time	Site 1-1-031	Change at Exit 37	Site 1-1-029	Change at Exit 38	Site 1-1-036
5:00	3876	-589	3287	200	3487
6:00	5805	-9	5796	-302	5494
7:00	5613	400	6013	-637	5376
8:00	5293	593	5886	-837	5049
9:00	4549	453	5002	-720	4282
10:00	4044	67	4111	-419	3692
11:00	4063	156	4219	-603	3616
12:00	4101	270	4371	-757	3614
13:00	4181	189	4370	-715	3655
14:00	4313	91	4404	-646	3758
15:00	4516	430	4946	-915	4031
16:00	4596	770	5366	-1261	4105
17:00	4509	564	5073	-996	4077
18:00	3804	159	3963	-547	3416
19:00	3002	-63	2939	-268	2671
20:00	2527	-64	2463	-249	2214

Source: Calculated from Hourly Freeway loop Detector Data, NJDOT

4.3 Average Vehicle Speed Data

We also collected the monthly and yearly average speed data by vehicle type from NJDOT at loop detector stations for demand modeling in SUMO which is explained later in Chapter 5 in detail.

4.4 Modeling the data

4.4.1 Homogeneous Poisson Process

Let N(t), for $t \in \{1, \ldots, T\}$, be the observed count at time t for any of the time-dependent counting processes, such as the freeway traffic one hour aggregate count process. The most common probabilistic model for count data is the Poisson distribution, whose probability mass function is given by $P(N; \lambda) = e^{-\lambda}\lambda^{N}/N!$ and N = 0, 1,2,3 etc.

where λ represents the rate, or average number of occurrences in a fixed time interval. When λ is a function of time, i.e. λ(t), (1) becomes a non-homogeneous Poisson distribution, in which the degree of heterogeneity depends on the function λ(t).Given that the observations at different stations are Poisson, we can easily estimate the λ of a time in a day by averaging the observed count at each station, which will give the maximum likelihood estimate. The following figure shows the vehicle count data when fitted into a Poisson distribution.

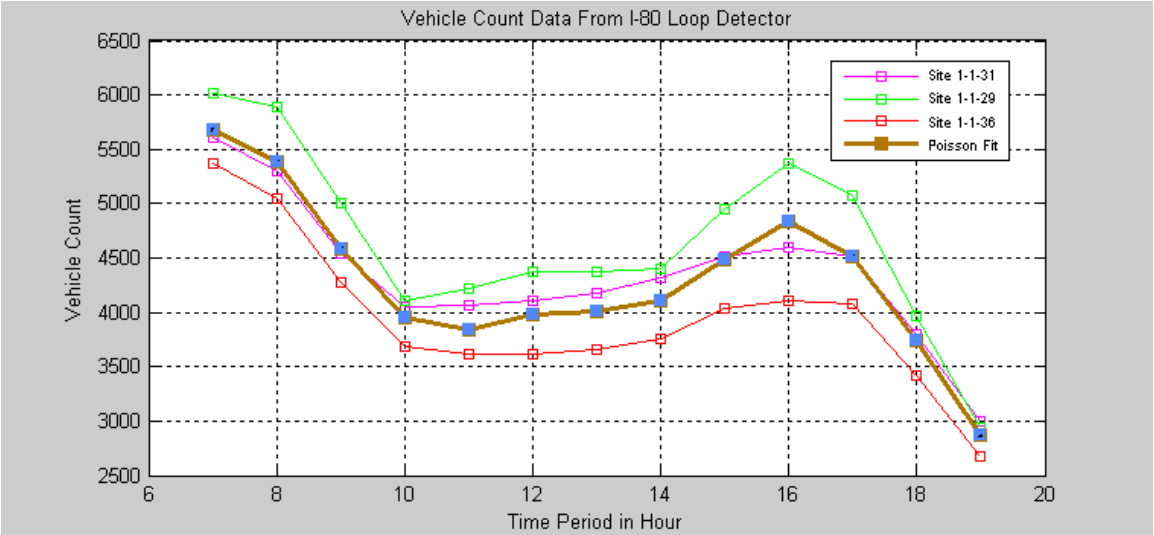

Figure 4.3 Poisson Fit of vehicle count data

4.4.2 Baseline Model and Limitation

If the modeling of N(t)L1 is derived from Alexander Ihler, Jon Hutchins & Padhraic Smyth(2006) where λ(t) is decomposed into day and time factor to represent the periodical nature of the data in their paper Learning to Detect Events with Markov-Modulated Poisson Processes , who used the same highway loop sensor data for modeling existence of events in the traffic. Proposed Poisson coefficient,

$$\lambda(t) = \lambda_0 \, \delta_{d(t)} \, \eta_{d(t),h(t)}$$

where d(t) takes on values {1, . . . , 7} and indicates the day on which time t falls (so that Sunday = 1, Monday = 2, and so forth), and h(t) indicates the interval (one hour) in which time t falls. We can ensure that the values λ_0, δ, and η are easily interpretable. λ_0 is the average rate of the Poisson process over a full week, δj is the day effect, or the relative change for day j (so that, for example, Sundays have a lower rate than Mondays), and $\eta_{j,i}$ is the relative change in time period i given day j (the time of day effect).While, implementing this non-homogeneous Poisson process for modeling the route and vehicle distribution in SUMO would result in more accurate prediction, it requires substantial customization in SUMO's source code. In reality, the poissonfit vehicle count data shows at most 10% deviation from the observed count on average.

CHAPTER 5

TRAFFIC SIMULATION MODEL IN SUMO

5.1 Traffic Model in SUMO

SUMO is a microscopic, space continuous and time discrete traffic simulation. In traffic research three or four classes of models are distinguished according to the level of detail of the simulation. In 'Macroscopic' models traffic flow is the basic entity, vehicle density and average speed is considered for modeling. 'Microscopic' models, simulate the movement of every single vehicle on the street – mostly assuming that the behavior of the vehicle depends on both, the vehicle's physical abilities to move and the driver's controlling behavior (Krauß 1998; Janz 1998). 'Mesoscopic' models, also sometimes called 'nanomodels' regard single vehicles like microscopic simulations, but submodels are included, that describe the engine's rotation speed in relation to the vehicle's speed or the driver's preferred gear switching actions, for instance. This allows more detailed computations of the emissions produced by the vehicle compared to a simple microscopic simulation (Diekamp 1995; Schreckenberg and Wolf 1998; Helbig et al. 2001). However, submicroscopic models require large computation times. This restrains the size of the networks to be simulated.

We used SUMO to simulate the traffic flow on a section of New Jersey I-80 freeway. We use the hourly average vehicle count and average velocity measured at loop detector sites on I-80 and the data is provided by NJDOT.

5.1.1 SUMO Model and Its Application on Freeway Data

SUMO is a multi-modal i.e. it not only simulates cars, but also simulates other form of transportation including commercial vehicles. The traffic flow is simulated microscopically. This means, that every vehicle that moves within the simulated network is modeled individually and has a certain place and speed. In every time step which has duration of 1sec by default but can be configured as needed, vehicle speed and position is calculated based on the following well known models.

1. Car following model
2. Lane change model

The simulation of street vehicles is time-discrete and space-continuous. As our car-driver model is continuous - as the majority of car-driver models are - we decided to use this approach. When simulating traffic, the street attributes, such as maximum velocity and right of way rules are regarded.

5.1.2 Classical Car Following Model

Positions and velocities of all vehicles are denoted by xi and vi respectively, where the index i rises in downstream direction. Deriving a car following theory can obviously always start from the quite reasonable assumption that a change of the velocity is only performed, if the momentary velocity does not coincide with some desired velocity Vdes, which is determined by safety considerations, legal restrictions and so on. The simplest dynamics that describes how a driver tries to approach the desired velocity is that of a relaxation on some time scale τ :

$$\frac{dv_i(t)}{dt} = \frac{V_{des} - v_i}{\tau}$$

Virtually all car following theories can be traced back to this simple idea. Usually, however, this dynamical relation is not interpreted as a relaxation process, but as a stimulus.

5.1.3 Lane Changes

The topic of lane changes has been addressed much less in the literature than that of car–following. Sparmann performed a partly empirical and partly theoretical analysis of lane changing on two–lane–freeways, while Leutzbach and Busch performed this kind of analysis for three–lane freeways. A model for the structure of lane–changing decisions in urban driving situations, where traffic signals, obstructions and heavy vehicles all exert an influence, has been developed by Gipps. The generalization of the minimalistic rules used in cellular automaton models to multilane traffic was done by Latour, Rickert et al., Wagner et al., and Chowdhury et al.

5.1.4 Using a Probabilistic Data Distribution Model in SUMO

To the best knowledge of this thesis writer, presently a custom data distribution model cannot be fed into SUMO directly. But in can be integrated indirectly in the demand modeling step. For example, it is explained in chapter 4 that the highway vehicle count data follows Poisson distribution model but this information cannot be fed into SUMO directly while creating the demand. This has been identified as a potential area of future work.

CHAPTER 6

EXPERIMENTAL DETAILS

6.1 SUMO features

In the current version – 0.12.3 – SUMO contains the following features including network generation and routes import, demand modeling and simulation.

6.1.1 Simulation

Using available custom parameter while running a simulation on SUMO, it can ensure and provide

1. Collision free vehicle movement
2. Different vehicle types
3. Multi-lane streets with lane changing
4. Junction-based right-of-way rules
5. Hierarchy of junction types
6. A fast openGL graphical user interface
7. Manages networks with several edges (streets)
8. Fast execution speed (up to 100.000 vehicle updates/s on a 1GHz machine)
9. Interoperability with other application on run time using TraCI
10. Network-wide, edge-based, vehicle-based, and detector-based outputs

6.1.2 Network

SUMO supports importing different network file (node, edge and connection) from different sources and can determine missing values as needed for simulation using heuristics. Many network formats (VISUM, Vissim, Shapefiles, OSM, Tiger, RoboCup, XML-Descriptions) may be imported.

6.1.3 Routing

While SUMO does not store vehicle history, i.e the vehicle is unique in every simulation. SUMO supports

1. Microscopic routes - each vehicle has an own routes. One can assign probabilistic distribution for each route a vehicle can use.
2. Demand modeling – It has two steps, first estimating the traffic demand and then estimating the dynamic user equilibrium.

6.2 SUMO network

6.2.1 Generating network diagram from map

The location information (latitude, longitude) is extracted for the selected loop detector sites .Three sites were selected on east bound freeway and there is only one exit in between every two sites. We selected these three sites so that we can easily estimate the exit and entry through on-ramp and off-ramp edges, when required. We used the open Street Map rest API (**http://api.openstreetmap.org/api/0.6/map?bbox=-74.52681753,40.91096134, -74.48420668,40.89237314**) to generate the network diagram as shown below in osm format. As the network generated for the given latitude-

longitude box boundary included local street information, another third party program "Osmosis" is used to extract only the freeway graph from the network in xml format.

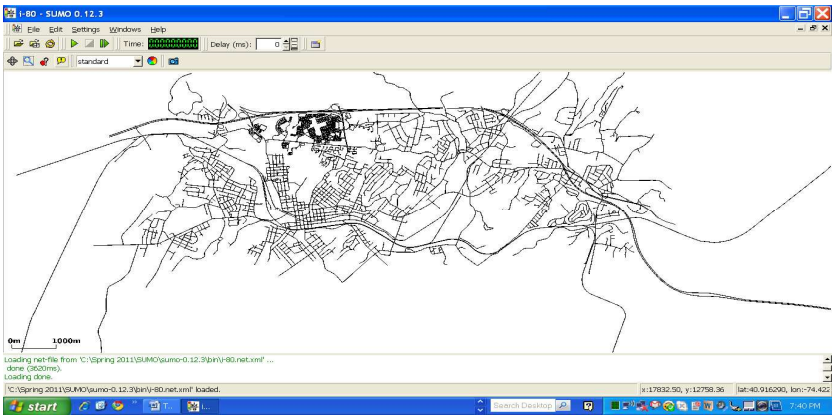

Figure 6.1 Network diagram of I-80 selected area

6.2.2 Create SUMO compatible network

SUMO provides a command line tool called NETCONVERT which is capable of using osm format network and converting into SUMO format. A full list of command line parameters used for our case is given in Apendix1.

In graph theory, a flow network is a directed graph where each edge has a capacity and each edge receives a flow. The amount of flow on an edge cannot exceed the capacity of the edge. Often in Operations Research, a directed graph is called a network, the vertices are called nodes and the edges are called arcs. A flow must satisfy the restriction that the amount of flow into a node equals the amount of flow out of it, except when it is a source, which has more outgoing flow, or sink, which has more incoming flow. A network can be used to model traffic in a road system, fluids in pipes, currents in an electrical circuit, or anything similar in which something travels through a network of nodes. G(V,E) is a finite directed graph in which every edge (u,v)ε E has a non-negative, real-valued capacity c (u,v). If (u,v) NOT ε E, it is assumed that c (u,v) = 0.

Two vertices are identified , a source, which "produces" flow, s and a sink, which "consumes" flow, t. e As the data on a multi-lane freeway is simulated , SUMO defines lanes inside an edge.The output is generated as XML file, so it is both human and machine readable.

6.3 Route Generation for I-80 Highway

Beside the static part – the network – the simulation consists of moving vehicles. With the increase of the quality of simulations, the need to model a populations' mobility has increased as well. In such cases, vehicles are not spread statistically over the network; instead a single person's daily plan consisting of routes with certain departure times is used. While data needed to describe the departure times and a route's origin and destination are given, the routes themselves must be computed. To avoid online-computation of these during the simulation, this computation is done using a separate module, the SUMO-ROUTER. This module reads the departure times, origins and destinations for a set of virtual humans that will be simulated, then computes the routes through the network itself using the well-known Dijkstra routing algorithm (Dijkstra 1959). As the speed on the streets changes with the traffic amount and therefore the computation of routes using a network where the traffic is not yet known does not regard the real-world situation, the routing will be done using the Dynamic User Equilibrium approach developed by Christian Gawron (Gawron 1998) where routing and simulation are repeated several times to achieve a real-world behavior of drivers. Furthermore, the router supports dynamic network load, the fact that the load on a edge depends on the time of day is also regarded. In order to estimate the demand of traffic i.e. route/flow/trip

generation, we followed two different procedures for two different datasets. The first one was for I-80 highway vehicle count data; we fed the simulator with the hourly averaged data for a given site from a loop detector. The second one was to use a cities demographic data to generate the daily demand. Here, we did not have access to real life data but used Wikipedia to create a realistic demand. This step is done offline and very time consuming depending on the network size. This activity data is used to create the vehicle and route distribution.

6.4 Simulation Output

Simulation is run using the network configuration, routing configuration and loop detector configuration files. A schematic view of the workflow is given in the appendix.

SUMO provides various options for generating output from the simulation; we created a dump output file which gives us the complete network state at every time period. Time period is configurable with a default value of 1 sec. The next output is created with vehicle state giving us each vehicle's position by time period. These outputs were parsed to insert the data into oracle database and indexed for future queries.

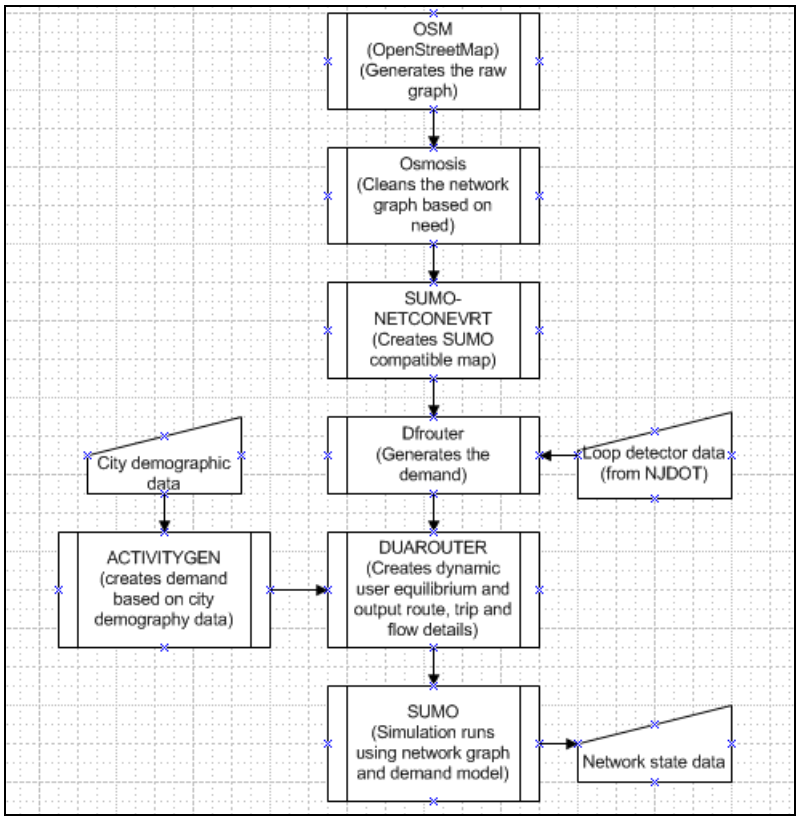

Figure 6.2 A schematic diagram of simulation experiment steps

CHAPTER 7

D-PARINET

7.1 Continuous Indexing of Incoming Trajectory Data

PARINET can be used to index historical datasets. In this case, the data are known in advance. Therefore, PARINET can be automatically tuned for near-optimal performance given a dataset and an expected query load. However, even if the data is historical, it does not mean that the queries are known in advance. Moreover, the queries at a given time may differ a lot (different users may pose totally different queries), and the queries may change across time (during the day or week or month). In such cases, the index itself should be able to handle very different queries at the same time and should be able to adapt to changes over time. We are interested in a robust index structures whose performance that does not degrade much with reasonable variations between the expected and actual query sizes.

The extended version of PARINET, i.e., T-PARINET, is intended to efficiently and continuously index trajectory data flows for querying history data. The concept of T-PARINET is used to build D-PARINET with interpolated future trajectories in order to answer predictive queries.

7.2 D-PARINET Cost model

In Section 3.3.2, we presented a cost model that estimates the number of disk accesses for a query load and a given index configuration. The tests show that the execution time of a query usually depends on the number of disk accesses. Therefore, the cost model can

estimate the performance of our access method for a given configuration, without effectively constructing the index. This is very important considering that the index creation is costly, i.e., it is not an option to actually test all the possible configurations in order to choose the best one. If the cost model is accurate, we can automatically find and materialize a good configuration among all the possible ones. Moreover, the cost model can be further employed in the context of continuously indexing MOs trajectories in order to optimize the index evolution in time. In this case, its role is to permit monitoring the efficiency of the current index and to compare it with a near-optimal index configuration.

In this section, we experimentally evaluate the proposed cost model. We also calculated the number of disk accesses as defined in Section 3.3.2. Practically, we implemented a program that takes as input the network graph with the data distribution for each road, a query load and a number of partitions, and outputs an estimated number of disk accesses. We consider a uniform temporal distribution of the data, which is a good approximation for the generated datasets. For non uniform data distribution, the real temporal distribution must be used in order to obtain a good estimation of disk accesses.

Another observation is that the cost model is more accurate for the larger datasets and also offers good estimations for the smaller datasets when the number of partitions is smaller or equal to 200. The reason the cost model is less accurate for small datasets and large number of partitions lays in the way the data partitioning is implemented under the Oracle Server. Each partition of a partitioned table has allocated a minimum of eight data pages regardless of the amount of data in that partition. For small datasets distributed over a large number of partitions, the partitions become under-occupied, i.e., they contain

less than eight pages of data. This will lead to an increase in the number of disk accesses to answer a query. The cost model can be easily adapted to take into account this specific case. Nevertheless, for the sake of generality, we used the cost model as initially proposed in Section 3.3.2.

In conclusion, the experiments show that the cost model is good enough to be used for tuning the PARINET index. Two types of queries are tested, 2D and path. For each type of query and for each map, three scripts are generated, each script containing queries of fixed size. For the 2D queries, a 2D square window is a randomly generated over a time interval. The intervals have the same relative size in all the dimensions. Then, the query is transformed and the final script is generated. For the path queries, some trajectories are randomly selected from the dataset and used to generate the spatial interval of the queries. A smaller spatial window is chosen due to the large number of roads in a network. The temporal interval is randomly chosen within the temporal interval of the dataset.

For a given query set, dataset and index configuration, the average time per query and the average number of disk accesses per query are measured. Similarly, given a large batch of updates that need to be executed, the average time and the average number of I/Os per one thousand processed updates are measured. The default page size in Oracle, i.e., 8KB is used. The resulted fan-out is 340 for the B+-tree index. Oracle logically implements the R-tree as a tree and physically using tables inside the database. Hence, the fanout does not depend on the page size. Oracle uses a LRU buffer cache. The size of the buffer cache is set to 32MB, which allows for good performances of the tested indexes (e.g., the minimum allowed size of the buffer cache under Oracle 11g is 8MB).

In all the tests that measure the query performance, the cache is emptied between each query run to limit the influence of the cache on query processing evaluations. In the tests that measure the update performance the buffer cache is cleared and commit changes were done after each 32 thousand processed updates. There is no intervention on the cache memory for the tests that measure the index throughput.

Due to incompatibility of SUMO format and METIS required format, the data is from SUMO is hand crafted to fit the previously created partition used by PARINET and T-PARINET.

7.3 Prediction using D-PARINET

The projected data for I-80 highway simulated in SUMO is compared with the observed data available from loop detector. We initialized SUMO with the hourly average count data and run the simulation for five minutes for every hour in a day starting 5 AM to 8PM. Then the data is interpolated refor the 24 hour span and fed into D-PARINET. In this case, the index structure is built in advance based on anticipated values for the data distribution and data density and the query size. Hence, the index should feature good robustness with the combined variation of both the data size and the query size. Clearly the prediction performance could be better if we could configure the simulation model based on history.

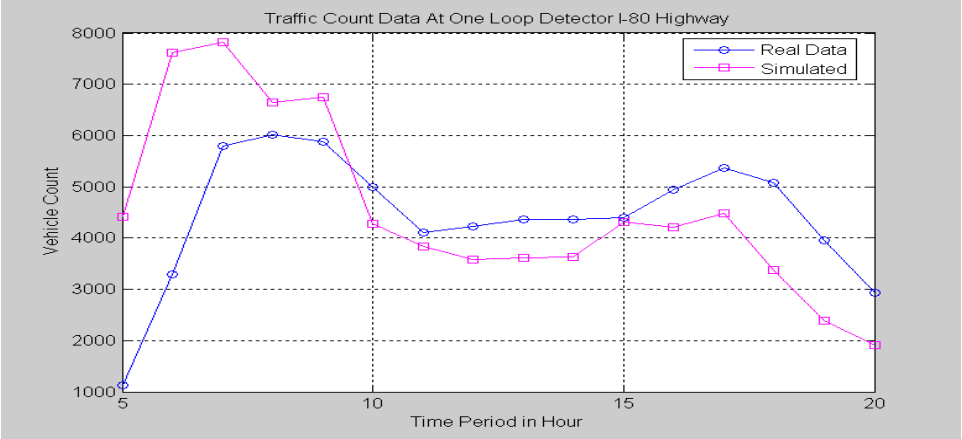

Figure 7.3 Prediction using simulated data

7.4 Conclusion and Future Work

While processing a big and complex network for city of Oldenburg with more than 10000 nodes and more than 31000 segments, building the partition based on data distribution become a challenge. The network data is created in SUMO compatible format and the partitioning software METIS expects a completely different format which requires huge amount of programming and testing time. Also the simulation software SUMO does not remember the vehicle id, a new vehicle is simulated. This prevents us interpolating data/route for a single vehicle without changing the source code for SUMO. Also, the distribution is used in SUMO is not configurable, which is a big factor in the prediction quality. These issues have not been addressed until now but it is under-work in-order to use large amount of synthetic data effectively to prove the robustness of proposed indexing method for D-PARINET. In addition to that, a more detail and thorough testing is required for D-PARINET for the SUMO generated data.

APPENDIX A

EXAMPLE CODE SNIPPET FOR GRAPH PARTITIONAING INPUT FILE

We tried to use available open source softwares for creating network, demand modeling, activity generation and graph partitioning as much as we can rather than building them from scratch. These softwares often require input in different formats and we wrote custom java program to create input files and parse output. The work was not trivial and we plan to use this as a base for developing parsing software in future.

```
public static void main(String argv[]) {

 try {
 File file = new File("C:\\Spring 2011\\SUMO\\sumo-0.12.3\\bin\\oldenburg.xml");
 DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
 DocumentBuilder db = dbf.newDocumentBuilder();
 Document doc = db.parse(file);
 doc.getDocumentElement().normalize();
 System.out.println("Root element " + doc.getDocumentElement().getNodeName());
 NodeList nodeLst = doc.getElementsByTagName("edge");
 System.out.println("Information of all edges");
 int num = 10;
 for (int s = 0; s < nodeLst.getLength(); s++) {
   Node fstNode = nodeLst.item(s);
   if (fstNode.getNodeType() == Node.ELEMENT_NODE) {
     Element fstElmnt = (Element) fstNode;
     String id = fstElmnt.getAttributeNode("id").getNodeValue();
     String types = fstElmnt.getAttributeNode("type").getNodeValue();
     String function = fstElmnt.getAttributeNode("function").getNodeValue();
     if (function.equalsIgnoreCase("normal") && (!types.equals("highway.unclassified") ) )
     {   if( num==10)
         {
         num=100;
         System.out.println("<street edge='" + id + "' population='"  + num + "' workPosition='10' />");
         }
         else
         {num=10;
```

Figure A.1 An example code for parsing city network data into SUMO format

APPENDIX B

DEMAND MODELING IN SUMO

Here's the sample activity statistics input for demand modeling and route generation

```
<city>
    <general inhabitants="16000" households="6500" childrenAgeLimit="18" retirementAgeLimit="65" carRate="0.58"
        unemploymentRate="0.05" footDistanceLimit="500" incomingTraffic="500" outgoingTraffic="6000" />
    <parameters carPreference="0.70" meanTimePerKmInCity="360" freeTimeActivityRate="0.15"
        uniformRandomTraffic="0.20" departureVariation="120" />
    <population>
        <bracket beginAge="0" endAge="30" peopleNbr="30" />
    </population>
    <workHours>
        <opening hour="30600" proportion="0.30" />
        <closing hour="64800" proportion="0.60" />
    </workHours>
    <streets>
        <street edge="-100011684#0" population="100" workPosition="50" />
        <street edge="-100011684#1" population="10" workPosition="100" />
        <!-- more street>
    <street edge="9930089" population="10" workPosition="100" />
   </streets>
   <cityGates>
                <entrance edge="-100011684#0" pos="1" incoming="0.1" outgoing="0.9" />
                <entrance edge="9928081#2" pos="250" incoming="0.2" outgoing="0.8" />
   </cityGates>
   <schools>
                <school edge="9929046#4" pos="20" beginAge="0" endAge="6" capacity="200" opening="32400" closing="64800" />
                <school edge="9928126#4" pos="100" beginAge="3" endAge="12" capacity="150" opening="30600" closing="64800" />
   </schools>
</city>
```

Figure A.2 An example demand modeling for Oldenburg city

APPENDIX C

ALGORITHM

Here are the algorithms used for road partitioning an index tuning of D-PARINET.

Algorithm 1: Determining index partitioning

Input: Network graph $G = (V, E)$, trajectory dataset D, query load $Q_L = \{Q_1, Q_2, \ldots Q_k\}$
Output: Road Partitioning function $RP : E \rightarrow \{1,2,..., p\}$

1. Compute $W : E \rightarrow \mathrm{N}$ given G and D
2. Compute $L(G)$ of G
3. $QI_{Q_L}^{optimal} = \infty$
4. **for** $m = 1$ **to** $card(E)$ **do**
5. $RP_m \leftarrow METIS(L(G), m)$
6. $QI_{Q_L}^{m} = \sum_{i=1}^{k} DA_{Q_i}^{m}$
7. **if** $QI_{Q_L}^{optimal} > QI_{Q_L}^{m}$ **then**
8. $RP \leftarrow RP_m$
9. $QI_{Q_L}^{optimal} = QI_{Q_L}^{m}$
10. **return** RP

Figure A.3 Determine Index Partitioning

Algorithm 2: D-PARINET On-line Tuning(Adopted from T-PARINET)

Input: Road Partitioning function RP_i of the current index $\mathrm{PARINET_i}$, global statistics $Stat_i$ of $\mathrm{PARINET_i}$, query load $\mathrm{Q_L} = \{\mathrm{Q_1}, \ldots \mathrm{Q_k}\}$ and $\left|Q_t^{\max}\right|$, thresholds GCD^{th} , LCD^{th}

1. current index $ci = i$
2. **while** *true*
3. update $Stat_{ci}$
4. **if** $(\mathrm{t_{c+\delta}} - \mathrm{t_c}) > 2\left|Q_t^{\max}\right|$ **then**
   if $Stat_{ci}.GCD_{Q_L}^{ci} > GCD^{th}$
5. **or** $Stat_{ci}.LCD_{Q_L}^{ci} > LCD^{th}$
   or $Stat_{ci}.LF^{i} > 0.9$ **then**
6. Create new index $\mathrm{PARINET_{i+1}}$
7. Compute $Stat_{i+1}$
8. current index $ci = i + 1$

Figure A.4 D-PARINET Online Tuning

APPENDIX D

INTEROPERABILITY OF VARIOUS SOFTWARES USED

Following figure demonstrates the interoperability problem among the open source software used in the experiment.

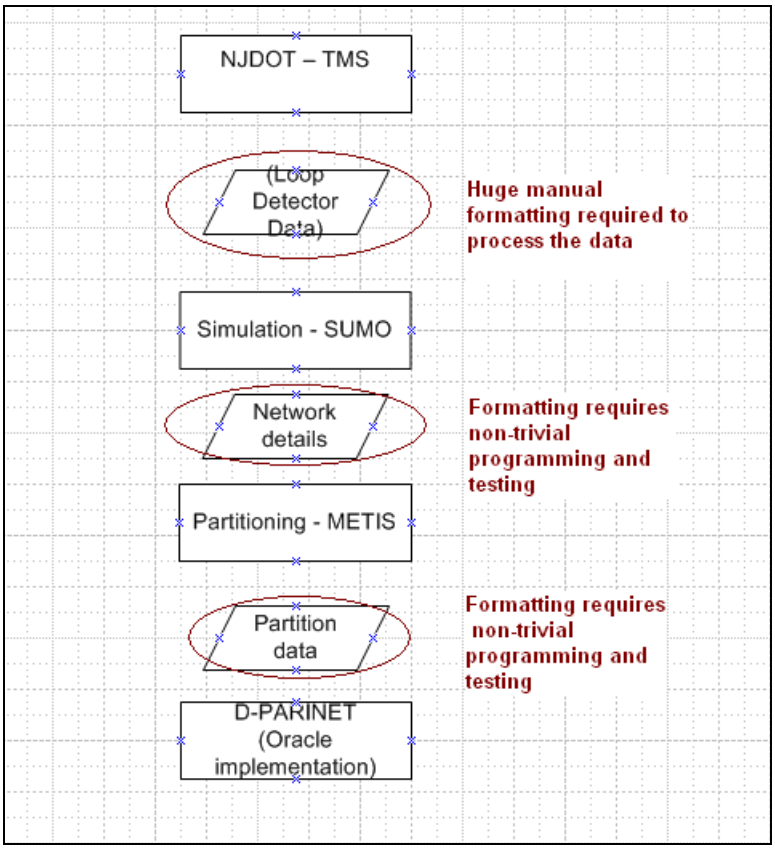

Figure A.5 A schematic diagram of software interoperability

REFERENCES

Sandu Popa, I., Zeitouni, K., Oria, V., Barth, D., Vial, S. *PARINET: A tunable access method for in-network trajectories*. Proc. ICDE: 177-188 (2010)

Ihler A., Hutchins J. & Smyth P. (2006). *Learning to Detect Events with Markov-Modulated Poisson Processes*

Behrisch M., Bonert M., Brockfeld E. *Event traffic forecast for metropolitan areas based on microscopic simulation,* Jan 2008

Almeida, V.T. de, Guting, R. *Indexing the Trajectories of Moving Objects in Networks. GeoInformatica 9(1):* 30–60 (2005)

Botea, V., Mallett, D., Nascimento, M.A., Sander, J. *PIST: An Efficient and Practical Indexing Technique for Historical Spatio-Temporal Point Data. GeoInformatica 12(2):* 143-168 (2008)

Brinkhoff, T. *A framework for generating network-based moving objects. GeoInformatica 6(2):* 153-180 (2002)

Chen, S., Ooi, B.C., Tan, K.L., Nascimento, M.A. *The ST2B-tree: A Self-Tunable Spatio-Temporal B+-tree Index for Moving Objects. Proc. ACM SIGMOD*, pp. 29–42 (2008)

Frentzos, E. *Indexing objects moving on fixed networks. Proc. SSTD,* 289–305 (2003)

Garey, M.R., Johnson, D.S. *Computers and Intractability; A Guide to the Theory of NP-Completeness, New York* (1990)

Güting, R.H., Almeida, V.T. de, Ding, Z. *Modeling and Querying Moving Objects in Networks. VLDB Journal 15(2)* 165-190 (2006)

Hadjieleftheriou, M., Kollios, G., Tsotras, J., Gunopulos, D. *Indexing spatiotemporal archives. VLDB Journal 15(2)* 143-164 (2006)

Karypis, G., Kumar, V. *A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM Journal on Scientific Computing, Vol. 20, No. 1, pp.* 359—392 (1999)

Kriegel, H.-P., Pötke, M., Seidl, T. *Managing Intervals Efficiently in Object-Relational Databases. Proc. VLDB* (2000)

METIS - *Family of Multilevel Partitioning Algorithms. [On-line]. Available: http://glaros.dtc.umn.edu/gkhome/views/metis*

Patel, J.M., Arbor, A., Chen, Y., Chakka, V.P. *STRIPES: an efficient index for predicted trajectories. Proc. ACM SIGMOD* 635–646 (2004)

Pelanis, M., Saltenis, S., Jensen, C.S. *Indexing the past, present, and anticipated future positions of moving objects. ACM Trans. Database Syst. 31(1):* 255-298 (2006)

Pfoser, D., Jensen, C.S. *Indexing of Network-Constrained Moving Objects. Proc. ACM-GIS,* 25–32 (2003)

Tao, Y., Papadias, D. *MV3R-Tree: A spatio-temporal access method for timestamp and interval queries. Proc. VLDB, pp.* 431-440 (2001)

Tao, Y., Papadias, D., Sun, J. *The TPR*-tree: an optimized spatio-temporal access method for predictive queries. Proc. VLDB* 790–801 (2003)

Saltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A. *Indexing the positions of continuously moving objects. Proc. ACM SIGMOD* 331–342 (2000)

Sun, J., Papadias, D., Liu, B. *Querying about the Past, the Present and the Future in Spatio-Temporal Databases. Proc. ICDE* 202–213 (2004)

Lin, D., Jensen, C.S., Ooi, B.C., Saltenis, S. *Efficient indexing of the historical, present, and future positions of moving objects. Proc. MDM,* 59–66 (2005)

Mokbel, M.F., Xiong, X., Aref, W.G. *SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. Proc. SIGMOD:* 623-634 (2004)

Prabhakar, S., Xia, Y., Kalashnikov, D.V., Aref, W.G., Hambrusch, S.E. *Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. IEEE Trans. on Computers,* 51(10) (2002)

Theodoridis, Y., Stefanakis, E., Sellis, T.K. *Efficient Cost Models for Spatial Queries Using R-Trees*. IEEE Trans. Knowl. Data Eng. 12(1): 19-32 (2000)

Jensen, C.S., Lin, D., Ooi, B.C. *Query and update efficient B+-tree based indexing of moving objects. Proc. VLDB*: 768–779 (2004)

Gipps P.G., *A model for the structure of lane changing decisions*, *Transpn.* Res.B 20B(5), 403–414 (1986)