Mining Long, Sharable Patterns in Trajectories of Moving Objects

Győző Gidófalvi[1] and Torben Bach Pedersen[2]

[1] Geomatic ApS — Center for Geoinformatics,
gyg@geomatic.dk
[2] Aalborg University — Department of Computer Science
tbp@cs.aau.dk

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

1 Introduction

In recent years Global Positioning Systems (GPS) have become increasingly available and accurate in mobile devices. As a result large amounts of spatio–temporal data is being generated by users of such mobile devices, referred to as *moving objects* in the

following. Trajectories of moving objects, or trajectories for short, contain regularities or patterns. For example, a person tends to drive almost every weekday to work approximately at the same time using the same route. The benefits of finding such regularities or patterns is many–fold. First, such patterns can help the efficient management of trajectories. Second, they can be used to facilitate various Location–Based Services (LBS). One LBS example is an intelligent rideshare application, which finds sharable routes for a set of commuters and suggests rideshare possibilities to them, is considered. Such a rideshare application can be one possible solution to the ever increasing congestion problems of urban transportation networks.

Patterns in trajectories for an intelligent rideshare application are only interesting if those patterns are sharable by multiple commuters, are reoccurring frequently, and are worthwhile pursuing, i.e., are long enough for the savings to compensate for the coordination efforts. The discovery of Long, Sharable Patterns (LSP) in trajectories is difficult for several reasons. Patterns do not usually exist along the whole trajectory. As a example, consider two commuters A and B living in the same area of town, leaving for work approximately the same time, and working in the same part of town. Given the underlying road network and traffic conditions, for a given support threshold the middle part of the trips of the two commuters may be frequent, the initial and final parts may not. In recent work [5] a general problem transformation method, called *pivoting*, was proposed for the analysis of spatio–temporal data. Pivoting is the process of grouping a set of records based on a set of attributes and assigning the values of likely another set of attributes to groups or baskets. Pivoting applied to spatio–temporal data allows the construction of spatio–temporal baskets, which can be mined with traditional association rule mining algorithms. When the items in the baskets are spatio–temporal identifiers and are derived from trajectories, the discovered rules represent frequently travelled routes. While there exist several efficient association rule mining methods [8], the straight–forward application of these algorithms to spatio–temporal baskets representing trajectories is infeasible for two reasons. First, all sub–patterns of frequent patterns are also frequent, but not interesting, as longer patterns are preferred. Second, the support criterion used in association rule mining algorithms is inadequate for a ride-

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

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

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

The remainder of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the transformation, the use of the framework in frequent itemset mining, and formally defines the task of mining LSPs in trajectories. Section 4 discusses a naïve method for mining LSPs and points out its shortcomings. Section 5 describes the proposed algorithm and a SQL–based implementation for mining LSPs. Section 6 presents alternative modelling of trajectories and derives variants of the proposed method based on these modelling options. Section 7 presents detailed experimental results. Finally Section 8 concludes and points to future research.

2 Related work

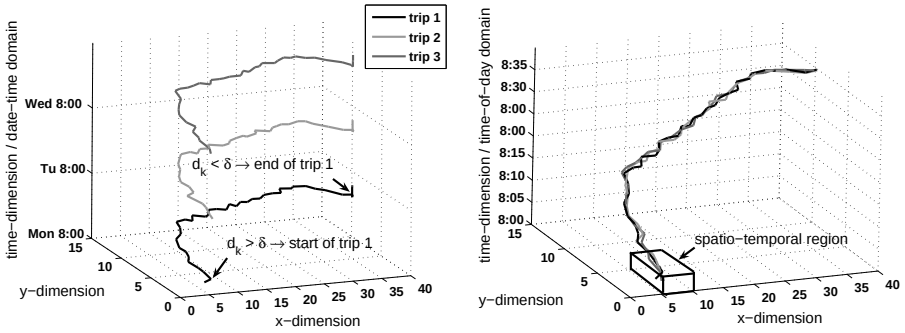
Frequent pattern mining is a core field in data mining research. Since the first solution to the problem of frequent itemset mining [1, 2], various specialized in–memory data structures have been proposed to improve the mining efficiency, see [8] for an overview. It has been recognized that the set of all frequent itemsets is too large for analytical purposes and the information they contain is redundant. To remedy this, two modification to the task have been proposed: mining of Closed Frequent Itemsets (CFI) and mining of maximal frequent itemsets. A frequent itemset X is *closed* if no itemset Y exists with the same support as X such that $X \subset Y$. A frequent itemset X is *maximal* if no frequent itemset Y exists such that $X \subset Y$. Prominent methods that efficiently exploit these modifications to the problem are MAFIA [4], GenMax [10], CLOSET [14], CLOSET(+) [19], and CHARM [22]. Later in the paper, a relationship between the problems of mining LSPs in trajectories and mining CFIs are described. While CFI mining methods can be modified to find the desired solution that meets the *sharable* criterion, they employ complex data structures and their implementation is quite involved; hence their augmentation is difficult. In particular, a projection-based CFI mining algorithm that employs an in–memory FP-tree to represent itemsets, would need to be modified at every node to maintain a set of distinct objects at that have transactions associated with them that support the itemset that is represented by the node. In comparison, the herein presented method –building on work presented in [16]–exploits the power of commercial RDBMSs, yielding a simple, but effective solution.

Since trajectories are temporally ordered sequences of locations, sequential pattern mining [3] naturally comes to mind. However, a straight forward interpretation of trips as transactions and application of a state–of–the–art closed frequent sequential pattern mining algorithm [21] does not yield the desired solution, since in this case sequences of frequent sub-trajectories would be found. Furthermore, since the trajectories can contain hundreds of items, closedness checking of frequent itemsets even for prominent methods would be computationally expensive. Interpreting single elements of trajectories as transactions and applying closed sequential pattern mining could find frequent sub–trajectories. However a number of problems arise. First, to meet the sharable cri-

terion, the in–memory data structures would need similar, non-trivial augmentation as described above. Second, since patterns in trajectories could be extremely long, even state–of–the–art sequential mining methods [17, 21] would have a difficulties handling patterns of such lengths. Third, patterns in trajectories repeat themselves, which cannot be handled by traditional sequential pattern mining algorithms. The extraction of spatio–temporal periodic patterns from trajectories is studied in [13], where a bottom–up, level–wise, and a faster top–down mining algorithm is presented. Although the technique is effective, the patterns found are within the trajectory of a single moving object. In comparison, the herein presented method effectively discovers long, sharable, periodic patterns.

Moving objects databases are particular cases of spatio–temporal databases that represent and manage changes related to the movement of objects. A necessary component to such databases are specialized spatio–temporal indices such as the Spatio–Temporal R–tree (STR–tree) and Trajectory–Bundle tree (TB–tree) [11]. An STR–tree organizes line segments of a trajectory according to both their spatial properties and the trajectories they belong to, while a TB–tree only preserves trajectories. If trajectories are projected to the time–of–day domain, STR–tree index values on the projected trajectories could be used as an alternative representation of trajectories. While this approach would reduce the size of the problem of mining LSPs in trajectories, it would not solve it. In comparison, the herein presented method solves the problem of mining LSPs in trajectories, which is orthogonal, but not unrelated to indexing of trajectories.

In [18] a way to effectively retrieve trajectories in the presence of noise is presented. Similarity functions, based on the longest sharable subsequence, are defined, facilitating an intuitive notion of similarity between trajectories. While such an efficient similarity search between the trajectories will discover similar trajectories, the usefulness of this similarity in terms of length and support would not be explicit. In comparison, there herein proposed method returns only patterns that meet the user–specified support and length constraints. Furthermore, the trajectory patterns returned by our method are explicit, as opposed to the only implicit patterns contained in similar trajectories.



(a) Identification of trips in raw trajectories.

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

Fig. 1. From trajectories to transactions

3 Long, sharable patterns in trajectories

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

3.1 From trajectories to transactions

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

Identification of trips

A trip is a temporally consecutive set or sequence of measurements such that for any measurement m_i in the sequence, the sum of spatial displacement during the k measurements immediately following m_i, denoted d_k, is larger than some user–defined displacement, δ. Trips can be identified in a straight–forward manner by linearly scanning through a trajectory, and calculating d_k using a look–ahead window of k mea-

surements. That is, scanning through the total trajectory from the beginning, the first measurement for which $d_k > \delta$, signals the beginning of the first trip. Consecutive measurements are part of this trip until a measurement is reached for which $d_k \leq \delta$, which signals the end of the first trajectory. Trips following the first trip are detected in the same fashion from the remaining part of the total trajectory. Figure 1(a) shows three example trips that are derived from the total trajectory of one moving object.

Projection of the temporal dimension

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

Spatio–temporal generalization and substitution

Trajectories are noisy. One source of this noise is due to imprecise GPS measurements. From the point of view of patterns in such trajectories, slight deviation of trajectories from the patterns can be viewed as noise. Examples of such deviations could be due to a few minute delay, or to the usage of different lanes on the route. Hence, while a person might be driving from home to work at approximately the same time of day using approximately the same route, the chance of two identical trajectories is highly unlikely. Consequently, patterns in raw trajectories are few and certainly not long. Thus, patterns have to be mined in trajectories that are represented in a generalized way, yielding general patterns in trajectories. There are at least two different approaches to achieve this generalization of trajectories: region–based spatio–temporal generalization and road network based spatio–temporal generalization.

In the region–based spatio–temporal generalization approach individual (x, y, t) measurements of a trajectory are discretized and mapped to the spatio–temporal regions they fall into. Thus, a generalized trajectory is constructed by substituting (x, y, t) measurements with the spatio–temporal regions they map to. If within a trajectory multiple (x, y, t) measurements map to the same spatio–temporal region, they are substituted with a single instance of the corresponding spatio–temporal region. The box in Figure 1(b) represents such a spatio–temporal region. Since region–based spatio–temporal substitution of a single database record can be achieved using simple arithmetics from the spatial and temporal coordinates, the processing time of this transformation step is optimal and linear in the number of database records.

In the road network based spatio–temporal generalization approach, objects are assumed to be moving on a road network and coordinates of individual (x, y, t) measurements of a trajectory are matched to road segments of the underlying road network. The process of matching trajectories to road segments is called map matching and has been studied extensively in the recent past. Figure 2 shows the outcome of map matching, where noisy GPS readings are “snapped” to the most likely road segments the object was actually moving on. In general, two map matching approaches exist: on–line

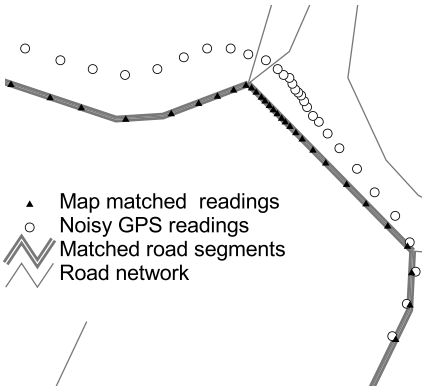


Fig. 2. Process / outcome of map matching

and off–line map matching. In on–line map matching, the noisy GPS readings are positioned onto the road network taking into account the past readings and the topology of the road network. In off–line map matching, the positioning of GPS readings onto the road network is performed with some delay, hence methods can take into consideration "future" measurements, which generally increases the matching accuracy and reduces the necessary computation. In [15] a summary of different on–line and off–line map matching algorithms is provided and disadvantages of each approach is described. Once the coordinates of trajectories are map matched, the individual (x, y, t) measurements of a trajectory are discretized and mapped to the spatio–temporal identifiers composed of a combination of road segment identifiers and temporal intervals. If within a trajectory multiple (x, y, t) measurements map to the same spatio–temporal identifier, they are substituted with a single instance of the corresponding spatio–temporal identifier. The map matching task can be performed in a distributed fashion by on–board navigation units of the moving objects. Based on the map matching results the road network based spatio–temporal substitution of a single database record can be achieved in constant time using simple arithmetics from the temporal values, hence the processing time of this transformation step is optimal and linear in the number of database records.

3.2 Example trajectory database

Figure 3 visualizes a sample trajectory database. It shows the trajectories of trips of 5 moving objects, which were derived using the three transformation steps described in Section 3.1. For clarity, the temporal dimension is projected down to the 2D–plane. Spatio–temporal regions are defined by the square cells and a five minute interval centered around time instances written inside the square. Each connected line represents specific trips of a particular object. The number of times that trip was performed by the

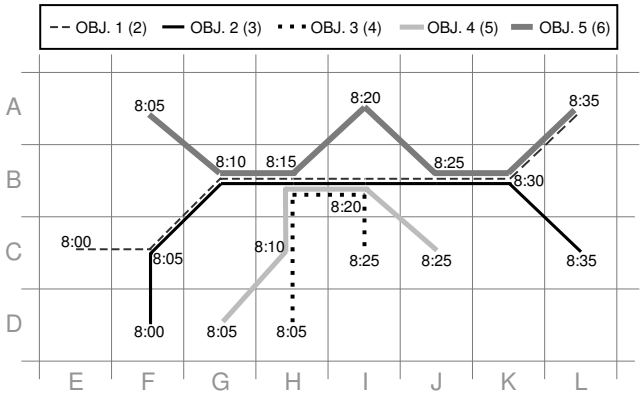


Fig. 3. Illustration of the sample trajectory DB

object is represented in the width of the line, and is also written in parenthesis next to the object name in the legend. For example, the trip trajectory associated with object 3 was performed 4 times by the object. The object was in spatial regions HD, HC, HB, IB, and IC during time intervals $8{:}05 \pm 2.5$ minutes, $8{:}10 \pm 2.5$ minutes, $8{:}15 \pm 2.5$ minutes, $8{:}20 \pm 2.5$ minutes, and $8{:}25 \pm 2.5$ minutes, respectively. In the following a spatio-temporal region will be referred to by its concatenated values of the cell identifiers along the x– and y–axis, and the corresponding time instance denoting the center of the time interval of the spatio–temporal region. Hence, trips associated with object 3 will be denoted by the a sequence {HD8:05, HC8:10, HB8:15, IB8:20, IC8:25}. Furthermore, the trajectory database T is assumed to be in a relational format with schema $\langle oid, tid, item \rangle$, where $item$ is a single item, that is part of the transaction tid associated with object oid. Hence, each of the four trips of object 3 is represented by 5 unique rows in T.

3.3 Problem statement

After performing the three above transformation steps, the data set can be represented in a database T containing tuples $\langle oid, tid, s \rangle$, where oid is an object identifier, tid is a trip identifier, and s is a sequence of spatio–temporal region identifiers. Since spatio–temporal region identifiers contain a temporal component, the sequence s can, without loss of information, be represented as a **set** of spatio–temporal region identifiers. Conforming to the naming convention used in the frequent itemset mining framework, a

spatio–temporal region identifier will be equivalently referred to as an *item*, and a sequence of spatio–temporal region identifiers will be equivalently referred to as a *transaction*. Let X be a set of items, called an *itemset*. A transaction t *satisfies* an itemset X iff $X \subseteq t$. Let ST_X denote the set of transactions that satisfy X. The following definitions are emphasized to point out the differences between the frequent itemset mining framework and the one established here.

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

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

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

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

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

4 Naïve approach to LSP mining

The here presented naïve approach uses the convenience and efficiency of an RDBMS. For ease of exposure, consider the problem of finding long sub–trajectories in trajectories. Meeting the unique support requirement of the original task does not substantially change the complexity of method to be described, but eases the description and analysis of it. Finding pairs of trajectories that have long sub–trajectories can be efficiently

solved using 2–way self–joins. Generally, K–way self–joins can be used to find groups of K trajectories that share parts of their trajectories. Consequently, to discover all long sub–trajectories, self–joins could be used in an iterative way, first discovering pairs, then triples, and so on, finally leading to groups of K trajectories that have long, sharable sub–trajectories. A solution based on self–joins has several drawbacks. As the number of trajectories is increasing, the maximum size of groups of trajectories that have a long sub–trajectory is expected to increase as well. Naturally, as this maximum group size is increasing, the number of self–joins that need to be performed is increasing as well. Although the sizes of the intermediate result sets of the consecutive joins that compose the K–way self–join are non–increasing with every join operation, and hence the required time to compute these joins is also non–increasing, the described K–way self–join method is inefficient. In fact its worst case running time is exponential in K, which is illustrated in the following. Consider a set of K trajectories that have a long sharable sub-trajectory in them. The iterative K–way self–join method in the first iteration, discovers all pairs of these K trajectories. Then in the next step, it discovers all groups of 3 of these trajectories, alternatively leading to the discovery of 2^K subsets of these K trajectories. This is clearly inefficient from a computational point of view not to mention the complexity it introduces in the discovered results. Since the ultimate goal of an intelligent rideshare application is the optimal coordination of possible rideshare opportunities of a set of commuters, the exponentially large number of discovered patterns is clearly a disadvantage from the user's point of view.

5 Projection–based LSP mining

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

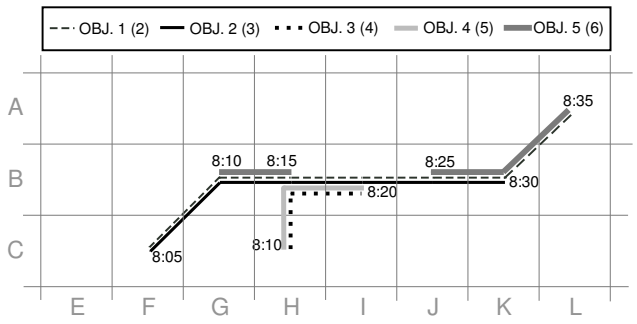


Fig. 4. The sample DB after STEP 1

STEP 1: Filtering infrequent items

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

Lemma 1 *An item i with $i.supp(n) < MinSupp$ cannot appear in a LSP p.*

Proof. The proof trivially follows from the minimum requirement of the n–support of a pattern p. If i appears in a pattern p, then the set of transactions satisfying p must be a subset of the transactions satisfying i. Consequently, $p.supp(n) \leq i.supp(n)$. For p to be a pattern $p.supp(n) \geq MinSupp$. This is a clear contradiction, hence i cannot appear in a pattern. □

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

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

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

The effects of the first step are illustrated in Figure 4. Spatio–temporal regions, which are part of trajectories that belong to less than 2 distinct objects, are removed from trajectories. From

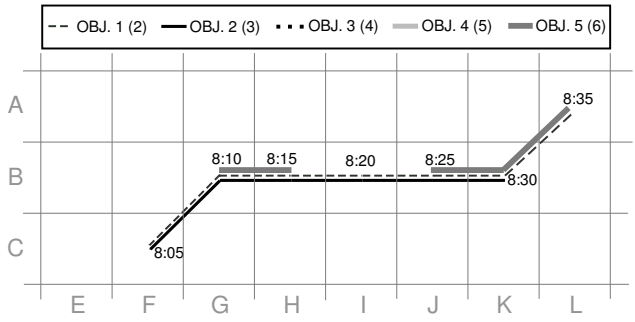


Fig. 5. The sample DB after STEP 2

the point of view of an intelligent rideshare application these spatio–temporal regions are uninteresting, since these parts of the trajectories cannot be shared by any objects, i.e., are not sharable.

STEP 2: Filtering of short transactions

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

Lemma 2 *A transaction t with $|t| < MinLength$ cannot satisfy a LSP p.*

Proof. The proof trivially follows from the definition of a LSP and the definition of a transaction satisfying a pattern. p is a LSP $\iff$ $p.supp(n) \geq MinSupp$ and $|p| \geq MinLength$. For t to satisfy p, by definition all the items in p has to be present in t. Since $|t| < MinLength$ and $|p| \geq MinLength$, there must exist at least one item in p that is not in t. Hence, t cannot satisfy p. □

The second step can be formulated in one SQL statement. The sub–select is used to find trip identifiers that have at least $MinLength$ number of items. The outer part of the statement selects all records belonging to these trip identifiers and inserts them into *TF*.

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

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

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

STEP 3: Item–conditional DB projection

For the following discussion, adopted from [14], let an item–conditional database of transactions, equivalently referred to as an item–projected database, be defined as:

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

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

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

Given n frequent items in T, the problem of finding CFIs can be divided into n subproblems of finding the CFIs in each of the n item–projected databases [14]. Using the divide–and–conquer paradigm, each of these n subproblems can be solved by recursively mining the item-projected databases as necessary.

STEP 4: Discovery of the single most frequent closed itemset

Since i is in every transaction of the item–projected database $T_{|i}$, and hence has maximum n–support, the items in $T_{|i}$ can be grouped in two: items that have the same n–support as i, and items that have n–support less than that of i. The set of items that have the same n–support in

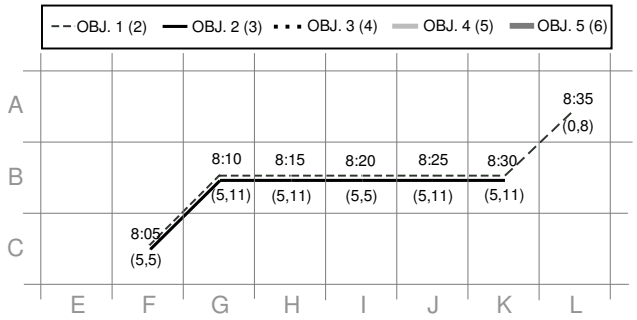


Fig. 6. Item–conditional sample DB $TF_{|\text{FC8:05}}$ and pattern discovery

the $T_{|i}$ as i is the Single Most Frequent Closed Itemset (SMFCI) in $T_{|i}$. The fourth step of the method discovers this SMFCI.

Lemma 3 *Let i be an item and $TF_{|i}$ its corresponding item–projected database. Let A be the set of items that have the same n–support in $TF_{|i}$ as i. Then A is the SMFCI in $TF_{|i}$.*

Proof. Complementary to A, let B be a set of items that have n–support less than i in $TF_{|i}$. For A to be closed there should not exist an itemset X such that $A \subset X$ and $A.supp(n) = X.supp(n)$. This implies that there should not exist an extra item i_e that is present in all transactions in which i is present. These transactions are exactly the set of transactions that make up $TF_{|i}$. The only remaining items that are not in A and are present in $TF_{|i}$ are items in B. Since items in B have n–support less than the items in A they could not be added to A to form an itemset X such that $A.supp(n) = X.supp(n)$. Hence A is a closed itemset. That A is the SMFCI trivially follows from the fact that the number of transaction in $TF_{|i}$ is $A.supp(n)$. □

The fourth step can be formulated in two SQL statements. The first statement derives the n–support of n–frequent of items in $TF_{|i}$,while the second statement selects those items from these n–frequent items that have maximum n–support.

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

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

Figure 6 shows the effects of projecting *TF* based on the item FC8:05. The numbers in parentheses show the n–support of the items in $TF_{|\text{FC8:05}}$ and *TF* respectively. The SMFCI that is immediately discovered from $TF_{|\text{FC8:05}}$ is {FC8:05, GB8:10, HB8:15, IB8:20, JB8:25, KB8:30}.

LA8:35 is the only item that is in $TF_{|\text{FC8:05}}$, but is not in the discovered SMFCI. Since further projecting $TF_{|\text{FC8:05}}$ on LA8:35 yields a database of transactions where no item meets the minimum n–support criterion, the discovered SMFCI is the only CFI present in $TF_{|\text{FC8:05}}$. Since the discovered SMFCI meets both the minimum length and an minimum n–support criteria it is a pattern.

Lemma 4 *Given an item–projected database $TF_{|i}$ and a partitioning of items in it into a set of most frequent items A, and a complementary set of items B, the recursive application of item–projection based on items in B followed by the discovery of the most frequent closed itemsets in the respective projected databases finds all CFIs in the item–projected database $TF_{|i}$.*

Proof. Given any CFI X with $X.supp(n) < A.supp(n)$ in $TF_{|i}$, we know that X contains at least 1 item $b \in B$. If not, then X contains only items in A, hence $X.supp(n) \geq A.supp(n)$, which is a clear contradiction. Then by Lemma 3, X will be found as the SMFCI in $TF_{|i|b}$, since $TF_{|i|b}$ contains all, and only those transaction from $TF_{|i}$ that satisfy b. □

STEP 5: Deletion of unnecessary items

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

Lemma 5 *After the construction of $TF_{|i}$, an item j can and must be deleted from* TF *if it has the same n–support in $TF_{|i}$ as in* TF.

Proof. If j has the same n–support in TF as in $TF_{|i}$, then the set of transactions that satisfy j in TF is exactly the same set of transactions that satisfy j in $TF_{|i}$. Since Lemma 4 guarantees that all closed frequent itmesets will be found in $TF_{|i}$, including those that j participates in, it is needless and incorrect to construct and mine $TF_{|j}$ at a later point to find same CFIs that j participates in again. Hence, j can and must be deleted from TF. □

The fifth step can be formulated in one SQL statement. The statement deletes all items in *TF* that have the same n–support in *TF* as in $TF_{|i}$.

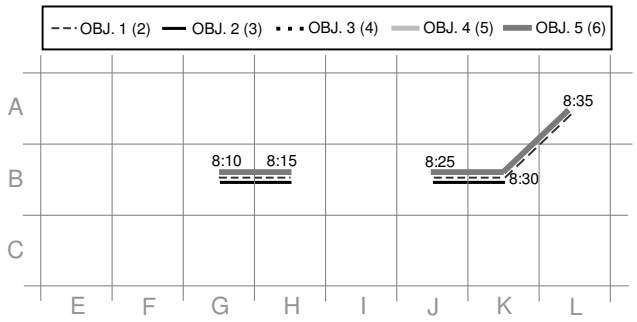


Fig. 7. The sample DB after STEP 5

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

Figure 7 shows the effects of deleting the unnecessary items after the mining of $TF_{|\text{FC8:05}}$. Since items FC:8:05 and IB8:20 have the same n–support in $TF_{|\text{FC8:05}}$ as in *TF*, shown in Figure 6, they are deleted from *TF*. Items remaining in *TF* are shown in Figure 7.

Item-projection ordered by increasing n–Support

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

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

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

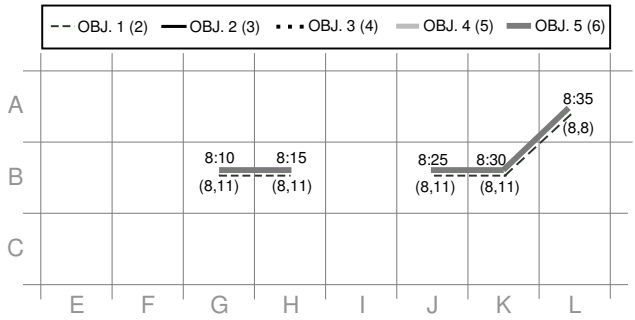


Fig. 8. Item–conditional DB $TF_{|\text{LA8:35}}$ and pattern discovery

Alternating pattern discovery and deletion

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

Figures 6 and 7 shows the effects of the first of these PDD phases. Figures 8 and 9 show the effect of the next pattern PDD phase. Since after the first PDD phase LA8:35 has the lowest n–support in *TF*, namely 8, it is chosen as the next item to base the database projection on. Figure 8 show $TF_{|\text{LA8:35}}$ with the corresponding n–support of the items in $TF_{|\text{FC8:05}}$ and *TF* respectively. Since all the items have the same n–support in $TF_{|\text{FC8:05}}$ as LA8:35, namely 8, the closed itemset $\{$GB8:10, HB8:15, JB8:25, KB8:30, LA8:35$\}$ is discovered. Since this closed itemset both meets the minimum length and n–support requirements it is recorded as a pattern. In the deletion part of this PDD phase, item LA8:35 is deleted from *TF* as the only item that have the same n–support in $TF_{|\text{LA8:35}}$ as in *TF*. The results of this deletion are shown on Figure 9.

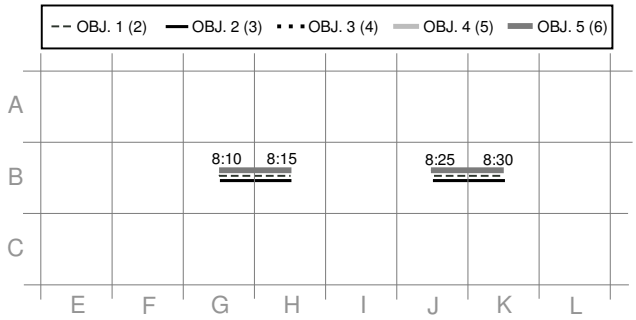


Fig. 9. The sample DB after the second PDD phase

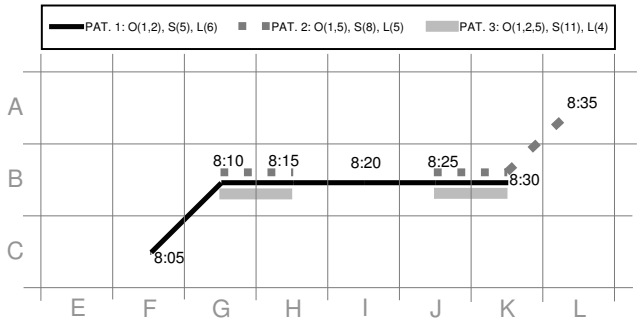


Fig. 10. Three patterns in the sample DB

The third and final PDD phase is implicitly shown in Figure 9. Since after the second PDD phase all the items in *TF* have the same n–support, the next projection is performed on any one of the items, i, and the resulting item–projected database, $TF_{|i}$, is identical to the current state of *TF*, depicted on Figure 9. Since all the items in $TF_{|i}$ have the same n–support as i, the closed itemset $\{$GB8:10, HB8:15, JB8:25, KB8:30$\}$ is discovered. Since this closed itemset meets both the minimum length and n–support requirements, it is recorded as a pattern. Finally, items having the same n–support in $TF_{|i}$ as in *TF*, which in this case means all the items in $TF_{|i}$, are deleted from *TF*. After this deletion part of the final PDD phase, *TF* becomes empty and the mining terminates. Figure 10 shows the three patterns that are discovered during the mining. Supporting oids, n–supports, and length for each discovered patterns are shown in the legend.

LSP mining algorithm

Using the observations and the associated steps, the complete algorithm for mining LSPs in trajectories is given in Figure 11. Since item–projected databases are constructed at every level of the recursion and are modified across levels when deleting unnecessary items, the level of recursion L is passed as an argument in the recursive procedure, and is used as a superscript to associate databases to the levels they were constructed in.

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

Lines 2 and 3 in the FindLSP procedure represent steps 1 and 2 of the method, and they construct the filtered database of transactions at the current level L. Line 4 represents step 4 of the method, and it finds the SMFCI P in TF^L_{long}. Line 5 represents step 5 of the method, and

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

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

Fig. 11. The LSP algorithm

it deletes all items from the filtered database of transactions of the previous level, TF^{L-1}_{long}, that have the same n-support in TF^{L-1}_{long} as in TF^{L}_{freq}, the current level. Lines 6 and 7 check if the single most frequent closed itemset P meets the minimum requirements and store it accordingly. Lines 8 and 9 processes frequent items in TF^{L}_{long}, which are not in P, in ascending order of n–support. Line 10 represent step 3 of the method, and for each such frequent item i it constructs the item–conditional database of transactions $TF^{L}_{|i}$ at the current level L. Finally, line 11 recursively calls procedure FindLSP to find LSPs in $TF^{L}_{|i}$ at the next level.

The structure and functionality of procedures MineLSP and FindLSP have a significant overlap. While the two functions can be merged into one, the separation of the two is used to emphasize the facts that (1) DeleteUnnecessaryItems presumes the existence of databases constructed

at the previous level, and (2) FindSMFCI correctly operates only on an item–projected database, and hence it can only be applied at level 1 and above.

Several implementation details are worth mentioning. First, DeleteUnnecessaryItems deletes items from TF^{L-1}_{long} based on the n–support of items in TF^{L}_{freq}, not TF^{L}_{long}. This is important, as it was noted that MinLengthFilter decreases the n–support of items in TF^{L}_{freq}, thereby possibly making an unnecessary item appear to be necessary. Second, arguments to functions and operands in statements are logical, i.e., the functions and statements can be more efficiently implemented using previously derived tables. For example, both FindSMFCI and DeleteUnnecessaryItems are implemented using previously derived n–support count tables not the actual trajectory tables. Third, simple shortcuts can significantly improve the efficiency of the method. For example, during the derivation of TF^{L}_{freq}, if the number of unique frequent items in TF^{L}_{freq} is less than $MinLength$, no further processing is required at that level, since none of the CFIs that can be derived from TF^{L}_{freq} are long. To preserve clarity, these simple shortcuts are omitted from Figure 11.

6 Alternative modelling of trajectories and mining of LSPs

The region–based and the road network based spatio–temporal generalization approaches, presented in Section 3, model trajectories at a local (micro) level. Consequently, the method presented in Section 5 analyzes the trajectories at the local level and derives local (micro) patterns. Alternatively, trajectories can also be modelled at the global (macro) level, whereby trips in trajectories are represented as origin–destination pairs. Global (macro) modelling and analysis of trajectories is a domain of considerable interest in transportation and urban analysis [9]. For example, recently a Cab–Sharing Service was proposed as an effective, door–to–door, on–demand transportation service [7]. One component of the proposed Cab–Sharing System is a Cab–Routing / Scheduling Engine. The task of this engine is to route idle cabs and assign cabs to requests or groups of requests, so called cab–shares, such that the demand for cabs is optimally served both in terms of the transportation cost of idle cabs and the service time of requests. To enable this optimization, as future work, the use of spatio–temporal patterns in cab requests for cab request demand prediction is proposed. Since cab requests are naturally represented as origin–destination pairs, the usefulness of macro analysis is apparent.

Hence, in the following two alternative options for modelling trajectories and mining of LSPs are described. Section 6.1 describes a simple method with an SQL implementation for mining

LSPs in trajectories modelled at the global (macro) level. Section 6.2, using some intuitive assumptions, combines the macro and micro modelling options and LSP mining methods to derive a hybrid version.

6.1 Macro modelling of trajectories and mining of LSPs

As briefly described above, when modelling trajectories at the global (macro) level, trips in trajectories are represented as origin–destination pairs. The preprocessing of raw trajectories can be achieved using the same three transformation steps as described in Section 3. This includes the possibility of using either the region–based or the road network based spatio–temporal generalization approaches as is required from the application at hand. In the so obtained transaction database, a trajectory belonging to a particular object has exactly two items.

Mining global (macro) LSPs in the so obtained trajectory database can be achieved using a single SQL statement as follows.

```
SELECT o_item, d_item, SUM(supp) AS nsupp FROM
    ( SELECT oid, o_item, d_item, COUNT(*) AS supp FROM T
    WHERE dist(o_item, d_item) >= MinDist
    GROUP BY oid, o_item, d_item ) a
GROUP BY o_item, d_item
HAVING COUNT(*) >= n AND SUM(supp) >= MinSupp
```

The statement, without loss of generality, assumes that the trajectory database T has the schema $\langle oid, tid, o\_item, d\_item \rangle$, where in addition to the previously used notation, o_item and d_item are generalized spatio–temporal regions, or identifiers of the origin and destination of the trajectory, respectively. Since all the trajectories in the database T have exactly two items, it does not make sense to talk about the length of a trajectory in terms of number of items in contains. Instead, a distance function $dist()$ between two locations or items can be defined, and patterns can be evaluated against a $MinDist$ criterion. The inner select statement calculates the supports for object–specific origin–destination item combinations that satisfy the $MinDist$ criterion. The outer select statement aggregates the results of the inner select statement, and identifies origin–destination item combinations that meet the n–support criterion, i.e., global (macro) LSPs, and calculates their corresponding n–supports.

6.2 Hybrid modelling of trajectories and mining of LSPs

The proposed global (macro) modelling approach of trajectories, the global (macro) LSP mining method and the SQL implementation given for it are likely to be very efficient due to the indexing and aggregation support provided by RDBMSs. However, the discovered global (macro) LSPs, will have very little relation to each other. For example, a set of individual global (macro) LSPs, considering the underlying road network, might give rise to local (micro) LSPs that do not exist in the macro model, but have a support that is equal to the sum of the n-supports of the individual patterns. As an illustrative example consider the road network represented by the solid black lines in Figure 12. Assume that for a particular setting of the parameters, the global (macro) LSP mining method finds two global (macro) LSPs, $\{$FA8:00, LA8:30$\}$ and $\{$FC8:00, LC8:30$\}$, with n–supports 10 for each. Assuming that the spatial regions FA, LA, FC, and LC cover the only four cities in the area, and hence trajectories only start and finish in these regions, the global (macro) LSP mining method will not discover the local (micro) LSP $\{$GB8:05, HB8:10, IB8:15, JB8:20, KB8:25$\}$ with n–support 20.

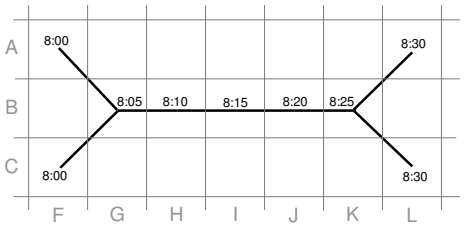


Fig. 12. Process / outcome of map matching

To overcome this deficiency of the global (macro) LSP mining method, the global (macro) and the local (micro) modelling approaches and LSP mining methods can be combined into a hybrid modelling and LSP mining method as follows. First, perform global (macro) LSP mining on the spatio–temporally generalized input trajectories. Then, using the global (macro) LSPs and the underlying road network, *approximate* trajectories for the global (macro) LSPs, i.e., find shortest paths between origin–destination pairs. Then, spatio–temporally generalize the approximated trajectories and mine local (micro) LSPs in them, taking into account the global (macro) n–supports of the approximated trajectories. Taking into account n–supports of the approximated trajectories can either be achieved by slightly modifying the local (micro) LSP mining method in Section 5, or simply the original version of it can be called with parameters $n = 1$ and $MinSupp = 1$. The latter is necessary and sufficient to ensure that (1) the approximated trajectories belonging to the global (macro) LSPs are found as local (micro) LSPs as well, and (2) the local (micro) LSPs found meet the original n–support criterion. Note that the spatio–temporal generalization of the

(1) **procedure** HybridMineLSP ($Traj_I$, $MinDist$, $MinSupp$, $MinLength$, n)
(2) $Traj_G \leftarrow$ STGeneralize ($Traj_I$)
(3) $LSP_{macro} \leftarrow$ MacroMineLSP ($Traj_G$, $MinDist$, $MinSupp$, n)
(4) $Traj_A \leftarrow$ ApproximateTraj (LSP_{macro})
(5) $Traj_{GA} \leftarrow$ STGeneralize ($Traj_A$)
(6) $LSP_{micro} \leftarrow$ MineLSP ($Traj_{GA}$, $MinLength$, $MinSupp = 1$, $n = 1$)

Fig. 13. The hybrid LSP mining method

input and approximated trajectories can either be regions–based or road network based. Figure 13 gives the pseudo code of the hybrid LSP mining method, as described above.

While the hybrid LSP mining method is likely to reduce the number of input trajectories to the local (micro) LSP mining method called internally, thereby achieving a significant speed–up in running time, it does not find *all* the local (micro) LSPs. As an example consider the trajectories in Figure 12. If $MinSupp = 20$, then the hybrid LSP mining method will not find any patterns in the global (macro) LSP mining phase, and consequently will not find any local (micro) LSPs, even though the local (micro) LSP $\{$GB8:05, HB8:10, IB8:15, JB8:20, KB8:25$\}$ has an n–support of 20. Similarly, it can be argued that the hybrid LSP mining method will not find any LSPs in the example trajectory database in Section 3.2, for the parameters used in the running example in Section 5. However, as the spatio–temporal generalization granularity used in the mining is decreased the chances not identifying global (macro) LSPs that give rise to local (micro) LSPs is decreased. In summary, the hybrid LSP mining method is likely an effective alternative that provides lossy and approximate results when compared to the local (micro) LSP mining method.

7 Experimental evaluation

The proposed LSP mining methods were implemented using MS-SQL Server 2000 running on Windows XP on a 3.6GHz Pentium 4 processor with 2GB main memory. Three groups of experiments were performed to test: (1) the parameter sensitivity of the local (micro) LSP mining method, (2) the scale–up properties of the local (micro) LSP mining method, and (3) the effectiveness of the global (macro) modelling and LSP mining method with respect to its parameters. The three groups of experiments were performed on the following three data sets respectively: (1) the publicly available INFATI data set [12], which comes from intelligent speed adaptation experiments conducted at Aalborg University, (2) the synthetic ST–ACTS trajectory data set, and (3) the ST–ACTS origin–destination data set, both of which were derived from ST–ACTS, a proba-

bilistic, parameterizable, realistic Spatio–Temporal ACTivity Simulator [6]. Sections 7.1, 7.2 and 7.3 describe these data sets in detail, while Sections 7.4, 7.5 and 7.6 present the results of the respective groups of experiments. Finally, Section 7.7 visualizes some of the mining results.

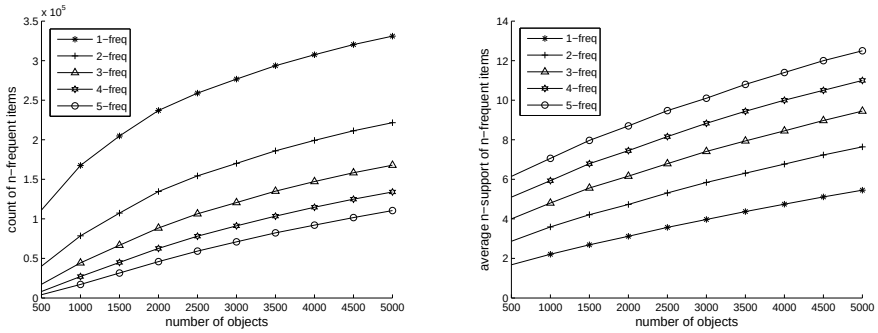
7.1 The INFATI data set

The INFATI data set records cars moving around in the road network of Aalborg, Denmark over a period of several months. 20 distinct test cars and families participated in the INFATI experiment; Team–1 consisting of 11 cars operated between December 2000 and January 2001 and Team–2 consisting of 9 cars operated between February and March 2001. Each car was equipped with a GPS receiver, from which GPS positions were sampled every second whenever the car was operated. Additional information about the experiment can be found in [12].

The method presented in Section 3.1 identifies trips from continuous GPS measurements, which is not the case in the INFATI data. Hence in this case, a trip was defined as sequence of GPS readings where the time difference between two consecutive readings is less than 5 minutes. Using the definition, the INFATI data contains 3,699 trips. After projecting the temporal dimension to the time–of–day domain and substituting the noisy GPS readings with 100 meter by 100 meter by 5 minutes spatio-temporal regions, the resulting trajectory database has the following characteristics. There are 200,929 unique items in the 3,699 transactions. The average number of items in a transaction is approximately 102. The average n–support of 1–, 2–, and 3–frequent items is 1.88, 4.2 and 6.4 respectively. Notice that the averages only include the n–supports of 1–, 2–, and 3–frequent items.

7.2 The ST–ACTS trajectory data set

The ST–ACTS trajectory data set is based on the output of ST–ACTS, a probabilistic, parameterizable, realistic Spatio–Temporal ACTivity Simulator [6]. Based on a number of real world data sources and a set of principles that try to model the *social* and some of the *physical* aspects of mobility, ST–ACTS simulates realistic spatio–temporal activity sequences of approximately 600,000 individuals in the city of Copenhagen, Denmark. Since the aim of ST–ACTS is to simulate realistic spatio–temporal activities of individuals that contain patterns, rather than to simulate detailed movements of individuals, the output of the ST–ACTS for each simulated individual is a sequence of timestamped locations and activities. Two consecutive locations in such a sequence can be seen as the origin and the destination of a trip's trajectory. To obtain a realistic approximation for the missing part of the trajectories, using the underlying road network, a segment–based shortest path calculation was performed between the origin–destination pairs of the trips. The

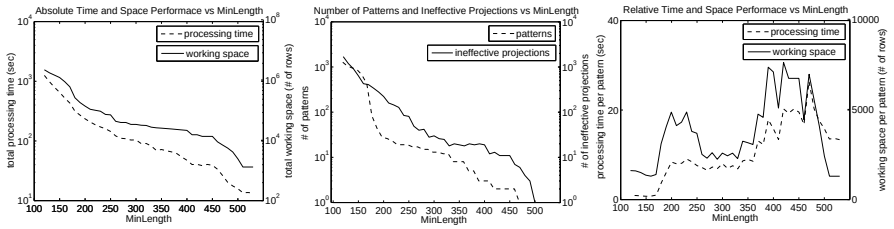


(a) Number of n–frequent items. (b) Average n–support of n–frequent items.

Fig. 14. Data characteristics of varying sized subsets of the ST–ACTS trajectory data set

so obtained trip trajectories are analogous in form and semantics to the trajectories that can be obtained using the road network based spatio–temporal generalization approach as explained in Section 3.1.

For the period of three working days, spatio–temporal activities of 5,000 individuals were simulated, resulting in a total of 64,144 trips. Using segment–based routing between origin–destination pairs, an average trip is 1,850 meters long with a standard deviation of 1,937 meters, and is made up of 28 road segments with a standard deviation of 27 road segments. An average road segment is 66 meters long with a standard deviation of 64 meters. After projecting the temporal dimension to the time–of–day domain and using the road segments as spatial–, and a 15–minute interval as temporal, generalization units, the resulting trajectory database has the following characteristics. There are 330,940 unique items in the 64,144 transactions. The average number of items in a transactions is approximately 28. To test the scale–up properties of the proposed method, varying sized subsets of the ST–ACTS trajectory data set were constructed. Figure 14 summarizes the characteristics of these subsets in terms of the number (Figure 14(a)) and n–support (Figure 14(b)) of n–frequent items. While not shown in Figure 14, the number of trajectories linearly scales with the number of objects in the data sets between 6,601 trajectories for 500 objects and 64,144 trajectories for 5,000 objects. Similar linear relationships exists between the number of number of objects and the average n–support of n–frequent items, Figure 14(b). The logarithmic like relationships between the number of objects and the number of n–frequent items is due to the fact that the increasing number of trajectories traverse, and make n–frequent, an increasing fraction of road segments of the total road network, see Figure 14(a). In other words, the number of n–frequent items naturally saturates as the density of the trajectories increases.



(a) Absolute time and space (b) Number of patterns (c) Relative time and space

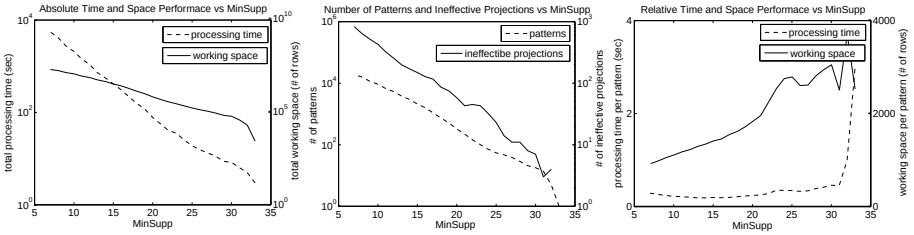
Fig. 15. Performance evaluation for various $MinLength$ settings

7.3 The ST–ACTS origin–destination data set

The ST–ACTS origin–destination data set, similarly to the ST–ACTS trajectory data set, is also based on the output of ST–ACTS. However, it includes the spatio–temporal activities of 50,000 individuals for a period of three working days, resulting in a total of 835,806 trips. The average Euclidean distance between the origins and destinations of the trips is 1,199 meters with a standard deviation of 1.555 meters. After projecting the temporal dimension to the time–of–day domain and substituting the origins and destinations of trips with 100 meter by 100 meter by 15–minute spatio-temporal regions, the resulting trajectory database has the following characteristics. There are 139,480 unique items in the 1,671,612 transactions. The number of items in every transactions is exactly 2, which correspond to an origin and a destination spatio–temporal region. The average n–support (and count) of 1–, 2–, 3–, and 4–frequent items is 1.12 (1,497,871), 2.14 (152,255), 3.24 (17,267), and 4.09 (3854) respectively. Notice that the averages only include the n–supports of 1–, 2–, 3–, and 4–frequent items.

7.4 Sensitivity experiments for $MinSupp$ and $MinLength$ parameters

The first group of experiments was performed to test the sensitivity of local (micro) LSP mining method with respect to the $MinSupp$ and $MinLength$ parameters, and was using the INFATI data set as input. Two sets of experiments were performed, each varying one of the two parameters of the algorithm, $MinSupp$ and $MinLength$. The performance of the algorithm was measured in terms of processing time and working space required, where the working space required by the algorithm was approximated by the sum of the rows in the projected tables that were constructed by the algorithm. Both measures were evaluated in an absolute and a relative, per pattern, sense. Figures 15(a), 15(b), and 15(c) show the results of the first set of experiments, where $MinSupp = 2$, $n = 2$ and $MinLength$ is varied between 120 and 530. Lower settings for $MinLength$ were also tested, but due to the very low $MinSupp$ value these measurement were terminated after exceeding the 2 hour processing time limit. Noting the logarithmic scale in Figure 15(a) it is evident that



(a) Absolute time and space (b) Number of patterns (c) Relative time and space

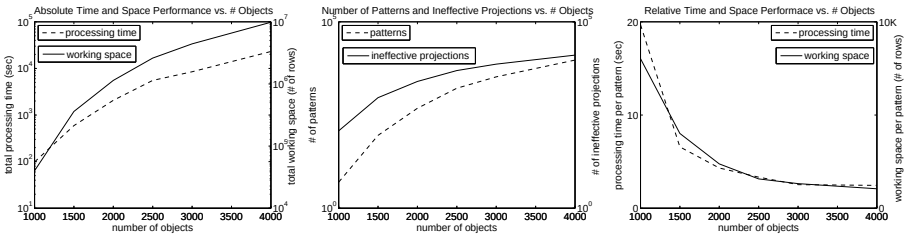
Fig. 16. Performance evaluation for various $MinSupp$ settings

both the running time and the working space required by the algorithm exponentially increase as the $MinLength$ parameter is decreased. Examining the same quantities in a relative, per pattern sense, Figure 15(c) reveals that the average running time and average working space required per pattern is approximately *linearly decreasing* as the $MinLength$ parameter is decreased. The presence of the two irregular bumps in Figure 15(c) can be explained in relation to the number of patterns found, and the number of ineffective projections that yield no patterns, shown in Figure 15(b). The sharp increases in relative processing time and working space are due to the fact that the algorithm is unable to take some of the shortcuts and it performs relatively more ineffective projections yielding no pattern discovery. The sharp decreases can be explained by the presence of an increasing number of patterns that share the total pattern discovery cost.

Similar observations can be made about the second set of experiments, shown in Figures 16(a), 16(b), and 16(c), where $MinLength = 50$, $n = 2$ and $MinSupp$ is varied between 7 and 33. For example, the sharp decrease in relative processing time in Figure 16(c) when going from $MinSupp = 33$ to $MinSupp = 32$ is simply due to the sudden appearance of patterns in the data for the given parameters. While there is only 1 pattern for $MinSupp = 33$, and an order of magnitude more number of patterns for $MinSupp = 32$, the projections performed and hence the absolute processing time to discover these patterns is approximately the same in both cases. Hence, the relative processing time for $MinSupp = 33$ is an order of magnitude larger than that for $MinSupp = 32$.

7.5 Scale–up experiments for various input data sizes

The second group of experiments was performed to test the scale–up properties of the local (micro) LSP mining method for varying size input data, and was performed using the ST-ACTS trajectory data set. For this group of experiments the algorithm's parameters were kept constant at $n = 4$, $MinSupp = 8$ and $MinLength = 25$. In other words, the patterns sought for were sub–



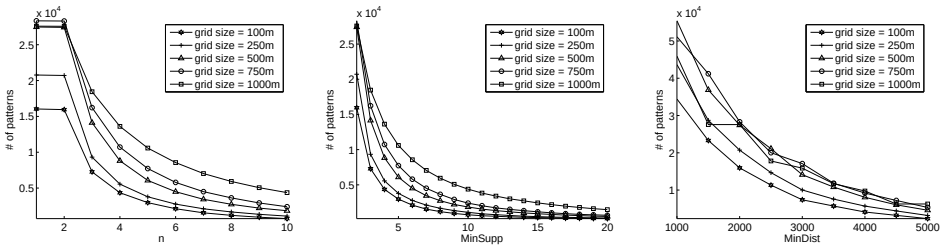
(a) Absolute time and space (b) Number of patterns (c) Relative time and space

Fig. 17. Performance evaluation for various sized data sets, i.e., number of objects

trajectories with a minimum length of 25 road segments that were supported by at least 4 objects on average at least two times per object. The evaluation measures used in the experiments were the same as in the sensitivity experiments described in Section 7.4. Figure 17 shows the results of this group of experiments. The results can be summarized as follows. As the number of objects increases linearly, i.e. the density of the trajectories increases linearly, the number of patterns increases sub–exponentially, see Figure 17(b). This naturally leads to a sub–exponential increase in absolute running time (Figure 17(a) left) and working space (Figure 17(a) right). However, the examination of the same quantities in a relative, per pattern sense, see Figure 17(c), reveals that the average running time required per pattern gradually decreases to a close to constant value of a few seconds as the density of the trajectories increases. This is due to the fact that as the density of the trajectories is increasing, the number of ineffective projections relative to the number of patterns, i.e., the gap between the two, is decreasing, see Figure 17(c). Similar observations can be made about the average working space required per pattern. In summary, the scale–up experiments show that both the running time of, and working space required by the local (macro) LSP mining method scale sub–exponentially with the input data size and linearly with the output data size.

7.6 Global (macro) modelling and LSP mining experiments

The third groups of experiments was performed to test the effectiveness of the global (macro) modelling and LSP mining method, as presented in Section 6.1. The experiments were performed for varying spatio–temporal generalization granularities for varying n, $MinSupp$ and $MinDist$ parameters, and were using the ST–ACTS origin–destination data set. Figure 18 shows the results of this group of experiments. The trends in the results are as expected. As the values for the parameters n, $MinSupp$ and $MinDist$ increases the number of global (macro) LSPs decreases, shown in Figures 18(a), 18(b), and 18(c), respectively. Similarly, all experiments show that as the



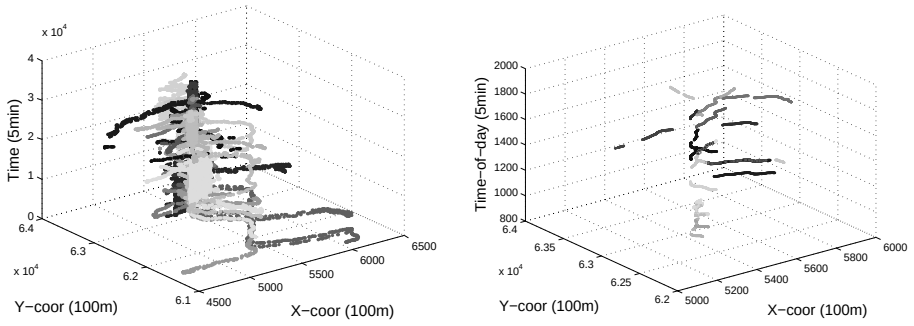
(a) Varying n. (b) Varying $MinSupp$. (c) Varying $MinDist$.

Fig. 18. Number of global (macro) LSPs for varying spatio–temporal granularities and parameter settings

spatio–temporal regions become coarser the number of patterns found increases. Perhaps more notable is the fact that for the rather large data set, with appropriate indexing the running time of the global (macro) LSP mining method is independent of the parameters and is under 2 seconds.

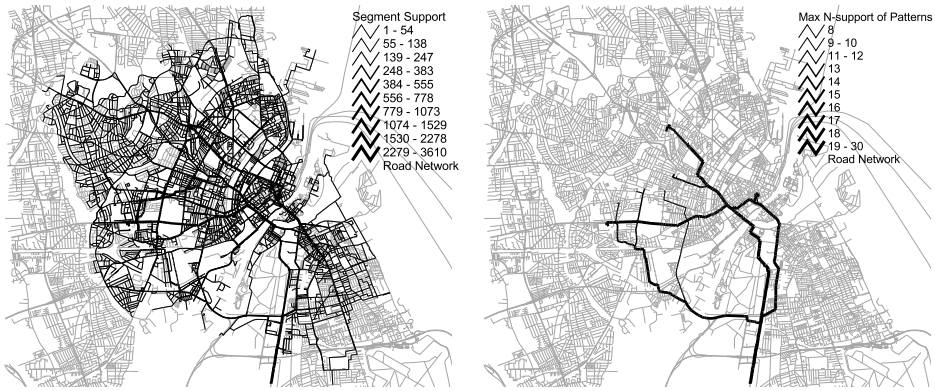
7.7 Visualization of patterns

To see the benefits of mining LSPs, the mining results of two mining tasks are visualized in the following. Figure 19 presents the mining results of finding local (micro) patterns in the region–based spatio–temporally generalized INFATI data using the LSP algorithm from Section 5. Figure 19(a) shows a 50–fold down–sampled version of the trajectories of the 20 moving objects in the INFATI data set. While some regularities are apparent in it to the human eye, to find LSPs in it seems like a daunting task. Figure 19(b) shows 28 LSPs in it that are at least 200 long, sharable for at least 2 distinct objects, and have a support of at least 2.



(a) Moving object trajectories. (b) 28 discovered LSPs.

Fig. 19. LSP discovery results in the INFATI data set



(a) Overall road segment support. (b) Max road segment n-support of 298 LSPs.

Fig. 20. LSP discovery results for the ST–ACTS origin–destination data set

Figure 20 presents the mining results of finding local (micro) patterns in the road network based spatio–temporally generalized ST-ACTS origin–destination data using the hybrid LSP mining algorithm from Section 6.2. The global (macro) mining of LSPs was performed for parameters $n = 4$, $MinSupp = 8$, and $MinDist = 5,000$ meters, and resulted in 109 global (macro) LSPs. After determining the value for the $MinLength$ parameter at 80, based on the minimum number of road segments in the approximated trajectories of the global (macro) LSPs, the local (micro) mining of LSPs resulted in 298 local (micro) LSPs. Figure 20(a) shows the overall supports (frequencies) of the road segments in the data. Figure 20(b) shows the maximum n–supports of the road segments induced by the collection of the 298 local (micro) LSPs. While, as described in Section 6.2, the hybrid LSP algorithm is not likely to find *all* local (micro) LSPs, it does find a relatively large number of additional local (micro) LSPs in a rather large data set under 144 seconds.

It is important to note that the LSPs contain additional information, which is only partially, or not presented in Figures 20(b) and 19(b), respectively. In particular, the n–supports, distances, and lengths are available for *individual* patterns, and the patterns naturally have a temporal aspect to them. With regards to the latter feature of the patterns, since the items in a pattern have a temporal component, an individual pattern refers to a particular time–of–day. Furthermore, since the spatio–temporally generalized items in a given pattern form a sequence in time, the patterns have a *direction*. While a simple temporal, frequency analysis of road segments can reveal *aggregated* information about the number of objects on the road segments at a given time–of–day, such analysis will not reveal movement patterns (including directions) of similarly moving objects.

This additional information of the LSPs is likely to be of immense value in transportation and urban planning, and is necessary for the application at hand, be that intelligent ride–sharing or cab–sharing.

7.8 Summary of experimental results

In conclusion, the experimental evaluations can be summarized as follows. First, the sensitivity experiments show that the LSP mining method, presented in Section 5, is effective and is robust to changes in the user–defined parameter settings, $MinLength$ and $MinSupp$, and is a useful analysis tool for finding LSPs in moving object trajectories. Second, the scale–up experiments show that while the absolute running time and space required to find LSPs scales exponentially with the input data size, this is mainly due to the fact that the number of LSPs that are present in the input data, i.e., the output data size, also scales exponentially with the input data size. The scale–up experiments also demonstrate that the relative, per pattern, performance of the LSP mining method gradually decreases to a constant value as the input data size is increased. This later property of the LSP method is due to the fact that as the input size is increased, i.e., the density of the trajectories is increased, the effects of the $MinLength$ and $MinSupp$ pruning criteria become more dominant and relatively less and less ineffective projections are performed. Third, the experiments relating to global (macro) modelling and LSP mining, show that this modelling option and LSP mining method is extremely effective on large data sets, and is rather insensitive to the user–defined parameter settings, n, $MinDist$ and $MinSupp$. Finally, brief experiments show that the hybrid modelling and LSP mining method, while missing some local (micro) LSPs due to the partial global (macro) modelling, is able to find local (micro) LSPs in large data sets effectively.

8 Conclusions and future work

The herein presented work, for the first time, considers the problem of mining LSPs in trajectories and transforms it to a framework similar to that used in frequent itemset mining. The transformation allows both region–based and road network based spatio–temporal generalizations of trajectories. Two methods and their simple SQL–implementations are presented for mining either local (micro) or global (macro) LSPs in such spatio–temporally generalized trajectories. In an attempt to speed up the local (micro) LSP mining method, the two methods are combined to a hybrid LSP mining method, which is able to rapidly find most of the local (micro) LSPs. The effectiveness of

the different LSP methods is demonstrated through extensive experiments on both a real world data set, and a number of large–scale, synthetic data sets.

Future work is planed along several directions. First, as discussed, the hybrid LSP method, while is able to achieve significant speed-up compared to the local (micro) LSP mining method, it does not find all the local (micro) LSPs in the trajectories. Hence future work will consider to quantify (1) the speed–up of the hybrid LSP method, and (2) the fraction of the local (micro) LSPs found by the hybrid LSP method. Second, the large number of patterns discovered are difficult to analyze. To reduce this number, future work will consider the mining of a compressed patterns in trajectories [20]. Finally, future work will consider the partitioning of trajectories using index structures designed for trajectories, like in [11], to allow the distributed / parallel mining of LSPs.

Acknowledgments

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

References

1. R. Agrawal, T. Imilienski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. of SIGMOD*, pp. 207–216, 1993.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of VLDB*, pp. 487–499, 1994.
3. R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proc. of ICDE*, pp. 3–14, 1995.
4. D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In *Proc. of ICDE*, pp. 443–452, 2001.
5. G. Gidófalvi and T. B. Pedersen. Spatio-Temporal Rule Mining: Issues and Techniques. In *Proc. of DaWaK*, pp. 275–284, 2005.
6. G. Gidófalvi and T. B. Pedersen. ST–ACTS: A Spatio-Temporal Activity Simulator. In *Proc. of ACM-GIS*, pp. 155–162, 2006.
7. G. Gidófalvi and T. B. Pedersen. Cab–Sharing: An Effective, Door–To–Door, On–Demand Transportation Service. In *Proc. of ITS*, 2007.
8. B. Goethals. Survey on frequent pattern mining: `http://citeseer.ist.psu.edu/goethals03survey.html`

9. Integrated land-use and transportation models, behavioural foundations, M.L Gosseling and S. Doherty, Elsevier, 2005.
10. K. Gouda and M. J. Zaki. GenMax: An Efficient Algorithm for Mining Maximal Frequent Itemsets. In *Data Mining and Knowledge Discovery*, 11(3), pp. 223–242, 2005.
11. C. S. Jensen, D. Pfoser, and Y. Theodoridis. Novel Approaches to the Indexing of Moving Object Trajectories. In *Proc. of VLDB*, pp. 395–406, 2000.
12. C. S. Jensen, H. Lahrmann, S. Pakalnis, and S. Runge. The INFATI data. Time Center TR–79, 2004: `www.cs.aau.dk/TimeCenter`
13. N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung. Mining, Indexing, and Querying Historical Spatiotemporal Data. In *Proc. of KDD*, pp. 236–245, 2004.
14. J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proc. of DMKD*, pp. 11-20, 2000.
15. M. Quddus, W. Ochieng, L. Zhao, and R. Noland. A general map matching algorithm for transport telematics applications. In *GPS Solutions Journal*, 7(3), pp. 157-167, 2003.
16. X. Shang, K.-U. Sattler, and I. Geist. Efficient Frequent Pattern Mining in Relational Databases. In *Proc. of LWA*, pp. 84–91, 2004.
17. I. Tsoukatos and D. Gunopulos. Efficient Mining of Spatiotemporal Patterns. In *Proc. of SSTD*, pp. 425–442, 2001.
18. M. Vlachos, D. Gunopoulos, and G. Kollios. Discovering Similar Multidimensional Trajectories. In *Proc. of ICDE*, pp. 673–685, 2002.
19. J. Wang, J. Han, and J. Pei. CLOSET+: searching for the best strategies for mining frequent closed itemsets. In *Proc. of KDD*, pp. 236–245, 2003.
20. D. Xin, J. Han, X. Yan, and H. Cheng. Mining Compressed Frequent-Pattern Sets. In *Proc. of VLDB*, pp. 709–720, 2005.
21. X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large datasets. In *Proc. of SDM*, pp. 166–177, 2003.
22. M. J. Zaki and C.J. Hsiao. CHARM: An Efficient Algorithm for Closed Itemset Mining. In *Proc. of SDM*, pp. 71–80, 2002.