

# Frequent Route Based Continuous Moving Object Location and Density Prediction on Road Networks

---

Manohar Kaul





UPPSALA  
UNIVERSITET

Teknisk- naturvetenskaplig fakultet  
UTH-enheten

Besöksadress:  
Ångströmlaboratoriet  
Lägerhyddsvägen 1  
Hus 4, Plan 0

Postadress:  
Box 536  
751 21 Uppsala

Telefon:  
018 – 471 30 03

Telefax:  
018 – 471 30 00

Hemsida:  
<http://www.teknat.uu.se/student>

## Abstract

# Frequent Route Based Continuous Moving Object Location and Density Prediction on Road Networks

*Manohar Kaul*

Emerging trends in urban mobility have accelerated the need for effective traffic management and prediction systems. Simultaneously, the widespread adoption of GPS-enabled mobile devices has opened radical new possibilities for such systems. Motivated by this development, this thesis proposes an end-to-end streaming approach for traffic management that encompasses a novel prediction model. The stream processing is achieved by a sliding window model.

In particular, the approach performs online 1) management of the current evolving trajectories, 2) incremental mining of closed frequent routes and 3) prediction of near-future locations of the moving objects based on the current object trajectories and historical frequent routes. The approach proposes storage of closed frequent routes and all possible turns a moving object can make at a junction, in a FP-tree like structure. This structure is created on-the-fly from the buffered contents of each constituent window of the trajectories stream and then used to determine probabilistic future locations of each moving object. It additionally calculates the densities of moving objects and parked objects for the entire road network. The prototype implements the approach as extensions to SCSQ - a data stream management system (DSMS) developed at UDBL. SCSQ is an extension of Amos II which is an extensible, main-memory OO DBMS. The solution utilizes SCSQ's stream manipulation and windowing capabilities coupled with Amos II's functionality to efficiently store, index and query frequent routes for prediction.

The approach is empirically evaluated on a large real-world data set of moving object trajectories, originating from a fleet of taxis, showing that detailed closed frequent routes can be efficiently discovered and used for prediction.

Handledare: Gyozo Gidofalvi  
Ämnesgranskare: Tore Risch  
Examinator: Anders Jansson  
IT 11 027  
Tryckt av: Reprocentralen ITC



# CONTENTS

---

1	INTRODUCTION	1
2	RELATED WORK	5
3	BACKGROUND TECHNOLOGIES	7
3.1	Data Stream Management System (DSMS)	7
3.2	Amos II and AmosQL	8
3.3	SCSQ	8
4	PRELIMINARIES AND DEFINITIONS	9
4.1	Road Network Based Routes of Moving Objects	9
4.2	Trip Trajectories of Moving Objects	10
4.3	Continuously Evolving Trajectories	11
4.4	Frequent Route Mining	11
4.5	Moving Object Location and Density Prediction	13
5	CLOSED FREQUENT ROUTE MINING IMPLEMENTATION	15
5.1	Design Background	15
5.2	Closedness Checks in trajectories	15
6	SYSTEM ARCHITECTURE	18
6.1	Map Matching	18
6.2	Tuples Stream in SCSQ	19
6.3	Mining Stream in SCSQ	19
6.4	Prediction Stream in SCSQ	19
6.5	Continuous Query (CQ) in SCSQ	20
7	PREDICTION MODEL	21
7.1	Running Example	22
7.2	Example trajectories and frequent routes	22
7.3	Closed frequent pattern tree creation	23
7.4	Core prediction algorithm	26
8	CONTINUOUS QUERIES IN SCSQ	32
8.1	Reading as stream	33
8.2	Closed Sequence Mining of Stream	33
8.3	Storing closed patterns	34
8.4	Extract the vehicle's latest trip	35
8.5	Find the mining window for trajectory	36
8.6	Predict	37
8.7	Predict moving object density	38
8.8	Predict parked object density	38
9	EXPERIMENTS	40
9.1	Real-World Data Set	40
9.2	Stream Processing Analysis	41
9.3	Scalability Analysis	41
9.4	Qualitative Pattern Analysis	42
9.5	Prediction Accuracy Analysis	43
10	CONCLUSIONS AND FUTURE WORK	46

I APPENDIX	47
A APPENDIX	48
A.1 Table of road segment's global average traversal time and support	48
BIBLIOGRAPHY	49

## LIST OF FIGURES

---

Figure 2.1	Taxonomy of Location prediction models . . . . .	5
Figure 4.1	Trip trajectories of a trajectory. . . . .	11
Figure 6.1	System Architecture . . . . .	18
Figure 7.1	An example of path prediction of moving object . . . . .	21
Figure 7.2	Sample trajectory . . . . .	22
Figure 7.3	Structure of nodes . . . . .	24
Figure 7.4	Stepwise insertion of patterns as branches of tree $\tau_c$ . . . . .	25
Figure 7.5	Complete prefix tree . . . . .	26
Figure 7.6	Complete prefix tree . . . . .	28
Figure 8.1	Functional Data Flow Diagram . . . . .	32
Figure 9.1	Mining analysis. . . . .	41
Figure 9.2	Scalability Analysis. . . . .	42
Figure 9.3	Qualitative pattern analysis. . . . .	43
Figure 9.4	Prediction accuracy analysis for individual moving objects averaged over stream windows. . . . .	44
Figure 9.5	Prediction Error analysis for road network density averaged over stream windows. . . . .	45

## LIST OF TABLES

---

Table 7.1	Probabilities calculated from patterns . . . . .	30
Table 7.2	Probabilities calculated from turn patterns . . . . .	31
Table 7.3	Probabilities for single recursion level . . . . .	31
Table A.1	Singleton road segments with corresponding global traversal time and support . . . . .	48

## LISTINGS

---

Listing 7.1	Sample trajectories on road network . . . . .	22
Listing 7.2	Sample frequent routes . . . . .	23
Listing 8.1	Reading in tuples as a stream in SCSQ . . . . .	33
Listing 8.2	Sample usage of <i>st_read_trajectory</i> in SCSQ . . . . .	33
Listing 8.3	Closed sequence mining of stream . . . . .	34
Listing 8.4	Sample usage of <i>st_clomine</i> in SCSQ . . . . .	34
Listing 8.5	Storing closed patterns . . . . .	34

Listing 8.6	Sample usage of <i>st_load_patterns</i> in SCSQ . . . . .	35
Listing 8.7	Extraction of vehicle trips . . . . .	35
Listing 8.8	Sample usage of <i>st_tracker</i> in SCSQ . . . . .	36
Listing 8.9	Finding mining window . . . . .	36
Listing 8.10	Sample usage of <i>get_mining_window_sql</i> in SCSQ . . . . .	37
Listing 8.11	Prediction . . . . .	37
Listing 8.12	Sample usage of <i>get_predictions</i> in SCSQ . . . . .	37
Listing 8.13	Prediction of future moving object density on road network	38
Listing 8.14	Sample usage of <i>get_moving_density</i> in SCSQ . . . . .	38
Listing 8.15	Prediction of future parked object density on road network	38
Listing 8.16	Sample usage of <i>get_parking_density</i> in SCSQ . . . . .	39

## ACRONYMS

---

## INTRODUCTION

---

**MOTIVATION:** The rapid growth of demand for transportation, and high levels of car dependency caused by the urban sprawl, have exceeded the slow increments in transportation infrastructure supply in many areas. This causes severe traffic congestion, with well-known negative effects such as increased fuel consumption, pollution, and carbon dioxide emissions; time and associated cost lost by motorists sitting in traffic jams; wear-and-tear on vehicles and infrastructure resulting from stop-and-go traffic; and, last but not the least, the inflicted stress and fatigue on motorists causing unnecessary accidents.

In dense urban areas, adding capacity through construction of new facilities is difficult due to lack of space and prohibitive costs. A more viable approach to cope with the congestion problem is to monitor traffic congestion, understand the causes of its formation and development, and use the aforementioned knowledge in traffic management systems and transportation planning to mitigate traffic congestion.

Early systems for traffic prediction and management have primarily used punctuated speed and flow measurements from fixed location sensors in conjunction with traffic models to tackle the prediction and management tasks. More recently, the wide-spread adoption of GPS-based on-board navigation systems and location-aware mobile devices have enabled radically new possibilities. This thesis aims to contribute to GPS-based traffic prediction and management systems.

In such systems, in order to improve the accuracy, it is common to use the trajectories of the moving objects [9],[1],[2] as follows. It is assumed that vehicles periodically submit their location to a central server. In turn the server extracts traffic/mobility patterns from the submitted locations. The extracted patterns, together with the current locations of the vehicles, are both used in short- and long-term traffic prediction, management and planning tasks. Additionally, the current and near-future traffic conditions are sent in real-time to the vehicles likely to be affected. It is a central requirement of such systems that the data collection, data analysis, and subsequent notifications are processed with online, near-realtime timeliness.

The detection of future movement patterns can furthermore be used to optimize the design of *location based services (LBS)*. The services offered to a moving user could not only be dependent on the actual position, but also on the estimated future location of the moving object.

Examples of typical queries a traffic logistics operator might pose can be as follows:

1. Given the current state of a vehicle's evolving trajectory and its historical trajectories, provide predicted future locations of this vehicle at exactly *five* minutes in the future and perform such a query every *ten* minutes.
2. What will the top five most dense road segments be at exactly ten minutes from now?
3. How many vehicles will still be in motion at five minutes from now?
4. How many vehicles will end their trips at five minutes from now?

Motivated by these needs, it becomes evident that the answer lies in a prediction model that can be formulated as a *continuous query (CQ)* as the knowledge of future locations changes with time and has to be constantly monitored.

Regular queries are run in an *ad-hoc* fashion where the user submits a query and gets back results on termination of the query's execution, whereas continuous queries (CQ) are started once, and they keep executing the query at set time intervals to return results. CQs are terminated when a stop condition is satisfied or when the user explicitly terminates them.

**CHALLENGES:** GPS devices and/or roadside sensors provide (*timestamp,location*) samples at discrete time intervals in a streaming fashion. These locations are then aligned on the road network by a *map-matching* algorithm to produce moving object trajectories. Traditional methods assume that either entire trajectories are available in a static *Moving Objects Databases (MOD)* or trajectories are evolving and the MOD is constantly being updated to reflect the latest state of the trajectories. In either case, there is a need to store every trajectory on disk in order to predict near future positions. Assuming that the rate at which these location updates arrive to the central server is increased drastically, due to more vehicles on the road or finer sampling intervals, storing and querying the MOD will result in degraded performance quality. Possessing the ability to compress the historical trajectories and attempting to maintain a near constant query time with high speed location update streams poses a new challenge.

Efficient execution of CQ in a streaming environment adds new challenges. Techniques developed for traditional databases cannot be applied directly or adapted easily for streaming data. Streaming data is highly dynamic in nature with its characteristics constantly changing over time.

Prediction also presents a set of key challenges. In realistic traffic scenarios one cannot make assumptions about the motion of objects; such as objects always following the shortest path/route between two points [21] or possessing prior knowledge of an object's final destination [20].

In an attempt to refine the problem by *relaxing* some crucial assumptions, interesting challenges present themselves. For example, on approaching an intersection, how can a vehicle's future location be determined at exactly *n* time units from now? At every junction, can we assume that a vehicle is *equally* likely to follow the connected on-going road segments? Does the knowledge

of a vehicle's past movement patterns manifest itself as higher accuracy in prediction of future locations? The aforementioned questions form the base of the hypothesis of this thesis.

**PROPOSED SOLUTION AND CONTRIBUTIONS:** In this thesis, the trajectories considered are incrementally evolving and are submitted continuously to a central server as parts of the trajectory become available. The following tasks are performed on the server:

1. Evolving trajectories of moving objects are received in a streaming fashion. In real-world situations, there is an inherent uncertainty in the location of a moving object between trajectory samples. An advantageous side-effect of streaming, is that reducing the sampling interval of GPS reads can reduce the *location uncertainty* [46] introduced between consecutive samples while the streaming solution can still scale well.
2. The incoming stream is split into two streams, say  $S_1$  and  $S_2$ , where  $S_1$  is used for mining frequent routes and  $S_2$  is used for prediction CQs.
  - a) The frequent routes are incrementally mined in streaming fashion from  $S_1$ , using a *Data Stream Management System (DSMS)* and contiguous historical patterns. This avoids storage of excessive amounts of mined routes and serves as a mechanism to capture the history of object trajectories in a compact format. These patterns are annotated with the average traversal times of vehicles over each road segment, thus we are equipped with temporal information intrinsically.
  - b) Predictive CQs are now carried out on "new" incoming partial trajectories in  $S_2$ , in conjunction with the results obtained from mining stream  $S_1$ . These CQs make use of a novel movement prediction model.

In summary the main contributions of this thesis are:

1. Proposition of a novel prediction model to answer future CQs and provide a way to make more informed predictions based on historical closed movement patterns collected.
2. Formulation of new CQs that capture probabilistic future locations of moving objects, objects that will stop/park and compute the network density [21] i.e. the total number of vehicles on a unit length road segment at a given time instant in the future.
3. Accurate distribution of location probabilities based on historical knowledge captured in closed patterns.
4. The evaluation of the performance and accuracy of the approach is empirically evaluated on a large real-world data set of moving objects to demonstrate the efficiency in discovering detailed closed frequent routes with potentially high predictive utility in an incremental fashion.

The rest of the thesis is structured as follows. Chapter 2 describes related work. Chapter 3 gives an overview of the technologies used. Chapter 4 provides formal definitions of the problem statement and other concepts used. Chapter 5 briefly describes the closed mining procedure. Chapter 6 shows the components involved in the system's architecture and how they interact with one another. Chapter 7 explains the crux of this thesis which is the prediction model and its underlying algorithms. Chapter 8 provides insight into the data flow of the system and illustrates with code snippets how information is *streamed* from one component to another. Chapter 9 outlines the experiments that were carried out followed by Chapter 10 which presents the conclusion and future work.

## RELATED WORK

*Predictive Query Processing* has lately been the focus of many research groups. The broader taxonomy of current research in location prediction models, as shown in Figure 2.1 is classified at the highest level as *Free terrain models* where land surfaces are modelled as 2-D euclidean space, on which the object's movements are unrestricted and *road network based models* where the motion of objects is restricted by the road network's topology. Free terrain models can further be classified into *linear models* [17], [35], [34], [23], [16] and *non-linear models* (E.g. polynomial movement models, recursive motion function (RMF)) [39], [3]. Road network based models can be classified as ones that form predictions based on short movement history while ones that base predictions on larger collections of past movement patterns.

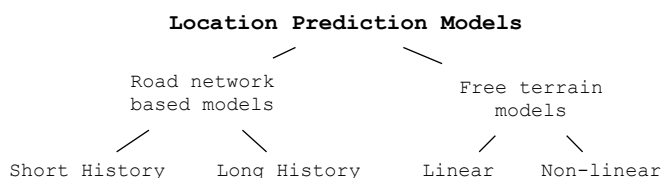


Figure 2.1: Taxonomy of Location prediction models

In a linear prediction model, given an object's initial position  $p_0$  and velocity  $v_0$  at time  $t_0$ , the future position at time  $t_f$  is given by the formula :  $p(t_f) = p_0 + v_0 \cdot (\delta t)$  where  $\delta t = t_f - t_0$

Linear models fail to capture turns in trajectories. Non-linear motion functions provide higher prediction accuracies as they capture object trajectories by applying more advanced mathematical equations. The *Recursive Motion Function (RMF)* [39] provides the most accurate prediction among the linear and non-linear models. The RMF uses a very short history of movement patterns and can capture smooth motions but falls short when vehicles take sudden turns at junctions or take u-turns. RMF can also exhibit drastically incorrect results with small increments of the future time parameter in the predictive query.

[Brilingaité](#) [6] suggests recording and storing past historical routes of objects to perform time and location queries against this information to predict a future location. The solution appears not to scale well as the trajectory database expands over time.

[Kim et al.](#) [20] assume that final destinations of all vehicles are known apriori. This thesis strays away from such pre-conditions.

[Jeung et al.](#) [18] and [Pinelli et al.](#) [27] employ a clustering algorithm to group and cluster trajectories in a MOD and form *regions*. A region's density is calculated by

the number of distinct trajectories that cross the region in question. In addition they assume that these clusters are non-overlapping. These approaches provide predictions at a higher granularity level i.e. regions as opposed to this thesis' approach which tracks vehicles down to the road segment level.

The ability to finally aggregate individual probability distributions of each moving object warranted investigation into current state of the art literature regarding network density predictions of road networks. [Kriegel et al. \[21\]](#) and [Hadjieleftheriou et al. \[13\]](#) both analyze and predict traffic density in a network. They employ statistical approaches which are based on short-term observations of traffic history. The main difference between these ideas is that the [Hadjieleftheriou et al.](#) assumes that the object trajectories are known in advance while [Kriegel et al.](#) doesn't make such an assumption. The [Kriegel et al.](#) approach assumes that each and every object on the road network acts rationally by choosing the shortest path to its destination. However, such an approach may not be completely realistic as sometimes vehicles have to take detours due to road works and many times drivers choose to drive on paths known to them with familiar landmarks, which may not necessarily be the shortest possible path [28].

Another approach by [Shekhar et al. \[36\]](#) derives current traffic data from sensors placed at certain road segments in the road network. The data from these sensors are collected in a large data warehouse and used to deduce traffic patterns. It differs in its aggregation analysis approach as it aggregates traffic at strategically placed sensors on road segments instead of observing single moving object trajectories.

[Jeung et al. \[19\]](#) follow an approach closest to this thesis' approach, where they propose a *network mobility model* that consists of (i) turning patterns at road junctions and (ii) mined travel speed on road segments. They then employ a maximum likelihood and greedy algorithm to predict the future travel segments of a moving object/vehicle. The difference being that my thesis uses both historical closed patterns and turn patterns to form a foundation for the prediction model.

## BACKGROUND TECHNOLOGIES

---

This chapter provides an overview of the technologies used in this thesis. We address the areas of data stream management systems and their relevance to the problem tackled in this thesis.

### 3.1 DATA STREAM MANAGEMENT SYSTEM (DSMS)

Traditional databases have been used in applications that require persistent data storage and simple or complex querying. Usually, a database consists of a set of objects, with data manipulation activities like insertion, updates and deletions, occurring less frequently than queries. Adhoc queries are issued and the result reflects the current state of the database object queried, that satisfies the query's conditions.

A recent proliferation of data collection devices shapes a new set of challenges that do not align with the static storage and ad-hoc querying paradigm. Information now has to be viewed in the context of a high-speed, high-volume *stream* of data values which cannot be stored on disk and has to be queried *on-the-fly*. Some known examples of such streaming data include sensor data [5], internet traffic [38] [11] and traffic updates.

A *data stream* is real-time, continuous and infinite sequence of items. The chief constraints on streaming data are that there is no control over the order in which items arrive for processing and it is not possible to store an entire stream on disk. A DSMS is employed to run queries called "Continuous Queries (CQ)" [22] [8] which run for a given period of time over these data streams and *continuously* generates results.

In this thesis, the data stream is generated by GPS enabled moving objects on a road network. The input stream is a time ordered sequence of  $(veh, seg\_id, \delta t)$  tuples where *veh* denotes a vehicle identifier, *seg\_id* represents a road segment and  $\delta t$  is the time it takes *veh* to traverse road segment *seg\_id*.

A DSMS generates *sliding windows* over the incoming data stream. The sliding window provides the user with a snapshot of the stream at any given time instant. The advantage of this approach is that the data in the current window reduces to a *virtual relation* which can then be processed by relational operators.

In general, every window has a start marker *S* and end marker *E*. The *window size* is defined as the distance between the markers *E* and *S* and the *window stride* is defined as the movement of both *S* and *E* by a fixed amount. Two types of windows are allowed; When the window is defined in terms of a time interval it is time-based while count-based windows are defined in terms of number of

items. So in a time-based context, the markers can be seen as timestamps while in the count-based model they are to be viewed as item indices.

As the traffic data is generated as a stream of timestamped location updates, the *time-based* windowing model was used extensively. CQs were written based on the output of the time-based windows. Amos II (explained in a later section), a main memory object-oriented DBMS was used to implement the relational operations on the data in each window.

### 3.2 AMOS II AND AMOSQL

AMOS II [31] [33] [32] is a distributed mediator system that allows heterogeneous data sources to be reconciled through a *wrapper-mediator* approach, where wrappers provide a common model for the different data sources and the mediator combines these views provided by the wrapper.

In standalone mode, Amos II is an extensible, main memory, OO DBMS that uses a functional query language called AmosQL to express both object-oriented and functional queries. Data is stored in the form of *objects* that model the user-defined and real-world entities. Furthermore, *functions* describe the properties and semantics of objects and the relations between participating objects.

Amos II can be extended by external interfaces [30] to regular languages like ANSI C, ALisp [29] (a common lisp subset) and Java. Amos provides a *callin* interface which allows external programs to use Amos II functionality. Additionally, it provides a *callout* interface to call *foreign functions* implemented in C, ALisp or Java.

This thesis took advantage of Amos II's Vector datatype which allowed for very efficient formation of evolving trip trajectories and compact storage. The mining algorithm to extract frequent routes and the prediction model to generate location predictions were both implemented as foreign functions in C. The DSMS's sliding window feature provided the inputs to these foreign functions. The output of these foreign functions was then re-introduced back to Amos II as objects that could then take advantage of all the querying features provided by Amos II.

### 3.3 SCSQ

SCSQ is a data stream management system built on top of Amos II with added facilities for querying of large volume, distributed streams through its high-level, declarative, continuous query language SCSQL and its highly scalable stream splitting facilities [45] [44] [10]. SCSQL is an extension of AmosQL but extended with parallel stream query facilities. SCSQ's stream operators were used in the thesis to perform the time-based windowing of the incoming streams.

## PRELIMINARIES AND DEFINITIONS

---

The following section first defines a formal framework for road network based continuously evolving trajectories of moving objects. Then, it defines the tasks of mining closed frequent routes from finite sets and unbounded streams of such trajectories. Finally, based on the extracted knowledge it defines the tasks of continuous moving object location- and density prediction on road networks.

### 4.1 ROAD NETWORK BASED ROUTES OF MOVING OBJECTS

Let  $O = \{o_1, \dots, o_M\}$  be a set of moving objects. Let the time domain be denoted by  $\mathbb{T}$  and be modeled as the totally ordered set of natural numbers  $\mathbb{N}_0$ . Then, the *free terrain movement model* describes the continuous movement of a moving object in the 2D Euclidean space with a *Euclidean trajectory*:

**Definition 1.** *The Euclidean trajectory of a moving object  $o \in O$  is a sequence of timestamped locations  $tr_{enc}^o = \langle l_1, \dots, l_n \rangle$ , where  $l_i = (x_i, y_i, t_i)$ ,  $x, y \in \mathbb{R}$ , and  $t_i \in \mathbb{T}$ .*

The movement of vehicles, the object of interest in this paper is by large restricted to movement on road networks. A typical geographical representation of a road network [7] [15] models the transportation infrastructure in the 2D space as follows.

**Definition 2.** *Let  $B \subset \mathbb{R}^2$  be the set of base points. A polyline is a sequence  $pl = \langle b_1, \dots, b_k \rangle$  of base points  $b_i \in B$ . A part of a polyline that connects two consecutive base points is referred to as a line segment, *ls*, or segment for short, and belongs to  $LS \subset B^2$ . Roads in the network-represented polylines only connect to each other at connections,  $C \subset B$ , which are locations where there is an exchange of traffic. Consequently, a road network is defined by the triplet  $R = (B, LS, C)$ .*

Given such a road network representation, an alternative and more realistic movement model, the *road network based movement model* describes the continuous movement of a moving object on the road network with a *road network based trajectory* as follows.

**Definition 3.** *The road network based trajectory of a moving object  $o \in O$  is a pair  $tr_{net}^o = (ts, s_{net})$ , where *ts* denotes the starting time of the trajectory and  $s_{net} = \langle (ls_1, \Delta t_1), \dots, (ls_m, \Delta t_m) \rangle$  is a temporally annotated sequence, i.e., a sequence of pairs of traversed segments  $ls_i \in LS$  and associated traversal times  $\Delta t_i$ , where  $\Delta t_i$  denotes the time it took  $o$  to traverse segment  $ls_i$ , also denoted as  $\Delta t^o(ls_i)$ .*

In the following, the term ‘road network based’ is omitted from the term ‘road network based trajectory’, and unless explicitly stated otherwise, the term

‘trajectory’ refers to ‘road network based trajectory’. Similarly, in the following, the ‘net’ subscript is omitted from the notations  $tr_{net}^o$  and  $s_{net}$ .

Considering the movement of an object  $o \in O$ , the number of elements in the sequences of  $tr_{euc}^o$  and  $tr_{net}^o$  are not equal, i.e.,  $n \neq m$ . In particular, several consecutive timestamped locations in  $tr_{euc}^o$  can be associated with a single element  $(s_i, \Delta t_i)$  in the sequence  $s_{net}$ , and two consecutive timestamped locations in  $tr_{euc}^o$  can be associated with several consecutive elements in the sequence  $s_{net}$ .

#### 4.2 TRIP TRAJECTORIES OF MOVING OBJECTS

Most objects do not move continuously, but rather remain stationary for some time after having reached their respective destinations. These movement pauses naturally subdivide the trajectory  $tr^o$  of an object  $o \in O$  into a sequence of *trip trajectories*  $\langle tr^o[1], \dots, tr^o[r] \rangle$ . The end of a trip trajectory of an object can either be explicitly signaled by the object, or can be automatically inferred by spatio-temporal analysis of the trajectory. A trip trajectory  $tr^o[i]$  of  $o$  is modeled in the same way as an object trajectory. In subsequent usage the term ‘trip’ is omitted from the term ‘trip trajectory’ when the distinction can be inferred from the context.

For privacy reasons and from an application perspective it is reasonable to assume that the location of a moving object is not always sampled. Consequently, the sequence of trip trajectories of a moving object does not form a spatially contiguous trajectory. This is illustrated by the trajectory and trip trajectories of an object that moves on a street grid, shown in Figure 4.1. In Figure 4.1, a connection in the street grid is referenced by the concatenation of its  $x$  and  $y$  coordinates, and a directed segment is referenced by a concatenation of the references of the starting and the ending base points of the directed segment. For example, the connection labeled as  $A$  is referenced by  $02$  and the directed segment from connection  $A$  to connection  $B$  is referenced as  $0212$ . Figure 4.1 shows 3 trip trajectories,  $tr^o[1]$ ,  $tr^o[2]$ , and  $tr^o[3]$  (path along solid black arrows) of an object  $o$ ’s trajectory (path along both solid black and dashed gray arrows) starting at connections  $02$ ,  $90$ , and  $63$ , at times  $ts_1 = 0$ ,  $ts_2 = 50$ , and  $ts_3 = 90$ , respectively. The first trip trajectory  $tr^o[1]$  starts at connection  $02$  ( $A$ ) at time  $0$ , ends at connection  $62$  ( $C$ ) at time  $13$ , and, given that the numbers above the segments denote traversal times, is represented by the pair  $(0, \langle (0212, 2), (1211, 1), (1121, 1), (2131, 1), (3132, 2), (3242, 2), (4252, 2), (5262, 2) \rangle)$ . During the period between time  $13$  and time  $50$ , the object has moved from connection  $62$  to  $90$ . The trajectory for this movement is indicated by the path along the dashed gray arrows in the figure, but for the previously mentioned reasons it is excluded from the object’s sequence of trip trajectories. The description of the remaining parts of the object’s trajectory is self explanatory.



temporally annotated sequence)  $r = \langle (ls'_1, \Delta t'_1), \dots, (ls'_l, \Delta t'_l) \rangle$ .  $tr_i$  supports  $r$ , or  $r$  is a *sub-sequence* of  $s_i$ , equivalently denoted as  $r \preceq tr_i^o$  and  $r \preceq s_i$ , respectively, iff there exist an index sequence  $1 \leq i_1 < \dots < i_l \leq m$  such that  $ls'_j = ls_{i_j}$   $\forall j$  where  $1 \leq j \leq l$ . The temporal annotation of the route, i.e., the traversal time of a given segment of the route is defined to be (and is calculated as) a sequence aggregate of the traversal times of the corresponding segment of the trajectories that support the route. For example, when the aggregate is the arithmetic average<sup>1</sup>, for a route  $r = \langle (ls'_1, \Delta t'_1), \dots, (ls'_l, \Delta t'_l) \rangle$ , the traversal time of segment  $ls'_j \in LS$  for  $1 \leq j \leq l$  is:

$$\Delta t'_j = \frac{\sum \{\Delta t^{(\cdot)}(ls'_j) : r \preceq tr_i, tr_i \in TR\}}{|\{tr_i : r \preceq tr_i, tr_i \in TR\}|}.$$

Effectively, the traversal times of segments in trajectories are not considered in determining the support of a route, but are instead calculated as an aggregate of the support set. Consequently, the *frequent route mining* task is defined as finding all routes that are supported by at least  $\min\_sup$  trajectories.

As a frequent route  $r$  with  $n$  segments has  $2^n - 2$  frequent *proper non-empty sub-routes* (most of which have the same support as  $r$  and are hence largely redundant), the task is further refined to mining only closed frequent routes. A route  $r_c$  is a *closed frequent route* iff  $sup(r_c) \geq \min\_sup$  and there exists no frequent extended route  $r_e$  such that  $r_c$  is a proper subsequence of  $r_e$ , i.e.,  $r_c \prec r_e$ , and the support of  $r_c$  is equal to the support of  $r_e$ , i.e.,  $sup(r_c) = sup(r_e)$ .

Finally, a frequent route does not encode the absolute traversal period-, but rather only encodes the aggregated traversal duration of every one of its constituent segments. Consequently, a frequent route in which segments are not spatially connected, i.e., have a 'gap', can only ambiguously describe the movement of its supporting objects. More concretely, such a frequent route does not contain any spatial or temporal information about the object's movements between the end of the segment before the gap and the beginning of the segment after the gap. Such frequent routes with gaps have limited use for moving object location- and density prediction. Hence, the task is further refined to mining only *contiguous frequent routes*. A route  $r$  is a contiguous frequent route iff  $r$  is *contiguously supported* by at least  $\min\_sup$  trajectories. A trajectory  $tr_i = (ts_i, s_i) \in TR$  *contiguously supports*  $r$ , or  $r$  is a *contiguous sub-sequence* of  $s_i$ , equivalently denoted as  $r \preceq_c tr_i^o$  and  $r \preceq_c s_i$ , respectively, iff there exist an index sequence  $1 \leq i_1 < \dots < i_l \leq m$  such that  $i_{j+1} - i_j = 1 \forall j$  where  $1 \leq j < l$  and  $ls'_j = ls_{i_j} \forall j$  where  $1 \leq j \leq l$ .

Given the continuously evolving nature of trip trajectories, trip trajectories of objects  $o \in O$  are not observed as a finite *set* of trip trajectories  $TR$ , but rather as a *continuous Stream of timestamped trip Trajectory Pieces of objects*, denoted as  $\mathcal{STP}$ . Formally, an  $\mathcal{STP}$  is an unbounded ordered sequence  $\langle e_1, e_2, \dots \rangle$  of elements where every element is a three-tuple  $e_i = (o_i, tp_i, t\_arr_i)$  in which  $tp_i$  represents a particular trajectory piece of object  $o_i$  (for some trip) with arrival time  $t\_arr_i$

<sup>1</sup> Other potentially meaningful sequence aggregate alternatives include the *last* function and the family of (*temporally*) *decaying average* functions. In the current proposal the simple arithmetic average is used.

and  $t_{arr_i} \geq t_{arr_{i-1}}$  for  $i > 1$ . Mining frequent routes from an  $\mathcal{STP}$  is facilitated by adopting a commonly used temporal sliding window processing model for streams as follows. Given a stream of trajectory pieces  $\mathcal{STP}$  of objects  $O$ , temporal sliding window parameters window size  $t_{wsize} \in \mathbb{N}$  and stride  $t_{wstride} \in \mathbb{N}$ , and support threshold  $\min\_sup$ , the *online closed contiguous frequent route mining* task is defined as finding for each  $t_{wstride}$ -slide of the temporal window (identified by the time interval  $(t_c - t_{wsize}, t_c]$ ) at *current time* instance  $t_c \in (a \times t_{wstride} + t_{wsize})$  where  $a \in \mathbb{N}^+$  all closed contiguous frequent routes in the contiguous trip sub-trajectories of objects that are formed by the trajectory pieces of objects which have an arrival time that falls within the time interval  $(t_c - t_{wsize}, t_c]$ . For the remainder of the thesis, unless the emphasis is needed, the terms ‘closed contiguous frequent route’ and ‘pattern’ are used synonymously.

#### 4.5 MOVING OBJECT LOCATION AND DENSITY PREDICTION

The motivation of mining frequent routes is to utilize a relevant subset of the extracted historical patterns in the fundamental task of predicting the near-future location of an object on the road network given its recent trip trajectory. The task is formally defined as follows.

**Definition 4.** *Moving Object Location Prediction: Given a road network  $R = (B, LS, C)$ , a set of closed contiguous frequent routes  $FR(\min\_sup)$ , and the contiguous trip sub-trajectory  $\langle tp_i^o[r], \dots, tp_k^o[r] \rangle$  of object  $o$  for its current trip  $r$  up to the current time  $t_c$ , for all segments  $ls_i \in LS$  calculate the probability that  $o$  will be located on the segment  $ls_i$  at the prediction time  $t_p \geq t_c$ , denoted as  $\Pr(ls_i^o | t_p)$ .*

Subsequently the derived task of predicting the density of objects on the road network in the near-future is formulated as follows.

**Definition 5.** *Road Network Density Prediction: Given a road network  $R = (B, LS, C)$ , a set of closed contiguous frequent routes  $FR(\min\_sup)$ , and the contiguous trip sub-trajectories of a set of object  $O$  up to the current time  $t_c$ , for all segments  $ls_i \in LS$  calculate the expected number of objects that will be located on the segment  $ls_i$  at the prediction time  $t_p \geq t_c$ , denoted and calculated as  $E(ls_i | t_p) = \sum_{o \in O} \Pr(ls_i^o | t_p)$ .*

Adopting the previously introduced temporal sliding window processing model for streams the online versions of the afore defined tasks can be formulated as follows.

**Definition 6.** *Online Moving Object Location Prediction: Given a road network  $R = (B, LS, C)$ , a stream of trajectory pieces  $\mathcal{STP}$  of objects  $O$ , an object  $o \in O$ , temporal sliding window parameters window size  $t_{wsize} \in \mathbb{N}$  and stride  $t_{wstride} \in \mathbb{N}$ , temporal mining window parameters mining window lag  $t_{wlag}^m \in \mathbb{N}$  and mining window size  $t_{wsize}^m \in \mathbb{N}$ , and prediction time  $t_p$ , at every current time instance  $t_c \in (a \times t_{wstride} + t_{wlag}^m + t_{wsize}^m)$  where  $a \in \mathbb{N}$ , based on the set of closed contiguous frequent routes  $FR(\min\_sup)$  for period  $(t_c - t_{wsize}^m - t_{wlag}^m, t_c - t_{wlag}^m]$  and the most current contiguous trip sub-trajectory of  $o$  for period  $(t_c - t_{wsize}, t_c]$ , for all segments  $ls_i \in LS$  calculate  $\Pr(ls_i^o | t_p)$ .*

**Definition 7.** Online Road Network Density Prediction: Given a road network  $R = (B, LS, C)$ , a stream of trajectory pieces  $STP$  of objects  $O$ , temporal sliding window parameters window size  $t_{wsize} \in \mathbb{N}$  and stride  $t_{wstride} \in \mathbb{N}$ , temporal mining window parameters mining window lag  $t_{wlag}^m \in \mathbb{N}$  and size  $t_{wsize}^m \in \mathbb{N}$ , and prediction time  $t_p$ , at every current time instance  $t_c \in (a \times t_{wstride} + t_{wlag}^m + t_{wsize}^m)$  where  $a \in \mathbb{N}$ , based on the set of closed contiguous frequent routes  $FR(\min\_sup)$  for period  $(t_c - t_{wsize}^m - t_{wlag}^m, t_c - t_{wlag}^m]$  and the most current contiguous trip sub-trajectories of objects  $o \in O$  for period  $(t_c - t_{wsize}, t_c]$ , for all segments  $ls_i \in LS$  calculate  $E(ls_i | t_p)$ .

## CLOSED FREQUENT ROUTE MINING IMPLEMENTATION

---

### 5.1 DESIGN BACKGROUND

The prototype implementation of the proposed online frequent route mining system consist of two main parts: a trip management module and a frequent closed route mining module. The former serves the purpose to maintain and filter the current trip set so that only the intersections of the available trips with the next mining window are passed to the mining module, which then performs the actual frequent closed route detection.

In the current prototype the trip management module is implemented as a simple list of “active” trips (active in the sense that they are relevant for the next mining run, because they have a non-empty intersection with the next mining window) together with an array-based access structure for incomplete trips, so that incoming partial trips can be chained to their already known prefix. The array structure is simply indexed by the vehicle/moving object identifier, since there can be at most one incomplete trip per vehicle/moving object.

The mining module works, like many frequent item set and frequent sequence miners do, by growing patterns in a depth-first fashion. That is, the search commences, on the top level, with single segment patterns, which are then recursively extended by appending one segment in each recursion step. As the data structure we use a simple flat array representation of the trips, into which we keep references to the current ends of the pattern occurrences in order to be able to quickly find and group possible pattern extensions. These extensions are collected with a technique called *occurrence deliver* in [40] [41]: the suffixes of the trips after the end of each pattern occurrence are traversed and the occurring segments are collected. In this way it is possible to efficiently form lists of the ends of the occurrences of all pattern extensions, which are then processed recursively.

### 5.2 CLOSEDNESS CHECKS IN TRAJECTORIES

The most demanding part of the mining module is the check whether a given frequent pattern is closed. In principle, there are two strategies how one may carry out this check. The first, which is popular in frequent item set mining, uses a repository of already found (closed) frequent patterns and checks whether there exists a superpattern in the repository that has the same support. If this is the case, the pattern is not closed (and neither are any of the superpatterns that would be explored in descendant nodes in the search tree). After having been used in CLOSET [24] and CHARM [43], a particularly efficient variant

of this approach was developed in [12], where an improved FP-growth [14] implementation was enhanced by FP-tree structured conditional repositories. The alternative to a repository is a direct check of possible superpatterns and their support by generating and testing all possible extensions of a given pattern. This approach has successfully been used for frequent item set mining, for instance, in LCM [40] [41].

According to the results reported in [42], the repository approach has certain disadvantages for frequent sequence mining. The main reason seems to be that, in contrast to frequent item set mining, the order of the items matters, due to which the pattern space cannot be structured quite as nicely. In particular, it cannot be ensured that for any pattern its superpatterns occur only as search tree descendants or in search tree branches that are traversed before the branch of the pattern itself. This makes it necessary to search the repository not only for super-, but also for subpatterns and to replace such subpatterns if they have the same support as the current pattern. As a consequence, the direct check of pattern extensions is the preferred strategy, which is therefore used in [42] and which we also adopted.

Extensions of the current pattern that are formed by appending segments are easily handled: these are considered in the recursive processing and thus it suffices to return from the recursion the highest support of such an extension in order to check for a closed pattern. The problem resides with extensions that are formed by prepending segments before or (worse) inserting segments between the items of the pattern under consideration. In order to test efficiently whether any of these extensions has the same support, we exploit (as a simplification over the more general BIDE algorithm [42]), that in our data a segment cannot occur more than once in each trip. As a consequence, any pattern can occur at most once in each trip, which makes it possible to use a single static flat array structure to represent all occurrences of a pattern under consideration.

Note that this representation of the occurrences of a pattern can easily be extended to yield the occurrences of extended patterns by drawing on the extension lists that were collected by the occurrence deliver technique (see above). Hence it is not necessary to find the occurrence of a pattern from scratch if this pattern is to be checked. The actual check is then carried out as follows: with the pattern occurrences we know where each segment of the pattern occurs in the trips. We traverse the possible insertion points (before the first and between any pair of pattern segments) and for each of them we successively intersect the sets of intermediate segments in the actual trips. As soon as the intersection gets empty, we know that there does not exist an extension segment for the current insertion point that can yield the same support, because there is no intermediate segment that exists in all pattern occurrences. If, however, there exists an insertion point where the intersection of the sets of intermediate segments is not empty after all pattern occurrences have been processed, we know that there exists a superpattern with the same support: we can insert any of the segments in the intersection.

Note that the check of the insertion points is carried out before appending segments, so that no patterns that are non-closed due to inserting or prepending

segments are ever extended. If a pattern passes the check and the recursion which appended segments returns with a maximum extension support less than the support of the current pattern, we know that the pattern is closed and hence it is reported (provided it is frequent). Otherwise the pattern is ignored.

In addition to the condition that a pattern must be closed and frequent to be reported, our prototype implementation allows to constrain the size of the patterns by specifying a minimum and a maximum number of segments.

## SYSTEM ARCHITECTURE

This chapter outlines the overall client-server architecture. The components and events that take place are outlined in the Figure 6.1 below along with detailed descriptions of each action.

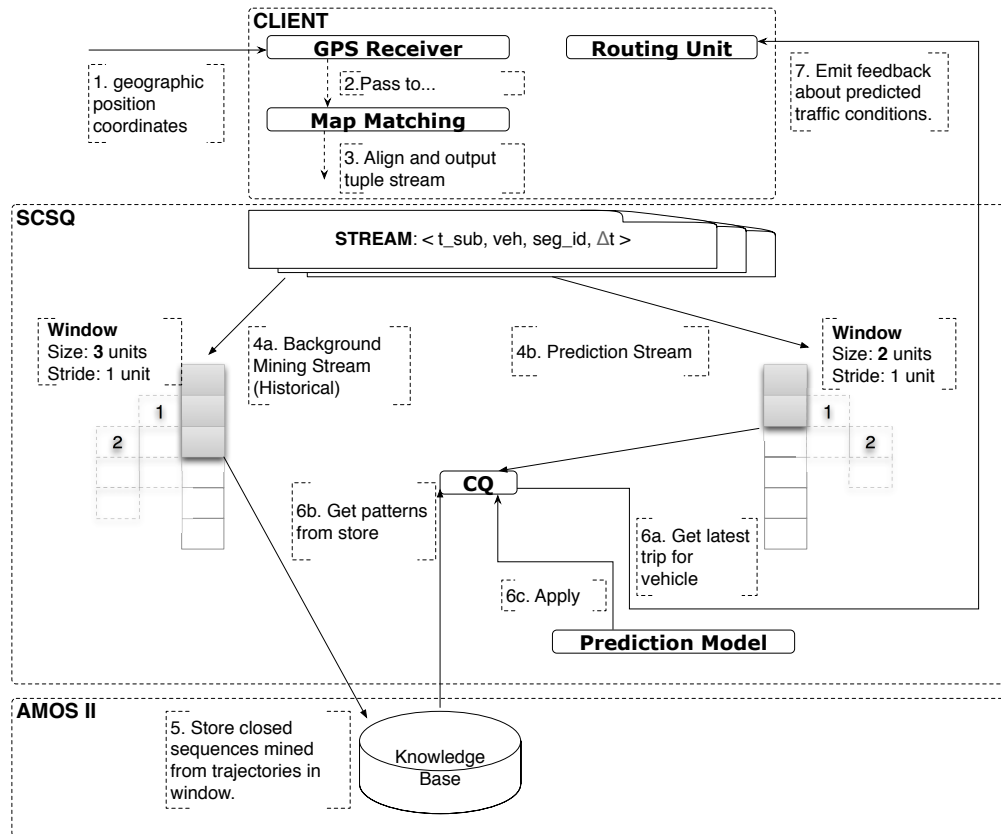


Figure 6.1: System Architecture.

## 6.1 MAP MATCHING

The map matching algorithm aligns the incoming GPS coordinates to the closest road segment on the road network. This algorithm has to cater for noise removal because GPS coordinates are measured within certain tolerance limits and hence may or may not correctly align with the geometry of the road network's segments. On completion of the alignment phase, the map matching algorithm collates the GPS coordinates to form road segments and outputs a stream of tuples. Further details of the map matching algorithm's functionality are out of scope of this thesis.

## 6.2 TUPLES STREAM IN SCSQ

The client outputs a stream of tuples of the form  $\langle t\_sub, veh, seg\_id, delta\_t \rangle$  where

- $t\_sub$  = the submission timestamp,
- $veh$  = an identifier for the vehicle by the application,
- $seg\_id$  = an identifier for the road segment assigned by the client and maps to a known road segment on the road network and
- $delta\_t$  = the time taken by  $veh$  to traverse  $seg\_id$

These incoming tuples are converted to a *Stream Object* in SCSQ, which allows it to then be easily manipulated for querying. SCSQ is responsible for performing a *split operation* to split the stream into two duplicate streams for feeding into the mining process and prediction process respectively.

## 6.3 MINING STREAM IN SCSQ

After a stream split operation in SCSQ, a stream is registered to the closed sequence mining algorithm. As an illustration in Figure 6.1, the mining window has a size of 3 time units and the sliding window has a stride of 1 time unit. The mining stream continues to run in synchronization with the prediction stream. The frequent routes from the mining stream are continuously written to the *Knowledge Store*, from which the prediction stream extracts the frequent routes for prediction.

The outputs are stored as follows:

- $t\_start$  = the window's start timestamp
- $t\_end$  = the window's end timestamp
- Temporally Annotated Sequence (TAS):
  - $pv$  = closed pattern vector
  - $tv$  = corresponding traversal time vector
- $supp$  = support of the closed pattern

*Note:* The  $t\_start$  and  $t\_end$  are indexed with B+-tree indexes for faster retrieval.

The closed sequences that are *continuously mined* are stored in an Amos II stored function. This produces a very compact footprint of the trajectories that are seen.

## 6.4 PREDICTION STREAM IN SCSQ

After the stream split in SCSQ, a stream is registered to the prediction algorithm which performs future location predictions on the actual evolving trajectories

in the stream. In Figure 6.1, the current trajectory window has a size of 2 time units and the sliding window has a stride of 1 time unit.

*Note* The window size for mining has to be larger than the current trajectory window size; this is done in order for the prediction CQs to be able to extract longer patterns that potentially match the future segments of the evolving trajectory.

## 6.5 CONTINUOUS QUERY (CQ) IN SCSQ

Several CQs can be run in order to extract future traffic predictions such as future moving vehicle locations, future parked vehicle locations, future network density. For example, a CQ that computes the future location of a moving object gets input from

1. the current trajectory stream and computes the latest trip of the vehicle marked as *6a* in Figure 6.1
2. the appropriate historical closed pattern from the knowledge base. The closed pattern must correspond to a time window which overlaps with the current trajectory window's  $t\_start$  and  $t\_start + ftime$  where  $ftime$  is the future time interval in seconds. Marked as *6b* in Figure 6.1
3. the prediction model is applied to the results from *6a* and *6b* to arrive at the final results. Note that the *Knowledge Store* is a stored function in Amos II that enables ease of querying and retrieval of patterns. Marked as *6c* in Figure 6.1

CQs piggyback on SCSQ's ability to generate a stream of tuples which form the basis for CQs. The CQ continues to run in SCSQ as long as there is a current trajectory stream and outputs results. These results are then summarized and sent back to the client *routing unit* as traffic prediction updates, which the client's routing unit can interpret as a signal to recalculate and reroute to a more optimal path to the destination.

Chapter 8 highlights the SCSQ code that was written in order to achieve the aforementioned functionality.

## PREDICTION MODEL

---

The main underlying philosophy of studying *frequent routes* is that trajectories traced by moving objects on a road network are constrained by the topology of the road network and are not completely random events. In fact there is a noticeable semblance of order in traffic behaviour mixed with unexpected anomalies like traffic jams. It's the knowledge of past ordered events in the form of frequent routes that forms the tenet of our proposed prediction model whose aim is to predict future locations of moving objects.

To get an intuitive understanding of the prediction task, consider Figure 7.1. It comprises of road segments labelled "a" to "l" and a moving object shown as a white rectangle. Let " $t_c$ " denote the current point in time at which a prediction query is issued, the moving object is located at the end of segment "b" at time " $t_c$ " and has traversed the path "a  $\rightarrow$  b" to get here. The dotted circle can be calculated to be the maximum upper bound of where the moving object can be at a future point in time " $t_c + \delta t$ " by taking into account the moving object's current location, maximum object movement speed and the maximum speed limits of the road segments. Hence, in the example at time " $t_c + \delta t$ " the moving object can be on road segments - c, d, h, i or k (assuming no u-turns). It is the task of the prediction model to assign probabilities to the event of this moving object to be located on any of the aforementioned road segments.

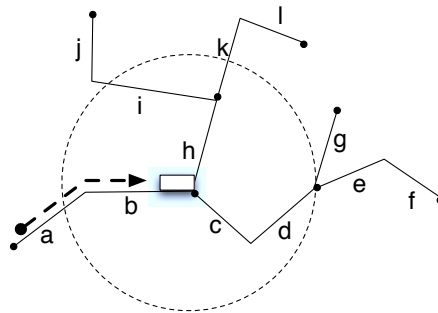


Figure 7.1: An example of path prediction of moving object

Due to the complex nature of the prediction model, the model is described with a running example of a sample road network and some sample trajectories traced over this sample road network. Then, a detailed look is taken at the algorithms employed in the prediction model referring to the same running example.

## 7.1 RUNNING EXAMPLE

For purposes of formalization, the sample road network (Figure 7.2) is defined over a regular grid on a x-y coordinate system. To describe the motion of a moving object over a road segment in a particular direction; the coordinates of the origin  $(ox, oy)$  are concatenated with the destination coordinates  $(dx, dy)$  as " $ox + oy + dx + dy$ ". For example, the horizontal segment in the lower left corner of the grid is defined as: 1222. The special edge of  $-1$  denotes the virtual edge that is used to signal the end of a trajectory. Figure 7.2 shows the topology of the grid network. The regular grid is an assumption without the loss of generality.

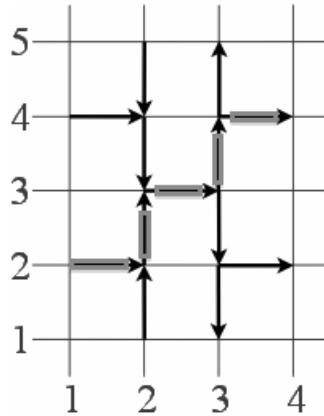


Figure 7.2: Sample trajectory on road network

## 7.2 EXAMPLE TRAJECTORIES AND FREQUENT ROUTES

Highlighted in Figure 7.2 is a single trajectory "1222 2223 2333 3334 3444  $-1$ " of a vehicle on the example road network.

To further expand the example, listed below are some more sample trajectories that are used as a running example to illustrate the base trajectories from which frequent routes are then extracted. Typically the numbers surrounded by brackets denote the support for a pattern, but in this example it is used to denote the number of vehicles that have traced that particular trajectory. For example, (20) indicates that our example trajectory was followed by 20 moving objects.

---

T <sub>1</sub>	1222	2223	2333	3334	-1	(20)
T <sub>2</sub>	1222	2223	2333	3334	-1	(5)
T <sub>3</sub>	1222	2223	2333	3334	3435	-1 (5)
T <sub>4</sub>	2122	2223	2333	3334	3435	-1 (15)
T <sub>5</sub>	1424	2423	2333	3332	3242	-1 (10)
T <sub>6</sub>	2524	2423	2333	3334	-1	(5)
T <sub>7</sub>	2524	2423	2333	3332	3231	-1 (20)
T <sub>8</sub>	3334	3444	-1			(10)

---

Listing 7.1: Sample trajectories on road network

Given the formal definition of the road topology and the moving object trajectories, *frequent routes* have to be determined. In order to extract the frequent routes from the given trajectories a mining algorithm as described in Chapter 5 is executed. The mining program is executed with a minimum support threshold of say 10%. The output is the union of the set of *all frequent routes that are present in more than 10% of the total number of trajectories* and the set of *all segment pairs with an absolute minimum support of 1*. The support threshold is a variable program argument and is a way of controlling the amount of information that needs to be stored about historical trajectories. If the support threshold is set too high then numerous routes can be termed *infrequent* and hence not reported. Shown below in Listing 7.2, are some sample frequent routes with their corresponding supports in brackets on the right.

---

P <sub>1</sub>	{2524, 2423, 2333, 3334}	(5)
P <sub>2</sub>	{2524, 2423, 2333, 3332, 3231}	(20)
P <sub>3</sub>	...	
P <sub>4</sub>	...	

---

Listing 7.2: Sample frequent routes

The frequent routes can be thought of as a *compressed database of historical trajectories* against which the users pose a future location query for a current trajectory. When querying for a current trajectory's future location, the first thing that has to happen is that the current trajectory needs to be *matched* with a frequent route existing in the compressed database. When such a match is found, then a probabilistic future location of the moving object can be calculated. But there can be instances where there is no matching route found in the database because the mining algorithm's support threshold was set too high and that route was suppressed.

To fill this void and to have the ability to make predictions at all times, all the possible turns a vehicle can make from a given road segment have to be captured and hence we also extract all segment pairs in the mining step.

### 7.3 CLOSED FREQUENT PATTERN TREE CREATION

The mined frequent routes are stored in a prefix tree similar to a FP-tree by Han et al. [14]. It consists of a prefix tree  $\tau_c$  and a header table  $H_c$  (which acts as an index into the prefix tree). Patterns are read in one at a time and inserted into a prefix tree to enable quick retrieval of frequent routes/patterns. Pinelli et al. [27] follow a similar approach where they identify *frequent regions* called *T-patterns* using a clustering algorithm to group together moving object trajectories using a notion of trajectory similarity which is distance-based. These T-patterns are then inserted into a *Prediction Tree* quite similar to the tree used here with the difference being that it is not a prefix tree and it doesn't include a header table that acts as an index to speed up pattern searches in the prediction tree.

Before discussing the algorithm to generate the tree, it is important to note the structure of the prefix tree  $\tau_c$ 's nodes and the nodes in the header table  $H_c$ .

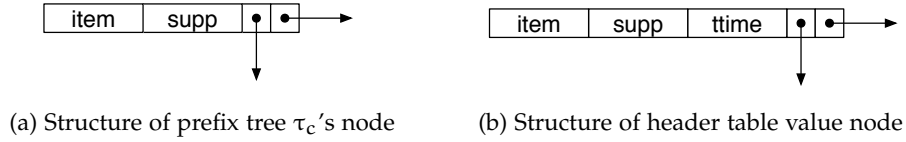


Figure 7.3: Structure of nodes

Figure 7.3a shows the structure of a node in  $\tau_c$ . It consists of the item <sup>1</sup> and its support. In addition, each  $\tau_c$  node contains a pointer <sup>2</sup> to the next node in  $\tau_c$  with exactly the same item. This is done in order to build a *linked list* of all the nodes in  $\tau_c$  that contain the same item. Hence, we construct a linked list for each unique item (also listed in the header table  $H_c$  acting as an index) in the tree  $\tau_c$ . The second pointer in the  $\tau_c$  node is used as an auxiliary pointer to reduce the search space when we calculate conditional probabilities later down the track.

Figure 7.3b shows the structure of a node in  $H_c$ . It consists of an item, its overall support and the average time it takes to traverse the item/road segment. Additionally it contains two auxiliary pointers pointing to the *head* and *tail* node of the linked list respectively.

Algorithm `genTree` shows the pseudo-code of the pattern tree generation algorithm.

---

**Algorithm genTree:** genTree(  $P, S$  )

---

**Input** : Set of *closed pattern vectors*  $P$ , Set of corresponding supports  $S$   
**Output**: Complete closed pattern prefix tree  $\tau_c$

```

1 begin
2    $k \leftarrow 0$ 
3   Initialize tree  $\tau_c$  and header table  $H_c$ .
4   for pattern  $cp \in P$  do
5     rootptr  $\leftarrow \tau_c$ .root
6     for item  $i \in cp$  do
7       for node  $n \in$  rootptr.children() do
8         if  $n$ .item =  $i$  then
9            $\tau_c \leftarrow$  Update node support as  $\max(n$ .supp,  $S[k]$ )
10          else  $\tau_c \leftarrow$  Add new node with ( $n$ .item, $n$ .supp)
11          rootptr  $\leftarrow n$ 
12          val  $\leftarrow$  lookup(  $H_c$ , item )           /* hash lookup */
13          val.supp  $\leftarrow$   $\max$ (val.supp,  $n$ .supp) /* update hash */
14           $k \leftarrow k + 1$ 
15  return  $\tau_c$ 

```

---

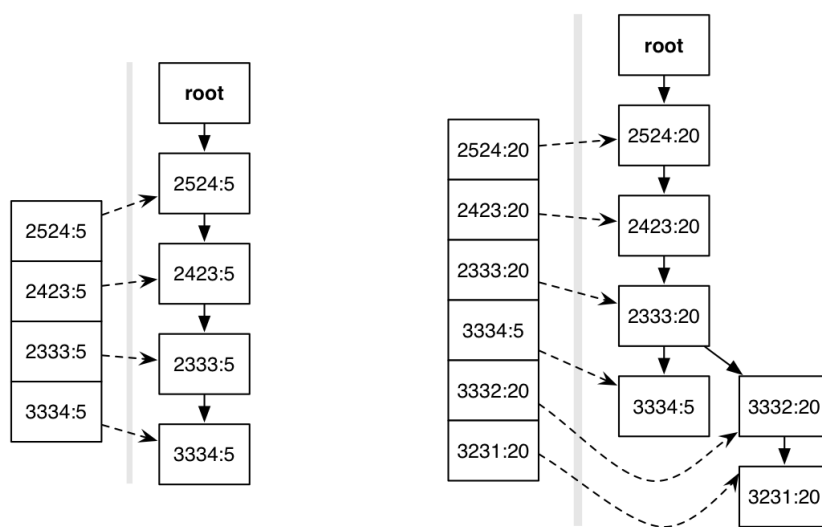
<sup>1</sup> Note the use of the term "item" to denote "road segment" in this context

<sup>2</sup> The tree ADT provides additional pointers such as a pointer to the first child, a pointer to the parent node and sibling node pointers.

The algorithm takes as input a set of closed patterns (as shown in Listing 7.2) and a corresponding set of supports for each pattern in the set of closed patterns. We will utilize the example patterns from Listing 7.2 to better understand the effects of this algorithm. In Lines 4-6, we iterate through the set of closed pattern vectors  $P$  and for each pattern  $cp$ , we iterate through each item  $i$  that constitutes this vector. Prior to inserting this  $i$  into the tree we must check if there already exists a child node under the  $rootptr$  that already contains  $i$ . In Lines 8-9, if the item already exists then the node  $n$  of the tree is updated with the maximum of its current support value and the support of the incoming item. In Line 10, if the item does not exist, then a new node is allocated and inserted into the tree, as a child of the  $rootptr$ , with the values of the incoming item  $id$  and support. The  $rootptr$  is moved down one level to match the position of the next item to be evaluated in the closed pattern vector  $cp$ .

Lines 12-13, update the header table by first looking up the item in the header table and then updating its support by the maximum of its current support value and the newly updated/inserted node's support in tree  $\tau_c$ . Not outlined in the algorithm is the fact that corresponding to each item in the header table is also a linked list of nodes of the tree where the same item exists, which must be built while inserting items in the tree  $\tau_c$  and header  $H_c$ . Having exhausted all the items in all the closed patterns in the set  $P$ , we finally end up with a completed prefix tree  $\tau_c$  which is then returned.

Figure 7.4 below demonstrates the stepwise insertion of patterns "P1" and "P2" from Listing 7.2 into the prefix tree  $\tau_c$  using the algorithm described in Algorithm [genTree](#).



(a) Step 1 :  $\tau_c$  after insertion of pattern P1      (b) Step 2 :  $\tau_c$  after insertion of pattern P2

Figure 7.4: Stepwise insertion of patterns as branches of tree  $\tau_c$

In step 1 (Figure 7.4a), after insertion of the first pattern we end up with a prefix tree  $\tau_c$  with a single branch, where each node is being shown in the format "item:support". In our example, we have all items with a support of 5 in  $\tau_c$

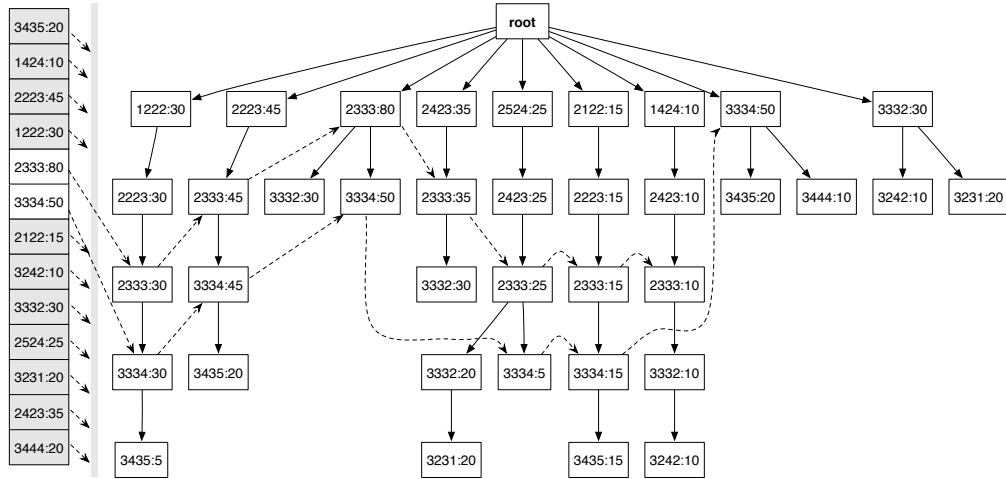


Figure 7.5: Complete prefix tree

and the same items filled in the header table on the left. After step 2 (Figure 7.4b), we notice that the first few items in the pattern we wish to insert i.e.  $\{2524, 2423, 2333, 3332, 3231\}$ , into the tree coincide with the items in  $\tau_c$ 's nodes. The effect of inserting these items into  $\tau_c$  is that the support in each of the coincident tree nodes gets updated to  $\max(20, 5)$  which is 20. The same effect is replicated in the header table.

Figure 7.5 shows the completed prefix tree  $\tau_c$  after inserting all example patterns from Listing 7.2.

#### 7.4 CORE PREDICTION ALGORITHM

This section explains the location prediction algorithm using the running example from Sections 7.1 and 7.2 in this chapter. To define some terms,  $qv$  represents a query vector which is a collection of road segments or items that a vehicle has traversed for example  $\{2333, 3334\}$ . We define the anchor  $anc$  to be the last element in  $qv$  which is 3334 in our example. The anchor always represents the most recent segment traversed by the moving object.

A major assumption of our prediction model is that the anchor *must* be present in the compressed database of closed patterns in order to make any prediction.

The prediction algorithm uses a local key-value store ( $lh$ ) and a global key-value store ( $gh$ ) to keep track of the conditional location probabilities it calculates. In Algorithm `predict`, which is recursive in nature, the local key-value store keeps track of all immediate road segments that are connected to the anchor, while the global key-value store holds the final road segments along with the future probability mass of the object.  $gh$  and  $lh$  both have the *item* as the key and the probability *prob* as the value.

Algorithm `predict` takes as input the the query vector  $qv$ , the time increment  $rt$  and the probability mass  $pm$ .

The initial call to the prediction algorithm in our example is as follows:

```
predict( {2333,3334}, 2.0, 1.0 )
```

which implies that a moving object has traced a path consisting of segments 2333 and 3334. At current time  $t_c$ , it is located at the end of the anchor segment 3334 and we wish to predict all possible road segments it could be located on at time  $t_c + 2.0$  time units. We start with an initial probability mass  $pm$  of 1.0 and attempt distributing it across possible road segments.

---

**Algorithm predict:** predict(  $qv, rt, pm$  )

---

**Input** : Query Vector  $qv$ , Remaining time  $rt$  and probability mass  $pm$

**Output:**  $gh$  with probabilistic future locations

```
1 begin
2    $anc \leftarrow \text{last}(qv)$ 
3    $lh \leftarrow \text{initialize local hash table}$ 
4    $rpm \leftarrow 0$ 
5   if  $rt \leq 0$  then
6      $gh \leftarrow \text{update using } lh$ 
7     return  $gh$ 
8    $MB \leftarrow \text{getMaxCostBranches}(qv)$ 
9    $(rpm, lh) \leftarrow \text{getPatternProbs}(lh, \tau_c, qv, MB, pm)$ 
10  if  $rpm > 0$  then
11     $(rpm, lh) \leftarrow \text{compute turn probabilities (like getPatternProbs)}$ 
12    if  $rpm > 0$  then
13       $x \leftarrow \text{assign } rpm \text{ to } (-1 * anc)$ 
14       $lh \leftarrow \text{insert } x$ 
15  for  $(key, val) \in lh$  do
16     $t_{seg} \leftarrow \text{get average traversal time of segment key}$ 
17     $\text{predict}(qv + key, rt - t_{seg}, val.prob)$ 
18  return  $gh$ 
```

---

In Algorithm [predict](#), Lines 2-4 initialize the anchor  $anc$  and the local key-value store  $lh$ . Lines 5-7 form the base case for the recursive algorithm. When the remaining time reaches or exceeds the time horizon ( $t_c + 2.0$  in our example),  $gh$  is updated from the values in  $lh$  and  $gh$  is returned to the user.

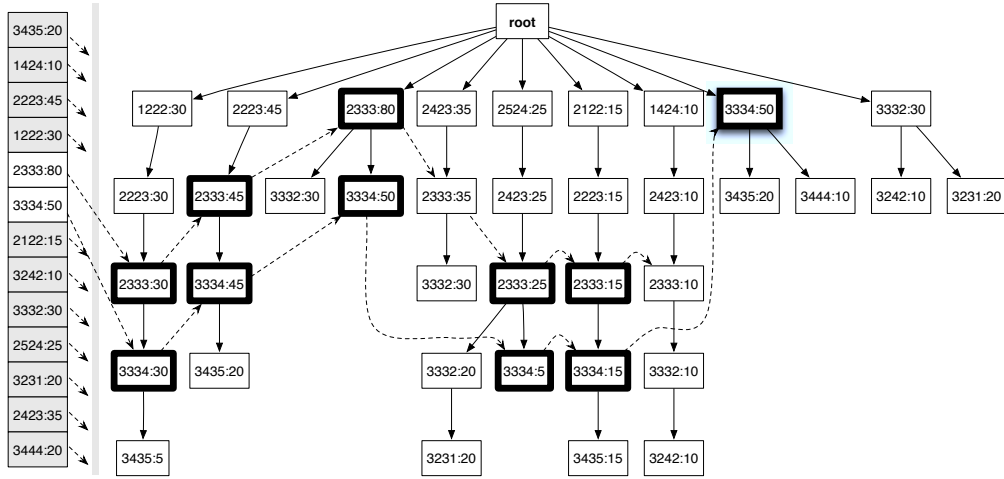


Figure 7.6: Complete prefix tree with highlighted anchor and other overlapping segments. For  $qv = \{2333, 3334\}$ , there is a total overlap in the highlighted branches except the last highlighted branch which has only the  $anc = 3334$ .

Line 8 computes all the branches in  $\tau_c$  which have a maximum cost. The cost calculation and determination of the set of branches that share the maximum cost is assigned to Algorithm [getMaxCostBranches](#).

---

**Algorithm getMaxCostBranches:** getMaxCostBranches(  $qv$  )

---

```

Input : Query Vector  $qv$ 
Output:  $MB$  : set of branches that share the maximum cost
1 begin
2    $anc \leftarrow \text{last}(qv)$ 
3    $qlen \leftarrow \text{length}(qv)$ 
4    $A \leftarrow$  Set of all nodes in tree  $\tau_c$  that contain item  $anc$ .
5   for  $node\ n \in A$  do
6      $st \leftarrow n.parent$ 
7     A: for  $item\ i \in (qv - anc)$  do
8       for  $x=st$  to  $\tau_c.root$  do
9         if  $i = x.item$  then
10           $bv \leftarrow$  set  $i^{th}$  bit
11           $st \leftarrow x$ 
12          goto label A
13        else  $st \leftarrow n.parent$ ; /* reset to anchor's parent */
14       $cv \leftarrow$  compute according to cost model CM
15       $branch\_cost \leftarrow bv.cv$ 
16       $MB \leftarrow$  compute set of maximum cost branches
17 return  $MB$ 

```

---

Focussing on Algorithm [getMaxCostBranches](#), Line 5 finds all the branches of  $\tau_c$  that contain the anchor item  $anc$  using the pointer links from the header table. Figure 7.6 shows all the occurrences of the anchor node in bold outlines. Lines 7, loops through each item in the query vector  $qv$ , except the  $anc$ . In our running example,  $qv$  is 2333, 3334 where  $anc$  is 3334. So we loop only through one item

= 2333 in line 7. Lines 8-13, search for this item in the branch starting from the parent node of the anchor (done to exclude anchor).

If the item is found, the  $i^{\text{th}}$  bit is set to 1 in a bit vector which corresponds to the position of the item in  $qv$ . This indicates that the item was found in the branch. When the item is found, the starting point is moved up, so the search space is shrunk for the search of the next item in  $qv$  in the branch. The search in the branch is stopped and the search for the next item in  $qv$  is resumed.

in our example. We form a *bit vector* ( $bv$ ) which indicates whether an element in  $qv - anc$  was found in the branch containing  $anc$ , with a 1, and 0 otherwise. Thus in our example, every branch has a bit vector of 1, except for the last branch which has  $anc$  3334 but no 2333, hence its bit vector computes to 0.

In Line 14, we compute a cost vector  $cv$  based on two simple cost models<sup>3</sup> as shown below.

- *Option 1: Assign increasing weights to segments that are closer to anchor.* The underlying idea being that knowledge of segments closer to the anchor might contribute more to the prediction than the ones further away.
- *Option 2: Assign decreasing weights to segments that are closer to anchor.* This is done to support the assumption that a moving object's source or origin might be more meaningful for a prediction. For example, a prediction query is executed when a vehicle is on a highway. Given the added knowledge that this vehicle's origin was a student accommodation, the probability of this vehicle taking the next exit to a university "X" is significantly increased.

To formulate a cost vector  $cv$  for  $qv - anc$  of " $l$ " elements where " $p$ " indicates the position of each element, we generate  $cv$  where each element is  $2^p$ . For example, if  $qv$  was  $\{a, b, c, d\}$  where " $d$ " was the anchor, then this would imply  $qv - anc$  would be  $\{a, b, c\}$  and  $l = 3$ . Then according to Option 2,  $cv$  would be calculated as  $\{2^2, 2^1, 2^0\}$ .

In Line 15, we finally calculate the cost as a scalar product of  $cv$  and  $bv$ . The branches that share the maximum cost are chosen and form the elements in  $MB$  and is returned.

Focussing our attention back to the main Algorithm [predict](#). In general the next few lines, after computing the set of maximum cost branches, deal with the task of computing the probabilities. In order to compute conditional probabilities, rules have to be *grown* in the form  $head \rightarrow tail$ , where in our case the *head* is the query vector  $qv$  and the *tail* is the resultant road segment where the moving object can be located in the future. We start by generating the rules and distributing the probability mass amongst the child nodes that are found below  $anc$  i.e. 3334. If some probability mass remains, then this indicates that the patterns did not cover all possible turns from the given segment (vertex) and hence we distribute the remaining probability mass amongst the rest of the possible *turns* (which will be explained in detail later on). If we are still left

---

<sup>3</sup> The cost models described are simplistic and further work needs to be done on arriving at more effective cost models to improve prediction accuracy

with probability mass that we couldn't distribute then its assumed that a vehicle has stopped its trajectory and hence this remaining probability is assigned to  $-1 * anc$  indicating that the car is parked at the anchor road segment.

Line 9, calls Algorithm `getPatternProbs` to calculate the probabilities from the child nodes under the *anc*.

---

**Algorithm `getPatternProbs`:** `getPatternProbs( lh,  $\tau_c$ , qv,  $B_{mcost}$ , pm )`

---

**Input** : Local hash table *lh*, Tree  $\tau_c$ , Query Vector *qv*, Set of maximum cost branches *MB* and probability mass *pm*

**Output**: (*rpm*, *lh*) = Remaining probability mass and local hash table

```

1 begin
2   rpm  $\leftarrow$  0
3   headSupport  $\leftarrow$  compute support for qv
4   totProb  $\leftarrow$  0
5   for pointer ptr  $\in$  MB do
6     for children c  $\in$  ptr do
7       if lookup( lh, c) not found then
8         hnode  $\leftarrow$  new node
9         hnode.prob  $\leftarrow$  pm * (supp(qv + c)/headSupport)
10        lh  $\leftarrow$  insert hnode
11        totProb  $\leftarrow$  totProb + hnode.prob
12      rpm  $\leftarrow$  pm - totProb
13  return (rpm, lh)

```

---

Focussing on Algorithm `getPatternProbs`, Lines 5-6 loop through each child node under the *anc* of each max cost branch that we calculated earlier on. Lines 7-11, ensure that we don't re-calculate the supports and probabilities for child nodes that we have already computed the conditional probabilities for.

Following our running example, keeping in mind the calculation for conditional probabilities, and referring to Table A.1 for total supports of each segment, Table 7.1 shows the probabilities and the remaining probability mass that are calculated for each unique child node below *anc* i.e. 3435 in the running example.

Table 7.1: Probabilities calculated from patterns

Rule	Calculation	Probability	Left Probability Mass
{2333, 3334} -> 3435	$1.0 * (20/50)$	0.4	0.6

These results are returned in the local key-value store *lh* and the remaining probability mass is also reduced to 0.6. This concludes the discussion of Algorithm `getPatternProbs` and we return to continue discussing the rest of the lines in Algorithm `predict`.

Lines 11-14 of Algorithm `predict`, distribute the remaining probability mass amongst the *turns* and parked vehicles. To compute the *turn probabilities*, we

follow exactly the same procedure as outlined in Algorithm [getPatternProbs](#), which computes probabilities from pattern extensions with the only difference being that we search for the anchor (Ex: 3334) in the *immediate child nodes* of the root node. If found, we iterate through the children and check them against *lh* and only pick the items that *do not* exist in *lh*. Items that exist in *lh* have already been *covered* by the patterns, and hence there is no need to assign a probability mass to the same segments from the turns. This can be thought of as a fallback mechanism which caters for the connecting road segments that might have been omitted due to the choice of a higher threshold support during mining.

In our example, we are left with a probability mass of 0.6. We find that the last branch which highlights our anchor 3334 in Figure [7.6](#) is an immediate child of the root node. It has child nodes 3435 and 3444. We have already assigned a probability mass to item 3435, hence we skip it. Table [7.2](#) shows the calculations for turn probabilities.

Table 7.2: Probabilities calculated from turn patterns

Rule	Calculation	Probability	Left Probability Mass
{2333, 3334} -> 3444	$0.6 * (10/50)$	0.12	0.48

The remaining probability mass is assigned to  $-1 * anc$ , because we assume that if a moving object cannot be found in the patterns or turns (at levels 1 and 2 under root node), then it must have concluded its trip and is parked at the anchor segment. Hence in our case,

$\{2333, 3334\} \rightarrow -3334 : \text{Probability} = 0.48$

At end of Line 14, *lh* now contains the values as shown in Table [7.3](#).

Table 7.3: Probabilities for single recursion level

Key: Item	Value: Probability
3435	0.40
3444	0.12
-3334	0.48

Lines 15-17 of Algorithm [predict](#), iterate through the values in *lh* and make a recursive call to *predict*. The newly calculated probability is passed as the remaining probability mass to distribute, the query vector *qv* is *extended* with the key (which is the new road segment) of *lh* and the remaining time is reduced by the key's global traversal time (from Table [A.1](#)). Note that the recursion continues to extend the query vector and reduces the time remaining and remaining probability mass to distribute and when the base condition is met i.e. time horizon  $t_c + \delta t$  is reached or exceeded, we exit with the final probabilities in the global hash table.

## CONTINUOUS QUERIES IN SCSQ

This chapter highlights the internals of the continuous queries with code snippets and sample runs from SCSQ. Figure 8.1, shows the high level *data flow diagram* that is implemented in SCSQ. The file with incoming location updates is read in for the *mining stream* and also for the *prediction stream*. The mining stream reads in the data with larger window sizes than the prediction stream. This is done in order to have longer patterns that will enable prediction for current trajectories that have shorter window spans. The functions and outputs will be discussed in more detail in the sections that follow. At a high level, all intermediate functions are implemented to read in a stream and produce their output as a stream for the next function down the line.

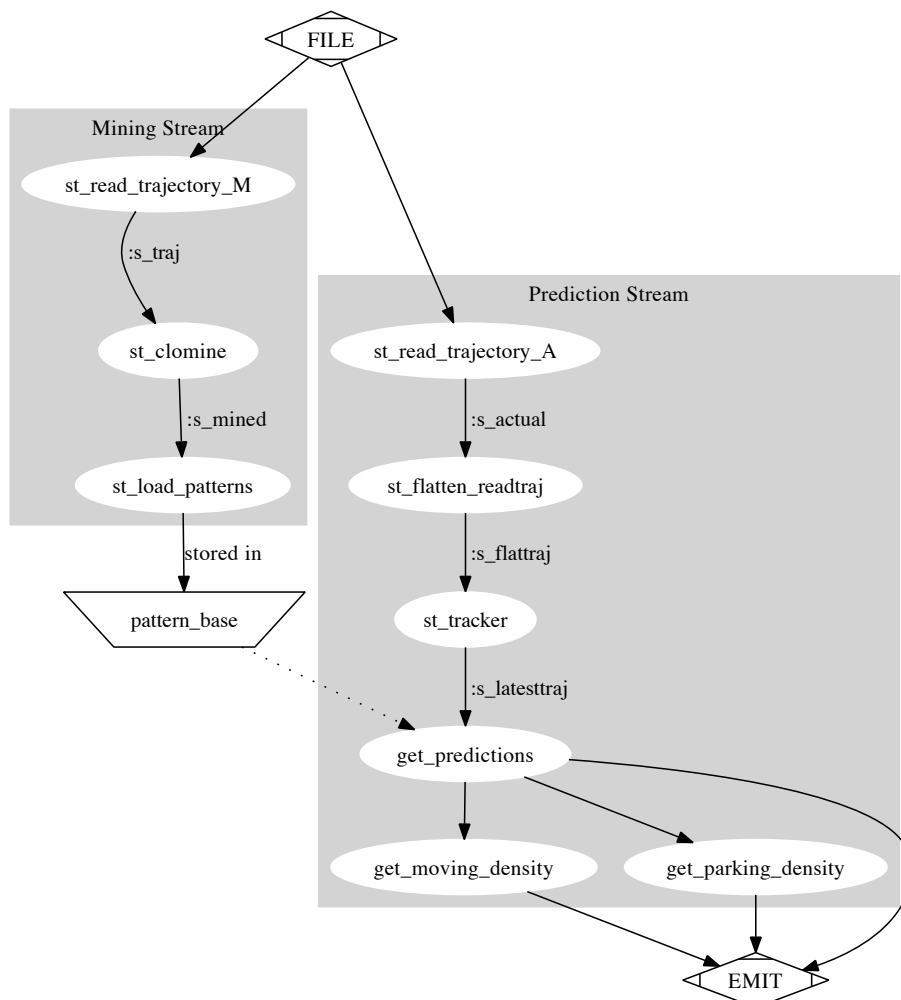


Figure 8.1: Functional Data Flow Diagram

## 8.1 READING AS STREAM

`st_read_trajectory()` function takes an `fname` filename which has the tuples coming in from the client, window related parameters for the stream - namely the window size and the stride along with two more optional boolean parameters which control the suppression of nil windows. The code snippet below displays the signature of this function. The output of this function is a Stream of Vector, which we store in the temporary variable `:s_traj` to pass on to other functions.

---

```
1 create function st_read_trajectory( Charstring fname,
2                                     Real sz,
3                                     Real stride,
4                                     Integer a,
5                                     Integer b )
6 -> Stream of Vector stv
7 as streamof(twinagg(streamof(readfile(fname)), sz, stride, a,b));
```

---

Listing 8.1: Reading in tuples as a stream in SCSQ

Sample usage is shown below. We assign the stream to `:s_traj` and extract its contents. The output is a vector of vectors, where the first component is a vector of  $t_{start}$  and  $t_{end}$  window boundary timestamps and the second component is a vector of vectors where each vector holds the  $t_{sub}$  (submission timestamp), Vehicle ID, Road Segment ID and  $\delta t$ .

---

```
[myscsq] 1> set :s_traj = st_read_trajectory('../.. / data/test_data.txt',
                                           3600.0,3600.0,0,0);

[myscsq] 1> in(:s_traj);

{{|2011-01-01/01:00:00|,|2011-01-01/02:00:00|},
 {
  {|2011-01-01/01:00:00|,10,763,0.3},
  {|2011-01-01/01:08:23|,60,-1,1.1}
 }
}
```

---

Listing 8.2: Sample usage of `st_read_trajectory` in SCSQ

## 8.2 CLOSED SEQUENCE MINING OF STREAM

`st_clomine()` function takes a Stream of vectors as its first argument (for example, `:s_traj`). The second argument is the minimum threshold support percentage that is needed for the mining algorithm ranging from 0.0 to 100.0. The third and last argument is the minimum length of output patterns that should be outputted. The closed sequence mining algorithm is developed as a foreign function interface to SCSQ written in C.

---

```

1 create function st_clomine( Stream of Vector str,
2                             Real minsupp,
3                             Integer minlen )
4 -> Stream of Vector stv
5 as streamof(
6   select vector(w,sts,ets,s,pv,tv)
7   from   Integer w, Timeval sts, Timeval ets, Integer s,
8         Vector of Integer pv, Vector of Real tv
9   where (w,sts,ets,s,pv,tv) = clomine(str,minsupp,minlen));

```

---

Listing 8.3: Closed sequence mining of stream

When extracted from the stream output, the tuples are of the following format in order of column number; Window ID,  $t_{start}$ ,  $t_{end}$ , Support of the closed pattern, Closed pattern vector  $pv$ , and the traversal time vector  $tv$  corresponding to  $pv$  as shown in Listing 8.4.

---

```

[myscsq] 1> set :s_mined = st_clomine( :s_traj, 30.0, 1);
[myscsq] 1> in(:s_mined);

(0,|2011-01-01/01:00:00|,|2011-01-01/02:00:00|,3,{10085},{200.0})
(0,|2011-01-01/01:00:00|,|2011-01-01/02:00:00|,5,{10084},{100.1})
(0,|2011-01-01/01:00:00|,|2011-01-01/02:00:00|,4,{10083},{1.1})

```

---

Listing 8.4: Sample usage of *st\_clomine* in SCSQ

### 8.3 STORING CLOSED PATTERNS

`pattern_base()` is the definition of a stored function in Amos II. It is composed of a function that takes as arguments the boundary timestamps of the window i.e.  $t_{start}$  and  $t_{end}$ . It returns the closed mining results as tuples of  $\langle pv, tv, support \rangle$ .

`st_load_patterns()` is the function that takes as an argument the stream that is output from `st_clomine`. In our example, `:s_mined` is the stream used as input.

---

```

1 create function pattern_base( Timeval sts,
2                             Timeval ets )
3 -> <Vector of Integer pv, Vector of Real tv, Integer s>
4 as stored;
5
6 create function st_load_patterns( Stream of Vector stv )
7 -> Boolean
8 as add pattern_base( sts, ets ) = (pv,roundto(tv,2),s)
9   from Integer w,Timeval sts, Timeval ets, Integer s,
10        Vector of Integer pv, Vector of Real tv
11   where {w,sts,ets,s,pv,tv} in (stv);

```

---

Listing 8.5: Storing closed patterns

The sample usage illustrates an example of `load_patterns()` and then being able to view the output using the function `view_pat_summary()` which is defined as a utility function and defined in the appendix.

---

```

/* To load closed patterns */
[myscsq] 1> st_load_patterns(:s_mined);

/* To view a summary of stored patterns */
[myscsq] 1> view_pat_summary();

({|2011-01-01/01:00:00|,|2011-01-01/01:02:00|},1323)
({|2011-01-01/01:00:05|,|2011-01-01/01:02:05|},1401)
({|2011-01-01/01:00:10|,|2011-01-01/01:02:10|},1420)
({|2011-01-01/01:00:15|,|2011-01-01/01:02:15|},1469)

```

---

Listing 8.6: Sample usage of `st_load_patterns` in SCSQ

#### 8.4 EXTRACT THE VEHICLE'S LATEST TRIP

`st_tracker()` implements *flattening* the vector output from `st_read_trajectory()` and grouping it by vehicle ID. It takes as an argument `:s_actuals` in our example which is a new stream generated using `st_read_trajectory()` but only this time notice that the window size argument is much smaller than the window size we used for the mining stream. This is done in order so that the mining window contains *longer patterns* than the actual trip in the prediction stream's window and hence can cater for the additional future time  $\delta t$ .

The use of "*vectorof*" in line 12, converts these collated road segments into a vector which represents all the segments that the vehicle traversed within the time window. In line 4, the use of `latest_traj()` ensures that only the latest trip is returned. This is done because within a given time window a vehicle can start and stop multiple times and we are only interested in making a prediction for the trip that is still ongoing.

---

```

1 create function st_tracker( Stream of Vector stv )
2 -> Stream of Vector
3 as streamof(
4   select vector(t1, t2, v, latest_traj(rev_vec(svec)))
5   from   Timeval t1, Timeval t2, Integer v, Vector of Integer svec
6   where {{t1,t2,v}, svec} in
7         (sort(groupby((
8           select {t1,t2,v}, s
9           from   Timeval t1, Timeval t2,
10              Integer v, Integer s
11              where {t1,t2,v,s} in (stv)),
12             #'vectorof'))));

```

---

Listing 8.7: Extraction of vehicle trips

In the usage below it becomes apparent how the vehicle trajectories are collated as a vector (shown in last column).

---

```

/* Read in actual trajectory */
[myscsq] 1> set :s_actuals = st_read_trajectory('.././ data/test_data.txt',
      20.0, 5.0,0,0);

/* Flatten the stream of vector of vectors */
[myscsq] 1> set :s_flattraj = st_flatten_readtraj(:s_actuals);

/* Find the latest moving trip of each vehicle */
[myscsq] 1> set :s_latesttraj = st_tracker(:s_flattraj);

/* Extract */
[myscsq] 1> in( :s_latesttraj );

(|2011-01-01/01:00:00|,|2011-01-01/01:06:00|,10,{13413,348})
(|2011-01-01/01:00:00|,|2011-01-01/01:06:00|,20,{805,10084,349})
(|2011-01-01/01:00:00|,|2011-01-01/01:06:00|,30,{11653,12101,983})

```

---

Listing 8.8: Sample usage of *st\_tracker* in SCSQ

## 8.5 FIND THE MINING WINDOW FOR TRAJECTORY

`get_mining_window_sql()` expects a stream of the latest trips of every vehicle (e.g. `:s_latesttraj`). and an additional argument which is the future time duration in seconds for which we would like a prediction (e.g. 5 minutes). The function goes through each vehicle's trips, and for each of these vehicles it attempts to find the mining window that corresponds to the boundary window of the vehicle's latest trip. Note that while finding this mining window the utility function `find_the_mwindow()` must check for timestamp overlaps between the mining results stored in `pattern_base()` and the bounding window of the vehicle's latest trip plus the future time argument. This is done in order to be able to capture patterns that extend into the future. These patterns will form the foundation for the prediction algorithm.

---

```

1 create function get_mining_window_sql( Stream of Vector stv,
2                                     Real ftime)
3 -> <Vector of Timeval st, Vector of Timeval en, Vector>
4 as select st, en, {veh,qv}
5   from   Timeval at1,
6          Timeval at2,Integer veh,Vector of Integer qv
7   where  {at1,at2,veh,qv} in (stv)
8   and    (st,en) = find_the_mwindow( at1, at2, ftime );

```

---

Listing 8.9: Finding mining window

The output is a bag of vector of vectors where the first component vector is the start and end timestamps of the trip trajectory, the second component vector the start and end timestamps of the chosen mining window, followed finally by a vector of the Vehicle ID and latest trip/trajectory that the vehicle has traced so far.

---

```
[myscsq] 1> get_mining_window_sql(:s_latesttraj, 10.0);

(
  {|2011-01-01/01:00:00|,|2011-01-01/01:00:20|},
  {|2011-01-01/01:00:00|,|2011-01-01/01:02:00|},
  {10030,{1177}}
)
```

---

Listing 8.10: Sample usage of *get\_mining\_window\_sql* in SCSQ

## 8.6 PREDICT

*get\_predictions()* expects a stream like *:s\_latesttraj* . In addition, it takes the future time for prediction as an argument. For each vehicle in the time windows, the foreign function "predict\_flocn", which is an external C function, is called to apply the prediction model. The prediction model is described in more detail in Chapter 7.

---

```
1 create function get_predictions( Stream of Vector stv,
2                               Real ftime )
3 -> <Timeval at1, Timeval at2, Integer veh,
4   Vector of Integer qv, Integer seg, Real prob>
5 as select at1, at2, veh, qv, seg,prob
6   from   Timeval mt1, Timeval mt2
7   where  ({at1,at2}, {mt1,mt2}, {veh,qv})
8         = get_mining_window_sql( stv, ftime)
9   and    (seg,prob)
10        = predict_flocn({pattern_base(mt1,mt2)},qv,veh,ftime)
11   and    mt1 != "NIL"
12   and    mt2 != "NIL";
```

---

Listing 8.11: Prediction

The output of this function is a probability of where a vehicle at time  $t_c$ , can be at a future point in time for example  $t_c + 5$  minutes . In the example the output shows the trajectory window's timestamps, the vehicle ID, the current trip that was used to query the stored knowledge base to predict, the resultant road segment where the vehicle can be and lastly the probability that the vehicle will be at the road segment. So in our example, we see that vehicle ID 10039 traversed on road segment 10173 and exactly after 5.0 minutes it can be on road segment 10112 with a 70% probability and road segment 179 with a 30% probability. Note that the probabilities for each vehicle must add up to 1.0.

---

```
[myscsq] 1> get_predictions(:s_latesttraj, 300.0);

(|2011-01-01/01:00:00|,|2011-01-01/01:00:20|,10039,{10173},10112,0.7)
(|2011-01-01/01:00:00|,|2011-01-01/01:00:20|,10039,{10173},179,0.3)
```

---

Listing 8.12: Sample usage of *get\_predictions* in SCSQ

## 8.7 PREDICT MOVING OBJECT DENSITY

`get_moving_density()` expects the same arguments as `get_prediction()`. This function computes the total number of moving vehicles by summing up the probability masses of moving vehicles on each road segment.

---

```
1 create function get_moving_density( Stream of Vector stv,
2                                     Real ftime )
3 -> <Vector, Real>
4 as (groupby(
5     (select {at1, at2, seg}, prob
6         from Timeval at1, Timeval at2,
7             Integer seg, Real prob,
8             Integer v, Vector of Integer vec
9         where (at1,at2,v,vec,seg,prob)
10              = get_predictions_proc(stv,ftime)
11         and   seg > 0
12         ), #'sum'));
```

---

Listing 8.13: Prediction of future moving object density on road network

The output displays that at  $t_c + 5.0$  minutes there will be 4 objects moving on road segment 11869 and 15.5 objects moving on road segment 10606.

Note that positive road segments output from the prediction model indicate moving objects while negative segments indicate that the objects have stopped moving and are parked.

---

```
[myscsq] 1> get_moving_density( :s_latesttraj, 300.0);

({|1970-01-01/01:00:30|,|1970-01-01/01:00:50|,11869},4.0)
({|1970-01-01/01:00:50|,|1970-01-01/01:01:10|,10606},15.5)
```

---

Listing 8.14: Sample usage of `get_moving_density` in SCSQ

## 8.8 PREDICT PARKED OBJECT DENSITY

`get_parking_density()` expects the same arguments as `get_predictions()`. This function computes the total number of stopped/parked vehicles by summing up the probability masses of stopped vehicles on each road segment.

---

```
1 create function get_parking_density( Stream of Vector stv,
2                                     Real ftime )
3 -> <Vector, Real>
4 as (groupby(
5     (select {at1, at2, seg}, prob
6         from Timeval at1, Timeval at2,
7             Integer seg, Real prob,
8             Integer v, Vector of Integer vec
9         where (at1,at2,v,vec,seg,prob)
10              = get_predictions_proc(stv,ftime)
11         and   seg < 0
12         ), #'sum'));
```

---

Listing 8.15: Prediction of future parked object density on road network

The output displays that at time  $t_c + 5.0$  minutes, there will be approximately 76 objects parked on road segment 100 and 14 objects parked on road segment 1001. Valuable information to car parks in the vicinity to perform more optimal capacity planning.

---

```
[myscsq] 1> get_parking_density( :s_latesttraj, 300.0);  
  
({|1970-01-01/01:02:05|,|1970-01-01/01:02:25|,-100},76.32)  
({|1970-01-01/01:00:40|,|1970-01-01/01:01:00|,-1001},14.28)
```

---

Listing 8.16: Sample usage of *get\_parking\_density* in SCSQ

## EXPERIMENTS

---

The following sections describe the set of experiments performed and the results obtained. The actual experiments were conducted on a laptop with Windows 7.0 and an Intel Core 2 Duo 2.2 GHz processor, 2MB L2 Cache and 4GB memory.

Section 9.1 describes the rea-world data set. Sections 9.2, 9.3 and 9.4, evaluate different aspects of the proposed frequent route mining method, and Section 9.5 evaluates the accuracy of the proposed frequent route based location and network density prediction methods.

### 9.1 REAL-WORD DATA SET

The data used to evaluate the proposed method includes the GPS readings of 1,500 taxis and 400 trucks traveling on the streets of Stockholm during the course of a full day. Each taxi produces a reading once every 60 seconds approximately. This reading includes only taxi identification and location information. Taxis produce readings less frequently when they are not carrying any passengers. Trucks use more recent and more accurate GPS devices that produce readings once every 30 seconds and include identification, location, speed and heading information. This knowledge about the sampling frequencies has been used to identify approximately 17,000 trips. Road network based trajectories have been constructed by using a modified road network representation that had approximately 6,000 directional segments with average segment length of 55 meters.

As the server-side of the architecture assumes a stream of continuously evolving network trajectories of moving objects as input, and as the focus of the current thesis is the development of the server side components of the architecture, the following paragraph briefly highlights the features of- and the techniques used in the pre-processing of the trajectories at the client-side as follows.

The location measurements obtained from GPS can often be noisy and innacurate. The process of converting a sampled trajectory in euclidean space to a road network based trajectory involves two processes. First, through a process called *map matching* the noisy GPS signals are aligned to the nearest locations on the road network. This is done in an online fashion which takes as inputs the object's recent movement history and the road network's geometry. Second, in our case of a road network based movement model, an *interpolation* is performed between two map matched locations where the shortest possible path on the road network between the two points is assumed. Finally, a stream of timestamped temporally annotated road segments is outputted for the server-side to process, where the times denote the traversal time of each road segment.

## 9.2 STREAM PROCESSING ANALYSIS

The performance of closed frequent sequence route mining is compared with variations in regards to mining frequency and time window sizes while keeping some of the parameters constant in adherence to the constraints levied by the data set.

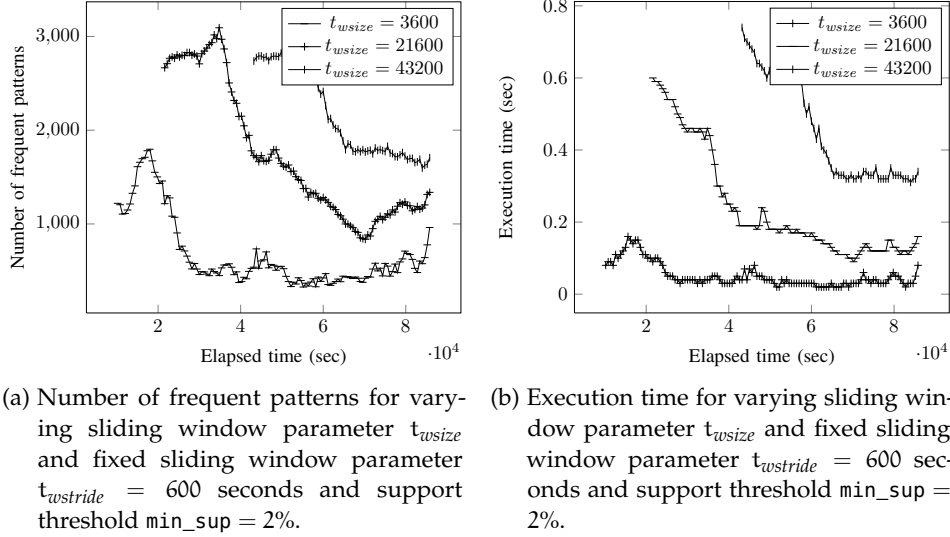


Figure 9.1: Mining analysis.

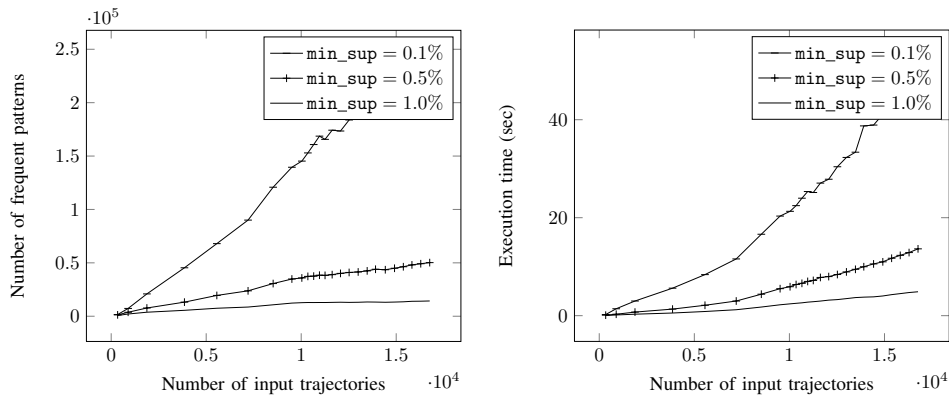
The first group comprises the graphs in figures 9.1a and 9.1b to measure pattern counts and execution time in seconds respectively. Minimum support is fixed at 2% and the sliding window parameter  $t_{wstride}$  is fixed at 600 seconds (10 mins) in the graph to represent a fine-grained mining interval.

Both graphs show an increase in the number of *closed frequent route sequences* as their sliding window sizes  $t_{wsize}$  are gradually increased. The peaks in the graph coincide with the peak hour traffic and suggest that maximum traffic occurs at about 0200 to 0400 hrs and at 1900 hrs (after converting elapsed time in seconds to clock times). The execution times are continuously measured to study the feasibility of the incremental mining approach given that the runtime of the mining per interval has an upper bound which is  $t_{wstride}$  (the size of a single sliding window).

Note that the initial mining starts after we have one complete sliding window, and hence as we increase  $t_{wsize}$  in our plots the starting points of the plots are right-shifted accordingly.

## 9.3 SCALABILITY ANALYSIS

The following set of experiments were conducted with the aim of measuring the scalability of the mining solution.



(a) Scaling behavior: Number of frequent patterns for varying input sizes (number of input trajectories), different  $\text{min\_sup}$  values, and fixed sliding window parameter  $t_{wsize} = 86400$  seconds (24 hours). (b) Scaling behavior: Execution time for varying input sizes (number of input trajectories), different  $\text{min\_sup}$  values, and fixed sliding window parameter  $t_{wsize} = 86400$  seconds (24 hours).

Figure 9.2: Scalability analysis.

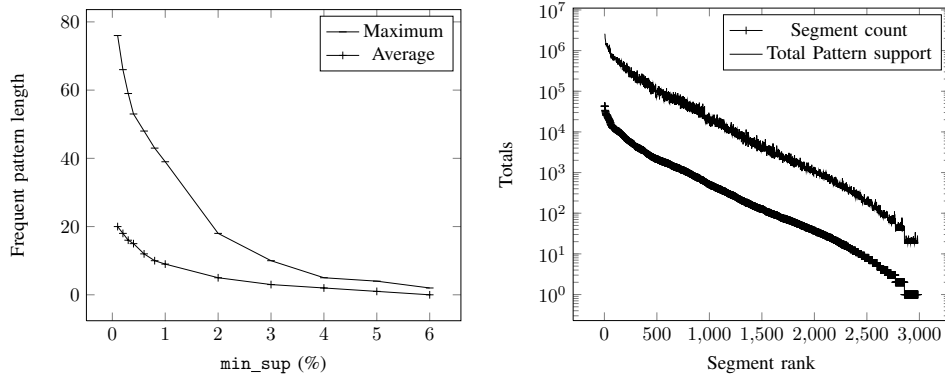
Increasingly large volumes of simultaneously moving object trajectories were simulated by fixing the sliding window parameter  $t_{wsize}$  to a 24hr period for the sample data and increasing the window stride  $t_{wstride}$  in significantly large steps to accommodate for a large number of input trajectories and hence simulating traffic inputs from a large number of simultaneously moving objects in the road network.

As suggested by the results in figures 9.2a and 9.2b The execution time for nearly 17K input trajectories mined at a minimum support of 0.1% was approximately 40 seconds with approximately 25K closed frequent patterns mined. The results pointed in the direction of a very robust and highly scalable solution for mining long patterns.

#### 9.4 QUALITATIVE PATTERN ANALYSIS

The following set of experiments were conducted on the pattern outputs generated in a *one-day* long sliding window in order to assess the quality of the patterns. Figure 9.3a displays the maximum and average closed frequent pattern lengths derived when the minimum support percentage is varied from 0.1% to 1.0% in increments of 0.1 followed by increments of 1.0 upto 6% which is the cut-off minimum support percentage after which no patterns appear in the dataset. The *longest/maximum pattern of length 76* is detected for the lowest support of 0.1% with an average length of 20.

Figure 9.3b displays statistics at the segment level. After pattern generation, a post-processing analysis step generated statistics per segment such as the count of closed frequent patterns which included a particular segment and the sum total of the supports of the patterns in which the segment was discovered. An important point to note is that the segments are sorted in decreasing order of count / support-sum magnitude and each segment gets assigned a rank.



(a) Distribution statistics for frequent pattern length for varying `min_sup` values and fixed sliding window parameter  $t_{wsize} = 86400$  seconds (24 hours). (b) Segment occurrence counts and support total sorted in decreasing order of magnitude

Figure 9.3: Qualitative pattern analysis.

The results displayed on a logarithmic y-axis clearly indicate the interesting segments which are frequently traversed at the start of the graph on the extreme left.

Knowledge of such segments aids spatial visualization of frequently traversed sections of the road network. A spatial trend analysis on the road networks can reveal valuable information, for instance patterns with very high support represent traffic on a busy highway and might deduce some trivial knowledge such as a vehicle travelling on a freeway is likely to remain on it for a while, on the other hand patterns leading “in or out of” these high support routes could convey more meaningful information such as the trend of vehicle routing when moving towards or away from the central parts of a city or the variations that could be introduced on such a trend with the introduction of a new spatial object like a new casino on the boundary of the city center or even a temporal event such as a launch of a new product in a shop or a rather unfortunate freeway accident. Although there are many such examples and studies, spatio-temporal data mining is in its infancy stages and more methods and applications of spatial classification and trend analysis, especially those associated with time, must be further explored.

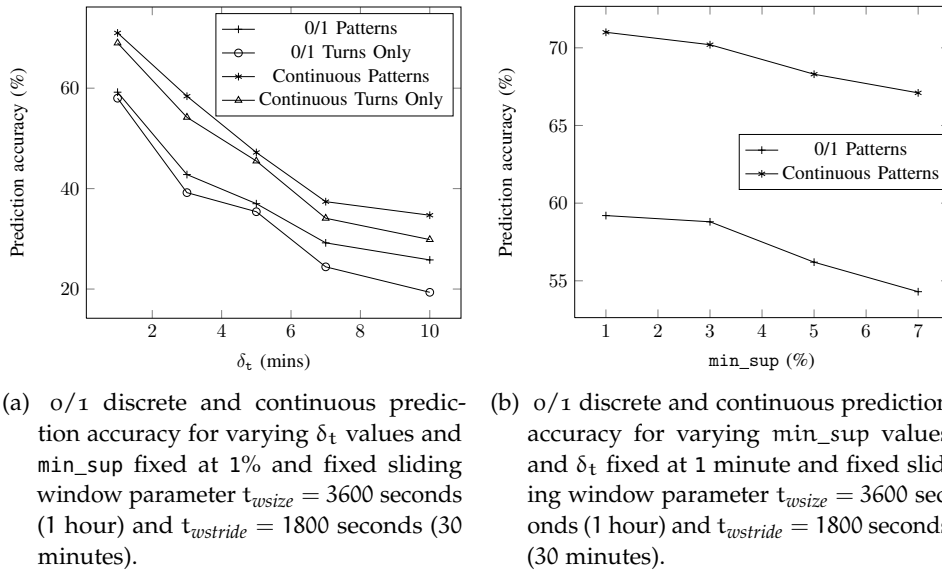
## 9.5 PREDICTION ACCURACY ANALYSIS

The following set of experiments were conducted in order to measure the prediction accuracy approach of this thesis. The experiments could be broadly classified as experiments that measured the average prediction accuracy of all moving objects as shown in 9.4 vs. experiments that measured the sum of squared error (SSE) of the road network density as shown in 9.5. We study the effects on prediction accuracy (or SSE in case of network density experiments) by gradually incrementing either the future point in time for location prediction

i.e. " $t_c + \delta_t$ " while keeping the minimum support threshold  $\text{min\_sup}$  fixed and vice versa.

Furthermore, where applicable, the experiments compare the prediction accuracy gained from mined patterns (inclusive of all possible turns at junctions) vs. prediction accuracy that is achieved by using just the single turn statistics at junctions. Prediction accuracy is further subdivided into two kinds of accuracy calculations - namely *o/1 discrete accuracy* where the road segment with the highest location probability of the moving object at time  $t_c + \delta_t$  is compared against the *actual road segment where the object is located* at time  $t_c + \delta_t$  and only when the road segment IDs match then we assign a 1, else we assign 0 to accuracy. The *continuous accuracy* approach compares the *actual road segment location* at time  $t_c + \delta_t$  to the predicted road segments and when a matching road segment is found then the probability percentage on that road segment is added and hence contributes to the accuracy measurement. For example, if the actual location was at segment id 1000, and the prediction had assigned 30% probability to segment 1000 and 70% to segment 2000, then the *o/1 discrete accuracy* approach considers this as a zero contribution to accuracy while the *continuous accuracy* approach considers this a 0.3 contribution to accuracy.

Focussing on Figure 9.4a, the continuous accuracy approach attributes a higher prediction accuracy than the *o/1 discrete* approach. Since  $\text{min\_sup}$  is fixed at 1% we mine longer patterns which culminate in better prediction accuracy over an approach that utilizes only turn statistics. This is indicated by the widening gaps between the two approaches especially at values of  $\delta_t$  between 7 and 10 minutes. In one minute in the future, the continuous prediction accuracy reaches a maximum of 71% accuracy.



(a) *o/1* discrete and continuous prediction accuracy for varying  $\delta_t$  values and  $\text{min\_sup}$  fixed at 1% and fixed sliding window parameter  $t_{wsize} = 3600$  seconds (1 hour) and  $t_{wstride} = 1800$  seconds (30 minutes).  
 (b) *o/1* discrete and continuous prediction accuracy for varying  $\text{min\_sup}$  values and  $\delta_t$  fixed at 1 minute and fixed sliding window parameter  $t_{wsize} = 3600$  seconds (1 hour) and  $t_{wstride} = 1800$  seconds (30 minutes).

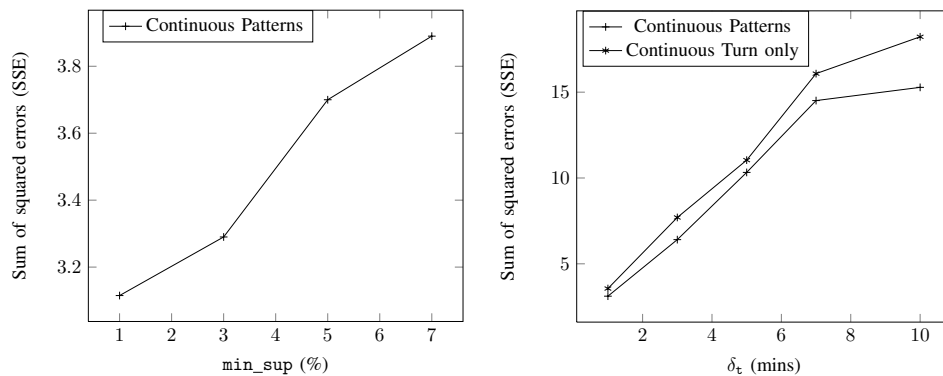
Figure 9.4: Prediction accuracy analysis for individual moving objects averaged over stream windows.

In Figure 9.4b,  $\delta_t$  is kept fixed at 1 minute and  $\text{min\_sup}$  is varied from 1% to 7%. This range is chosen because exceeding the  $\text{min\_sup}$  of 7% gives no patterns and hence prediction accuracy falls back on purely turn statistics and the plot

produces a flat line for  $\text{min\_sup} = 7\%$  and higher which is uninteresting. We notice that the prediction accuracy drops by approximately 5% for both discrete and continuous accuracy calculation methods.

In Figure 9.5a, the average number of moving objects per window of the stream is approximately 270. The SSE measurement between actual and predicted density of moving objects on the entire road network highlights the fact that our prediction accuracy is dropping, which results in an increase in the SSE value from 3.2 to 4.1 with incremental changes in  $\text{min\_sup}$ . This plot is an *aggregation* of errors noticed in 9.4a.

In Figure 9.5b, we notice that as the time horizon " $t_c + \delta_t$ " is gradually incremented, the gap between the SSE of mined patterns vs. pure turn statistics widens, indicating that the network density calculation which uses long patterns actually outperforms the approach which relies on short term turn statistics only.



- (a) Continuous network density prediction SSE for varying  $\text{min\_sup}$  values and  $\delta_t$  fixed at 1 minute and fixed sliding window parameter  $t_{wsize} = 3600$  seconds (1 hour) and  $t_{wstride} = 1800$  seconds (30 minutes).
- (b) Continuous network density prediction SSE for varying  $\delta_t$  and  $\text{min\_sup}$  fixed at 1% and fixed sliding window parameter  $t_{wsize} = 3600$  seconds (1 hour) and  $t_{wstride} = 1800$  seconds (30 minutes).

Figure 9.5: Prediction Error analysis for road network density averaged over stream windows.

## CONCLUSIONS AND FUTURE WORK

---

Motivated by the accelerating need for effective traffic prediction and management systems and the new possibilities for such systems allowed by GPS-enabled mobile devices, this thesis presented a novel approach to using moving object trajectories for traffic prediction/management. The approach was based on realistic real-world application requirements, and performed online management of current trajectories of moving objects in road networks.

The trajectories considered were *incrementally evolving*, i.e., current, on-going trajectories were incrementally delivered in near-real-time to the central server. The approach performed on-line, incremental mining of closed frequent routes of the moving objects. The closed frequent routes were later used for the prediction of the *near-future-locations* of the moving objects, based on the current object trajectories and historical frequent routes. The thesis proposed a number of concrete methods and data structures for managing and mining closed frequent routes from incrementally evolving trajectories of moving objects in road networks. The approach was empirically evaluated on a large real-world data set of moving object trajectories, originating from a fleet of taxis.

In future work, several areas are very interesting.

First, more effective and complex cost models have to be devised which for example take into account the *discriminative frequent routes* to weight the individual trajectory's road segments prior to prediction. Second, many more interesting CQs can be formulated to answer questions like "*Given a road segment  $x$ , which vehicles will arrive here in exactly 10 minutes from now?*" (useful to alert drivers of a situation at the road segment) or "*What patterns in the traffic lead to congestion?*". Third, more work can be done on *visualization* of the road networks to monitor the most frequent routes, also called "*hot routes*" and viewing the actual and predicted traffic movements in real-time. Finally, the developed methods should be deployed and tested in a large-scale real-world scenario to assess the possible gains with respect to traffic management.

Part I

APPENDIX

APPENDIX

---

## A.1 TABLE OF ROAD SEGMENT'S GLOBAL AVERAGE TRAVERSAL TIME AND SUPPORT

Table A.1: Singleton road segments with corresponding global traversal time and support

<b>Segment/Item</b>	<b>Traversal Time</b>	<b>Support</b>
1222	1.00	30
3444	2.00	20
3435	3.00	20
3332	4.00	30
3242	5.00	10
3231	6.00	20
3334	7.00	50
2524	8.00	25
2423	9.00	35
2333	10.00	80
2223	11.00	45
2122	12.00	15
1424	13.00	10

## BIBLIOGRAPHY

---

- [1] Ieee icdm contest: Tomtom traffic prediction, 2010. URL <http://www.hitachi-automotive.co.jp/en/products/cis/03.html>.
- [2] Hitachi automative systems. traffic information service/solution, 2010. URL <http://tunedit.org/challenge/IEEE-ICDM-2010>.
- [3] C.C. Aggarwal and D. Agrawal. On nearest neighbour indexing of nonlinear trajectories. *PODS*, pages 252–259, 2003.
- [4] R. Agrawal and R. Srikant. Mining sequential patterns. *Proc. of ICDE*, pages 3–14, 1995.
- [5] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. *Mobile Data Management*, pages 3–14, 2001.
- [6] A. Brilingaitė. *Location-related context in mobile services*. PhD thesis, Aalborg University, Denmark, 2006.
- [7] A. Brilingaitė and C.S. Jensen. Enabling routes of road network constrained movements as mobile service context. *GeoInformatica*, pages 55–102.
- [8] J. Chen, D. DeWitt, F. Tian, and Y. Wang. Niagaracq: A scalable continuous query system for internet databases. *In Proc. ACM International Conference on Management of Data*, pages 379–390, 2000.
- [9] G. Gidofalvi and E. Saqib. From trajectories of moving objects to route-based traffic prediction and management. *GIScience*, 2010.
- [10] G. Gidofalvi, T.B. Pedersen, T. Risch, and E. Zeitler. Highly scalable trip grouping for large scale collective transportation systems. *Proc. of International conference on extending database technology, EDBT*, March 2008. URL <http://user.it.uu.se/~udbl/publ/TaxiEDBT2008.pdf>.
- [11] A.C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M.J. Strauss. Quicksand: Quick summary and analysis of network data, December 2001. URL [citeseer.nj.nec.com/gilbert01quicksand.html](http://citeseer.nj.nec.com/gilbert01quicksand.html).
- [12] G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. *Workshop on Frequent Itemset mining implementations (FIMI)*, 2003.
- [13] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V.J. Tsotras. Online discovery of dense areas in spatio-temporal databases. *SSTD*, 2003.
- [14] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD*, pages 1–12, 2000.
- [15] C.S. Jensen, T.B. Pedersen, L. Speicys, and I. Timko. Data modeling for mobile services in the real world. *Proc. of SSTD*, pages 1–9, 2003.

- [16] C.S. Jensen, D. Lin, and B.C Ooi. Query and update efficient b+-tree based indexing of moving objects. *VLDB*, pages 768–779, 2004.
- [17] C.S. Jensen, D. Lin, B.C Ooi, and R. Zhang. Effective density queries on continuously moving objects. *ICDE*, page 71, 2006.
- [18] H. Jeung, Q. Liu, H.T. Shen, and X. Zhou. A hybrid prediction model for moving objects. *ICDE*, pages 70–79, 2008.
- [19] H. Jeung, M.L. Yiu, X. Zhou, and C.S. Jensen. Path prediction and predictive range querying in road network databases. *VLDB*, pages 585–602, 2010.
- [20] S. Kim, J. Won, J. Kim, M. Shin, J. Lee, and H. Kim. Path prediction of moving objects on road networks through analyzing past trajectories. *KES*, pages 379–389, 2007.
- [21] H. Kriegel, M. Renz, M. Schubert, and Zuefle A. Statistical density prediction in traffic networks. *8th SIAM Conf. on Data Mining (SDM'08), Atlanta, GA, USA*, 2008.
- [22] L. Liu, C. Pu, and W. Tang. Continual queries for internet-scale event-driven information delivery. *IEEE Transaction, Knowledge and Data Engineering*, pages 610–628, 1999.
- [23] J.M. Patel, Y. Chen, and V.P. Chakka. Stripes: an efficient index for predicted trajectories. *SIGMOD*, pages 635–646, 2004.
- [24] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DKMD)*, pages 21–30, 2000.
- [25] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. Mining access patterns efficiently from web logs. *Proc. of PAKDD*, pages 396–407, 2000.
- [26] J. Pei, J. Han, B. Mortazavi-Asl, and H. Pinto. Prefixspan: mining sequential patterns efficiently by prefix-projected pattern growth. *ICDE*, pages 214–224, 2001.
- [27] F. Pinelli, A. Monreale, R. Trasarti, and F. Giannotti. Location prediction within the mobility data analysis environment deaedralus. *MobiQuitous*, 2008.
- [28] K.-F. Richter. A uniform handling of different landmark types in route directions. *International Conference on spatial Information Theory (COSIT)*, 2007.
- [29] T. Risch. Alisp v2 user's guide, Feb 2006. URL <http://user.it.uu.se/~torer/publ/alisp2.pdf>.
- [30] T. Risch. Amos 2 external interfaces, January 2001. URL <http://user.it.uu.se/~torer/publ/external.pdf>.
- [31] T. Risch and V. Josifovski. Distributed data integration by object-oriented mediator servers. *Concurrency and Computation: Practice and Experience*, 13(11): 933–953 (2001), 2001.

- [32] T. Risch, V. Josifovski, and T. Katchaounov. Amos 2 concepts, June 2000. URL [http://www.it.uu.se/research/group/udbl/amos/doc/amos\\_concepts.html](http://www.it.uu.se/research/group/udbl/amos/doc/amos_concepts.html).
- [33] T. Risch, V. Josifovski, and T. Katchaounov. Functional data integration in a distributed mediator system. *Functional Approach to Computing with Data*, Springer, ISBN 3-540-00375-4, 2004., 2004.
- [34] S. Saltenis and C.S. Jensen. Indexing of moving objects for location-based services. *ICDE*, pages 463–472, 2002.
- [35] S. Saltenis, C.S. Jensen, S.T. Leutenegger, and M.A. Lopez. Indexing the positions of continuously moving objects. *SIGMOD*, pages 331–342, 2000.
- [36] R. Shekhar, C.T. Lu, S. Chawla, and P. Zhang. Data mining and visualization of twin-cities traffic data. Technical report, Dept of CSE, University of Minnesota, 2000.
- [37] R. Srikant and R. Agrawal. Mining sequential patterns: generalizations and performance improvements. *EDBT*, pages 3–17, 1996.
- [38] M. Sullivan and A. Heybey. Tribeca: A system for managing large databases of network traffic. *Proc. of USENIX Annual Technical Conference*, 1998.
- [39] Y. Tao, D. Papadias Faloutsos, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. *SIGMOD*, pages 611–622, 2004.
- [40] T. Uno, T. Asai, Y. Uchida, and H. Arimura. Lcm: An efficient algorithm for enumerating frequent closed itemsets. *Proc of workshop on frequent itemset mining implementations (FIMI)*, 2003.
- [41] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver 2: Efficient mining algorithms for frequent/closed/maximal itemsets. *Proc of workshop on frequent itemset mining implementations (FIMI)*, 2004.
- [42] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. *ICDE*, 2004.
- [43] M.J. Zaki and C.J. Hsiao. Charm: An efficient algorithm for closed itemset mining. *SDM*, 2002.
- [44] E. Zeitler and T. Risch. Scalable splitting of massive data streams. *Proc of Database Systems for Advanced Application, DASFAA*, 1-4 April 2010. URL <http://user.it.uu.se/~udbl/publ/DASFAA2010.pdf>.
- [45] E. Zeitler and T. Risch. Massive scale-out of expensive continuous queries. *Very Large Databases (VLDB)*, 2011. URL <http://www.vldb.org/2011/>.
- [46] K. Zheng, G. Trajcevski, X. Zhou, and P. Scheuermann. Probabilistic range queries for uncertain trajectories on road networks. *EDBT*, March 2011.

#### COLOPHON

The topic for this thesis culminated from earlier discussions between Asst. Prof. Gyozo Gidofalvi, Christian Borgelt and myself, in regards to a paper we submitted to EDBT 2011, on moving object trajectory mining. The topic for my thesis was further refined by discussions that ensued between Prof. Tore Risch, Erik Zeitler, Gyozo Gidofalvi and myself.

The prediction algorithm was discussed and designed by Gyozo, Christian and myself. The model was then implemented by me in C and used as a foreign function in SCSQ to extend it.

In the stream mining procedures, I implemented the route management ADTs while Christian contributed the core closed mining code by adapting a previously self-written and very stable, closed item mining C program to mine sequences and the routes generated by the ADT. Integration of the mining procedure into SCSQ by modifying it to mine a stream of incoming tuples and using SCSQ windowing features, was implemented by myself.

The SCSQ routines that were used in mining and prediction tasks were designed and written by me, with a lot of guidance and support from Prof. Tore Risch. Gyozo's guidance was instrumental in completing the *Preliminaries and Definitions* chapter.

It would not have been possible to complete this thesis without the aforementioned contributions, for which I am truly thankful.