Mining Long, Sharable Patterns in Trajectories of Moving Objects

Gyozo Gidófalvi / Torben Bach Pedersen

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

Abstract

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

1 Introduction

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

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

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

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

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

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

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

2 Related work

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

Since trajectories are temporally ordered sequences of locations, sequential pattern mining [3] naturally comes to mind. However, a straight forward interpretation of trips as transactions and application of a state–of–the–art closed frequent sequential pattern mining algorithm [15] does not yield the desired solution, since in this case sequences of frequent 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 criterion, 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 [12, 15] would have a difficulties handling patterns of such lengths. Third, patterns in trajectories repeat themselves, which cannot be handled by traditional sequential pattern mining algorithms. The extraction of spatio–temporal periodic patterns from trajectories is studied in [9], where a bottom–up, level–wise, and a faster top–down mining algorithm is presented. Although the technique is effective, the patterns found are within the trajectory of a single moving object. In comparison, the herein presented method effectively discovers long, sharable, periodic patterns.

Moving objects databases are particular cases of spatio–temporal databases that represent and manage changes related to the movement of objects. A necessary component to such databases are specialized spatio–temporal indices such as the Spatio–Temporal R–tree (STR–tree) and Trajectory–Bundle tree (TB–tree) [7]. An STR-tree organizes line segments of a trajectory according to both their spatial properties and the trajectories they belong to, while a TB–tree only preserves trajectories. If trajectories are projected to the time–of–day domain, STR–tree index values on the projected trajectories could be used as an alternative representation of trajectories. While this 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 [13] a way to effectively retrieve trajectories in the presence of noise is presented. Similarity functions, based on the longest sharable subsequence, are defined, facilitating an intuitive notion of similarity between trajectories. While such an efficient similarity search between the trajectories will discover similar trajectories, the usefulness of this similarity in terms of length and support would not be explicit. In comparison, there herein proposed method returns only patterns that meet the user–specified support and length constraints. Furthermore, the trajectory patterns returned by our method are explicit, as opposed to the only implicit patterns contained in similar trajectories.

3 Long, sharable patterns in trajectories

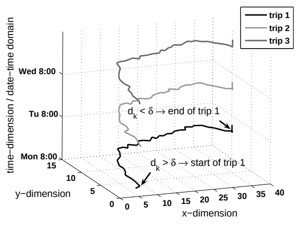
The following section describes a novel transformation of raw trajectories. This transformation allows (1) the formulation of the problem of mining LSPs in trajectories in a framework similar to that used in frequent itemset mining, (2) to establish a relationship between the two problems.

3.1 From trajectories to transactions

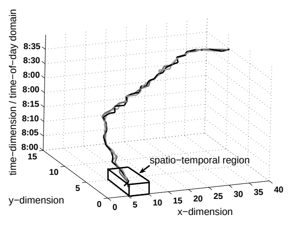
The proposed transformation of raw trajectories consists of three steps: identification of trips, projection of the temporal dimension, and spatio–temporal region substitution. It is assumed that locations of moving objects are sampled over a long history. That is, a raw trajectory is a long sequence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

Identification of trips

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



(a) Identification of trips in raw trajectories



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

Figure 1: From trajectories to transactions

Projection of the temporal dimension

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

Spatio–temporal region substitution

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

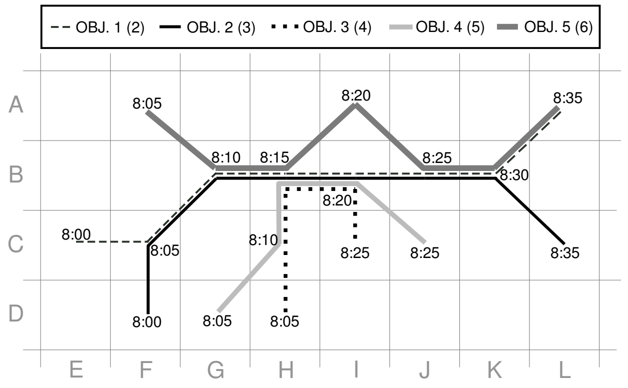


Figure 2: Illustration of the sample trajectory DB

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

3.2 Example trajectory database

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

3.3 Problem statement

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

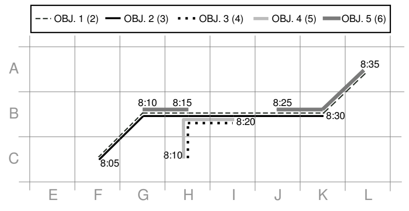
Definition 1 *The* **n–*support of an itemset*** X *in* T*, denoted as* $X.supp(n)$*, is defined as the number of transactions in* ST_X *if the number of distinct oids associated with the transactions in* ST_X *is greater than or equal to* n*, and* 0 *otherwise. The* **n–*support of an item*** i *in*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

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

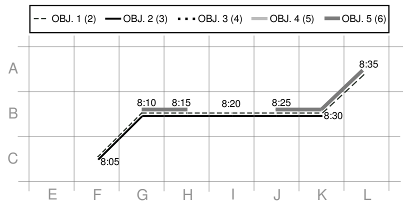
Definition 3 *An itemset* X *is* **n–*frequent*** *in* T *if* $X.supp(n) \geq MinSupp$*, and* X *is* ***long*** *if* $|X| \geq MinLength$*, where MinLength, MinSupp, and* n *are user–defined values.*

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

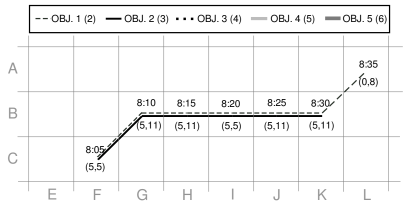
The task of mining LSPs in trajectories can be defined as finding all long, n–closed, n–frequent itemsets. Itemsets that meet these requirements are also referred to as LSPs, or just patterns.



(a) The sample DB after STEP 1



(b) The sample DB after STEP 2



(c) STEP 3: Item–conditional sample DB $TF_{|\mathrm{FC8:05}}$ and pattern discovery

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

4 Projection–based LSP mining

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

STEP 1: Filtering infrequent items

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

The first step can be formulated in two SQL statements:

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

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

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

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

STEP 2: Filtering of short transactions

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

The second step can be formulated in one SQL statement:

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

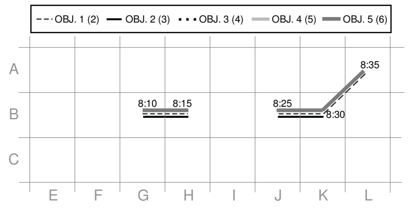
The sub–select is used to find trip identifiers that have at least *MinLength* number of items. The outer part of the statement selects all records belonging to these trip identifiers and inserts them into *TF*.

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

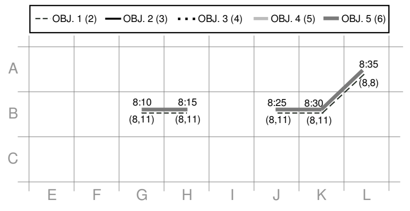
Before stating further observations and continuing with the development of the proposed method it is important to note the following. The set of discoverable LSPs from T is equivalent to the set of discoverable LSPs from *TF*. This is ensured by first two observations. Since further steps of the proposed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

STEP 3: Item–conditional DB projection

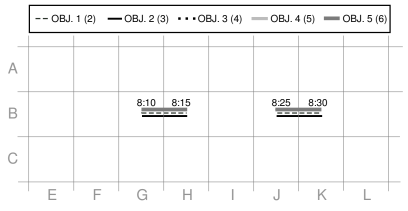
For the following discussion, adopted from [10], let an item–conditional database of transactions, equivalently referred to as an an item–projected database, be



(a) The sample DB after STEP 5



(b) Item–conditional DB $TF_{|\mathrm{LA8:35}}$ and PDD



(c) Sample DB after the second PDD

Figure 4: Illustration of Pattern Discovery and Deletion phases

defined as:

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

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

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

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

STEP 4: Discovery of the single most frequent closed itemset

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

The fourth step can be formulated in two SQL statements:

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

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

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

Figure 3(c) shows the effects of projecting *TF* based on the item FC8:05. The numbers in parentheses show the n–support of the items in $TF_{|\mathrm{FC8:05}}$ and *TF* respectively. The SMFCI that is immediately discovered from $TF_{|\mathrm{FC8:05}}$ is {FC8:05, GB8:10, HB8:15, IB8:20, JB8:25, KB8:30}. LA8:35 is the only item that is in $TF_{|\mathrm{FC8:05}}$, but is not in the discovered SMFCI. Since further projecting $TF_{|\mathrm{FC8:05}}$ 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 $TF_{|\mathrm{FC8:05}}$. Since the discovered SMFCI meets both the minimum length and an minimum n–support criteria it is a pattern.

STEP 5: Deletion of unnecessary items

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

The fifth step can be formulated in two SQL statements:

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

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

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

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

Item-projection ordered by increasing n–Support

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

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

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

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

Alternating pattern discovery and deletion

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

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

The third and final PDD phase is implicitly shown in Figure 4(c). Since after the second PDD phase all the items in TF have the same n–support, the next projection is performed on any one of the items, i, and the resulting item–projected database, $TF_{|i}$, is identical to the current state of TF, depicted on Figure 4(c). Since all the items in $TF_{|i}$ have the same n–support as i, the closed itemset {GB8:10, HB8:15,

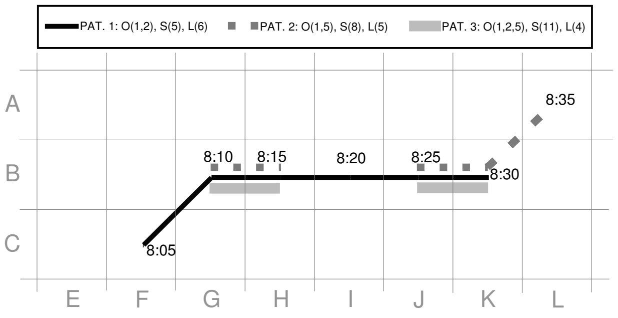


Figure 5: Three patterns in the sample DB

JB8:25, KB8:30} is discovered. Since this closed itemset meets both the minimum length and n–support requirements, it is recorded as a pattern. Finally, items having the same n–support in $TF_{|i}$ as in TF, which in this case means all the items in $TF_{|i}$, are deleted from TF. After this deletion part of the final PDD phase, TF becomes empty and the mining terminates. Figure 5 shows the three patterns that are discovered during the mining. Supporting *oid*s, n–supports, and length for each discovered patterns are shown in the legend.

LSP mining algorithm

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

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

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

```
(1)  procedure MineLSP ($T$, $MinSupp$, $MinLength$, $n$)
(2)    $TF^0_{freq} \leftarrow$ MinSupportFilter ($T$, $MinSupp$, $n$)
(3)    $TF^0_{long} \leftarrow$ MinLengthFilter ($TF^0_{freq}$, $MinLength$)
(4)    for each freq item $i$ in $TF^0_{long}$ ordered by asc $n$–supp
(5)      $TF^0_{|i} \leftarrow$ 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)    $TF^L_{freq} \leftarrow$ MinSupportFilter ($T$, $MinSupp$, $n$)
(3)    $TF^L_{long} \leftarrow$ MinLengthFilter ($TF^L_{freq}$, $MinLength$)
(4)    $(P, P.supp(n)) \leftarrow$ FindSMFCI ($TF^L_{long}$)
(5)    $TF^{L-1}_{long} \leftarrow$ DeleteUnnecessaryItems ($TF^{L-1}_{long}$, $TF^L_{freq}$)
(6)    if $P.supp(n) \geq MinSupp$ and $|P| \geq MinLength$
(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)       $TF^L_{|i} \leftarrow$ ConstructConditionalDB ($TF^L_{long}$, $i$)
(11)       FindLSP ($TF^L_{|i}$, $L+1$, $MinSupp$, $MinLength$, $n$)
(12)   end for each
```

Figure 6: The LSP algorithm

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

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

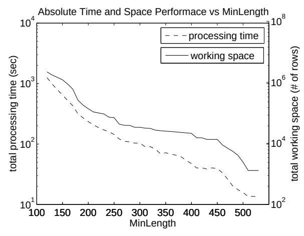
Several implementation details are worth mentioning. First, DeleteUnnecessaryItems deletes items from TF^{L-1}_{long} based on the n–support of items in TF^L_{freq}, not TF^L_{long}. This is important, as it was noted that MinLengthFilter decreases the n–support of items in TF^L_{freq}, thereby possibly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

5 Experimental evaluation

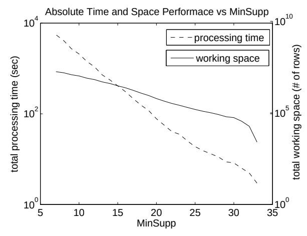
The proposed method was implemented using MS-SQL Server 2000 running on Windows XP on a 3.6GHz Pentium 4 processor with 2GB main memory. The method was tested on the publicly available INFATI data set, which comes from intelligent speed adaptation experiments conducted at Aalborg University. This data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

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

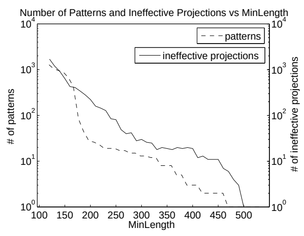
Two sets of experiments were performed, each varying one of the two parameters of the algorithm, *MinSupp* and *MinLength*. The performance of the algorithms was measured in terms of processing time and working space required, where the working space required by the algorithm was approximated by the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



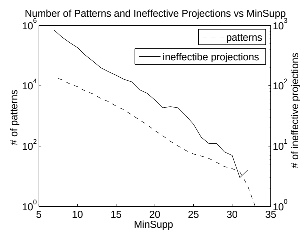
(a) Absolute time and space



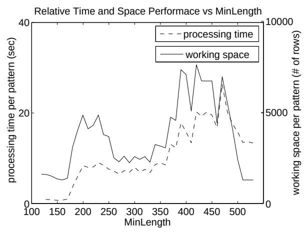
(b) Absolute time and space



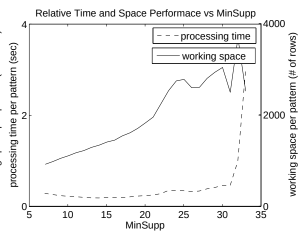
(c) Number of patterns



(d) Number of patterns



(e) Relative time and space



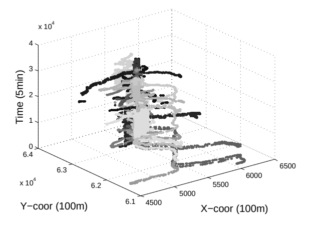
(f) Relative time and space

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

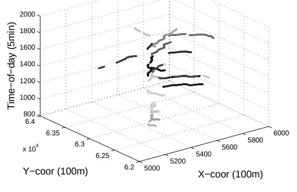
imately *linearly decreasing* as the *MinLength* parameter is decreased. The presence of the two irregular bumps in Figure 7(e) can be explained in relation to the number of patterns found, and the number of ineffective projections that yield no patterns, shown in Figure 7(c). The sharp increases in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

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

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



(a) Moving object trajectories



(b) 28 discovered LSPs

Figure 8: LSP discovery results in INFATI

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

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

6 Conclusions and future work

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

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

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

Acknowledgments

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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