International Journal of Geographical Information Science
Vol. 00, No. 00, Month 200x, 1–23

THIS IS A COPY OF THE POST-PRINT MANUSCRIPT.

The Version of Record of this manuscript has been published and is available in International Journal of Geographical Information Science, published on 2017.10.31 with link: http://www.tandfonline.com/doi/full/10.1080/13658816.2017.1393542

Preferred citation format

Can Yang and Győző Gidófalvi (2018) Mining and visual exploration of closed contiguous sequential patterns in trajectories, International Journal of Geographical Information Science, 32:7, 1282-1304, DOI:10.1080/13658816.2017.1393542

ISSN: 1365-8816 print/ISSN 1362-3087 online

DOI: 10.1080/1365881YYxxxxxxxx
http://www.informaworld.com

International Journal of Geographical Information Science
Vol. 00, No. 00, Month 200x, 2–23

ORIGINAL ARTICLE

Mining and visual exploration of closed contiguous sequential patterns in trajectories

Can Yang * and Győző Gidófalvi

Division of Geoinformatics, Department of Urban Planning and Environment, KTH - Royal Institute of Technology in Sweden, Drottning Kristinas väg 30, Stockholm 10044, Sweden

(*Received 00 Month 200x; final version received 00 Month 200x*)

Large collections of trajectories provide rich insight into movement patterns of the tracked objects. By map matching trajectories to a road network as sequences of road edge IDs, contiguous sequential patterns can be extracted as a certain number of objects traversing a specific path, which provides valuable information in travel demand modeling and transportation planning. Mining and visualization of such patterns still face challenges in efficiency, scalability and visual cluttering of patterns.

To address these challenges, this paper firstly proposes a Bidirectional Pruning based Closed Contiguous Sequential pattern Mining algorithm (BP-CCSM). By employing tree structures to create partitions of input sequences and candidate patterns, closeness can be checked efficiently by comparing nodes in a tree. Secondly, a system called Sequential Pattern Explorer for Trajectories (SPET) is built for spatial and temporal exploration of the mined patterns. Two types of maps are designed where a conventional traffic map gives an overview of the movement patterns and a dynamic offset map presents detailed information according to user-specified filters.

Extensive experiments are performed in this paper. BP-CCSM is compared with three other state-of-the-art algorithms on two datasets: a small public dataset containing clickstreams from an e-commerce and a large GPS dataset with more than 600,000 taxi trip trajectories. The results show that BP-CCSM considerably outperforms three other algorithms in terms of running time and memory consumption. Besides, SPET provides an efficient and convenient way to inspect spatial and temporal variations in closed contiguous sequential patterns from a large number of trajectories.

Keywords: Closed contiguous sequential pattern; Trajectory pattern mining; Trajectory pattern visualization

*Corresponding author. Email: cyang@kth.se

1. Introduction

With the rapid development of positioning and tracking technologies, Global Positioning System (GPS) sensors have been widely deployed to collect movement information from mobile objects such as vehicles, bicycles and pedestrians. The sensors generally report ID, location of the tracked object and timestamp of record periodically. Consequently, continuous movement on a road network, e.g., a taxi trip, is commonly recorded as a trajectory containing a sequence of sampled points with noise. In transportation context, GPS data collected from vehicles is often called floating car data (FCD), which is the focus of this paper. Compared with traditional sensors such as loop detectors collecting velocities and traffic flows at fixed locations (Yildirimoglu and Geroliminis 2013), GPS sensors can record the detailed movement of an individual object with a broad spatial coverage and low cost. With these advantages, FCD has been widely recognized as a valuable data source (Rahmani *et al.* 2010).

In previous studies, various types of movement patterns have been extracted from FCD, such as origin destination patterns (Ferreira *et al.* 2013), pick-up and drop-off patterns (Li *et al.* 2011a, 2012b), inflow and outflow patterns (Guo *et al.* 2012). Without incorporating the entire trajectory into analysis, most of these studies extract patterns either constrained to a local area or in a coarse manner. In order to analyze the details of movement, it is common to snap the noisy trajectories to a road network, which is formulated as a map matching problem (Krumm 2011). In the literature, map matching has been intensively studied with considerable progress in both accuracy and performance (Li *et al.* 2011b, Chen *et al.* 2014), which increases the availability of map matched trajectories and also provides new research opportunities.

By storing a map matched trajectory as a sequence of road edge IDs, a contiguous sequential pattern in a set of trajectories reveals a certain number of objects traversing a specific path or route on the road network. Compared with other aforementioned patterns, contiguous sequential pattern contributes to identifying the set of routes followed by the objects moving from one region to another. That information is valuable in travel demand modeling and transportation planning. However, mining and visualization of contiguous sequential patterns in trajectories still face challenges of increasing data volume, variations in spatial and temporal regularities and heavily overlapping of patterns, i.e., visual cluttering.

To address these challenges, the objective of this paper is to design an algorithm and a set of tools to support efficient and effective mining and visualization of contiguous sequential patterns in large collections of trajectories. The novelty and contributions of this paper are summarized below:

(1) A Bidirectional Pruning based Closed Contiguous Sequential pattern Mining algorithm (BP-CCSM) is developed in this paper. An elegant and efficient pruning procedure is designed in the algorithm by creating partitions of input sequences and candidate patterns where closeness is checked by comparing nodes in a tree as opposed to performing a lot of sequence comparisons.
(2) A system called Sequential Pattern Explorer for Trajectories (SPET) is built with the backend of BP-CCSM. It supports interactive and efficient mining and visualization of patterns by specifying spatial and temporal predicates. To alleviate the problem of visual cluttering, two types of maps are designed where a conventional traffic map can present an overview of a large number of patterns and a dynamic offset map displays detailed information according to user-specified filters.
(3) Extensive experiments are performed in this paper where the computational and

representational aspects of the proposed mining algorithm and system are evaluated. BP-CCSM and three state-of-the-art algorithms are compared with two datasets: a small public dataset containing click streams from an e-commerce and a large taxi trajectory dataset. The results demonstrate that BP-CCSM achieves a superior performance in terms of running time and memory consumption. Besides, SPET provides an efficient and convenient way for exploring spatial and temporal variations of contiguous sequential patterns in trajectories.

The remainder of this paper is organized as follows. Section 2 gives a comprehensive review of related work followed with preliminaries presented in Section 3. Section 4 describes the closed contiguous sequential pattern mining algorithm and visual analytics system. Section 5 presents a case study where the algorithm and system are evaluated with real world datasets. Finally, Section 6 summarizes the main contributions and points to future research directions.

2. Related Work

2.1. *Mining sequential patterns in trajectories*

In previous studies, the approaches proposed for mining sequential patterns in trajectories can be divided into two steps: sequence transformation and pattern mining. In the first step, a trajectory of points is transformed into a sequence where each element can represent ID of a grid (Giannotti *et al.* 2007, Bachmann *et al.* 2013), line segment (Cao *et al.* 2005) or cluster (Ye *et al.* 2009). The second step mines patterns in the set of sequences generated, which belongs to a widely studied problem in data mining called sequential pattern mining. In that field, a large amount of effort has been devoted to mining patterns with a closeness constraint, which significantly reduces the redundancy in the output by pruning the patterns that can be losslessly reconstructed from others. Examples of such algorithms are CloSpan (Afshar 2003), BIDE (Wang and Han 2004), ClaSP (Gomariz *et al.* 2013), CMAP (Fournier-Viger *et al.* 2014). However, they are not directly applicable to mining sequential patterns in map matched trajectories, which imposes an additional contiguity constraint. That constraint preserves the adjacency of elements in the output so that a pattern is extracted as a sequence of spatially contiguous edges representing a path on the road network, which is different from the patterns mined in Cao *et al.* (2005), Giannotti *et al.* (2007) and Ye *et al.* (2009). For a detailed review of general sequential pattern mining algorithms, the reader is referred to Mabroukeh and Ezeife (2010).

Considering both contiguity and closeness constraints, several algorithms have been developed in the literature (Li *et al.* 2012a, Bachmann *et al.* 2013, Zhang *et al.* 2015, Abboud *et al.* 2017). Based on the order of patterns exported, two strategies emerge as by length and by starting item.

The first strategy is used in Zhang *et al.* (2015) where an algorithm called CC-Span is developed and patterns are mined in ascending order of their length. The algorithm iteratively enumerates length-$(k+1)$ patterns from length-(k) patterns, where the former is used to check the closeness of the latter. In each iteration, the whole sequence set is scanned in order to get the support of a pattern, which is inefficient.

The second category exports patterns ordered by their starting item, which reduces the search space greatly so that it is more commonly used in previous studies (Li *et al.* 2012a, Bachmann *et al.* 2013, Abboud *et al.* 2017). In Li *et al.* (2012a), an algorithm

called Gap-BIDE is proposed to mine closed sequential pattern with gap constraint. The problem solved in this paper is equivalent to gap constraint of 0. Gap-BIDE checks closeness by evaluating the extensions of a pattern in forward and backward direction without creating partitions, which still needs to scan the whole sequence set. Along that direction, Abboud *et al.* (2017) develops a more efficient algorithm called CCPM, which recursively creates and scans a smaller projected sequence subset for each pattern. As a pattern grows longer, its projected subset becomes smaller. However, the closeness check still performs a lot of sequence comparisons. That issue is alleviated by the algorithm Incremental Closed Contiguous Frequent Route (Inc-CCFR) developed by Bachmann *et al.* (2013). By recording the positions of a pattern including the sequences containing it and the index within each sequence, non-closed patterns can be pruned efficiently at the sacrifice of much higher memory consumption. Another limitation of Inc-CCFR is that the pattern support is exported as the number of occurrences of a pattern instead of sequences containing it.

As a summary, previous closed contiguous sequential pattern mining algorithms suffer from several common drawbacks. Firstly, the whole sequence set, or a subset of it, needs to be scanned many times in order to find longer candidate patterns. Without exploiting the inner structure of the contiguous sequential patterns, their closeness check either needs to perform a lot of sequence comparisons or stores details of pattern occurrences, which degrades the performance significantly or consumes much memory. The algorithm developed in this paper follows the second strategy of mining patterns by the starting item. By employing three trees to create partitions of sequences and patterns, closeness check is performed by directly comparing nodes in a tree, which achieves a much higher efficiency and saves memory at the same time.

2.2. *Visualization of trajectories*

A large number of visual analytics have been designed to enable the user to discover regularities in trajectories. Andrienko and Andrienko (2013) defines four categories of visual analytics: looking at trajectories, looking inside trajectories, bird's-eye view of movement and investigating movement in context. In this paper, we restrict our review to the visualization approaches applicable to a large number of trajectories, which is faced with two major challenges: visual cluttering and scalability to data size. Three strategies are commonly employed to address these challenges.

The first strategy is to draw raw trajectories with opacity so that areas with a high density of trajectories get highlighted and can be visually inspected, which is widely used in Kraak (2003), Willems *et al.* (2009), Andrienko and Andrienko (2013). It is effective in capturing spatial similarities in trajectories. At the same time, different colors can be assigned to the trajectories to indicate attributes, e.g., ship category (Willems *et al.* 2009). However, quantitative information such as the number of trajectories following a pattern is generally not available.

The second strategy applies filtering to confine the data to be visualized to a small area or subset. In Guo *et al.* (2011), TripVista is designed for visualization of microscopic traffic flow patterns at a road intersection. It provides three views including a traffic view rendering geometry of trajectories, a ThemeRiver metaphor displaying volume of directional flows and a parallel coordinates plot demonstrating the distribution of attributes such as entrance time and speed. In Tominski *et al.* (2012), time lens is designed where its interior displays a map of trajectory points and its external rings plot temporal distribution of trajectories. Since filtering is effective in visualizing patterns locally, in practice

it is commonly used in collaboration with other approaches such as stacked trajectory bands (Tominski *et al.* 2012).

The third strategy employs aggregation or clustering of trajectories to substantially reduce the number of features to be rendered (Andrienko *et al.* 2007, Giannotti *et al.* 2011). In Andrienko *et al.* (2007), large collections of trajectories are clustered and each cluster is simply drawn as an arrow representing directions of movement and the width indicating the size. In Giannotti *et al.* (2011), M-Atlas, for mobility atlas, is designed where trajectories are clustered first then each cluster is visualized individually, which is not applicable in case of a large number of patterns.

Although a contiguous sequential pattern can be regarded as a trajectory with the support as an attribute, its visualization suffers from a more serious visual cluttering problem. One pattern can be contained by many other patterns and their geometries are heavily overlapped. Moreover, the support of a pattern needs to be visualized together with its geometry. To alleviate these problems, the visual analytics system built in this paper adopts the second and third strategy summarized above. It provides two types of maps where a traffic map presents an overview of a large set of patterns and a dynamic offset map presents details of user-filtered patterns. Benefiting from the efficient pattern mining algorithm, sequential patterns in a large number of trajectories can be visually explored efficiently.

3. Preliminaries

Let E be a set of edges in a road network where moving objects can be located. Each edge $e \in E$ is stored as a polyline and connects to one another at their end nodes that coincide with intersections in the road network. For simplicity, e is also referred as the ID of an edge in subsequent text.

After map matching, a trajectory is transformed into a sequence of spatially contiguous edges traversed by the object, denoted by $s = \langle e_1, e_2, \ldots, e_n \rangle$. In data mining terminologies, e_i is also called an *item* in s. A temporal attribute is stored for a trajectory, which is selected as the starting time t_s of a trip in this paper. It is used primarily in partitioning trajectories according to day-of-week $dow \in \{1, \ldots, 7\}$ and hour-of-day $hod \in \{0, \ldots, 23\}$. Consequently, a trajectory tr is stored as a tuple of (s, t_s).

Given a sequence $s = \langle e_1, e_2, \ldots, e_n \rangle$, a *contiguous subsequence* of s is denoted by

$$s' = s[i : j] = \langle e_i, e_{i+1}, \ldots, e_j \rangle$$

where $1 \leq i \leq j \leq n$. Equivalently, the sequence s' is *contiguously contained* by s, denoted by $s' \sqsubset_c s$.

A *trajectory pattern* p is defined as a *contiguous sequential pattern* representing a sequence of spatially contiguous edges denoted by $p = \langle e'_1, e'_2, \ldots e'_m \rangle$. A trajectory tr *follows* p if p is a contiguous subsequence of $tr.s$, namely $p \sqsubset_c tr.s$. Given a trajectory set $TR = \{tr_1, tr_2, \ldots, tr_N\}$, a pattern p in it is also written as $p \in TR$. The number of trajectories *following* p in TR is called the *support* of p, denoted by

$$supp(p) = |\{tr | tr \in TR, p \sqsubset_c tr.s\}|$$

For notational convenience, p is equivalently represented in subsequent figures and text as $\langle e'_1, e'_2, \ldots e'_m \rangle (supp(p))$.

Given a minimum support *min_sup*, p is *frequent* if $supp(p) \geq min\_sup$ meaning that there are at least *min_sup* number of trajectories following p in TR.

A contiguous sequential pattern p is *closed* if there does not exist another contiguous sequential pattern p' such that p is a contiguous subsequence of p' with the same support, denoted by $\nexists p' \in TR : p \sqsubset_c p' \wedge supp(p) = supp(p')$. For instance, $\langle 1, 2, 3 \rangle (2)$ is not closed if there exists another pattern $\langle 1, 2, 3, 4 \rangle (2)$. More formally, when p is closed, it is also called a *Closed Contiguous Sequential Pattern* (CCSP). Unless it is stated explicitly or can be inferred from the context, the term 'pattern' refers to a CCSP in the remainder for simplicity.

4. Methodology

4.1. *Bidirectional pruning based closed contiguous sequential pattern mining algorithm (BP-CCSM)*

The section describes the Bidirectional Pruning based Closed Contiguous Sequential pattern Mining algorithm (BP-CCSM) with the input of a sequence database *SDB* and a minimum support *min_sup*. The sequence database can be obtained from a trajectory set *TR* as $SDB = \{tr.s | tr \in TR\}$. The minimum support *min_sup* can be specified either as an absolute value or a percentage relative to the number of sequences in *SDB*. The output of BP-CCSM is the set of frequent CCSPs in *SDB* whose supports are larger or equal than *min_sup*, as illustrated in Figure 1.

In BP-CCSM, the support of a pattern p is actually exported as the number of occurrences of p in *SDB* instead of the number of sequences containing p in *SDB*. The two values are different only when a pattern occurs multiple times within a sequence. For instance, by observing $\langle 1, 2, 3, 4, 1, 2 \rangle$, the number of occurrences of pattern $\langle 1, 2 \rangle$ is incremented by 2 whereas its support is only incremented by 1. In map matched trajectories, that kind of sequence containing repeated items is rarely observed, which can be easily filtered out from the input data without significant impact on the mining result. Therefore, the support exported is consistent with the definition in the preliminaries.

BP-CCSM can be divided into four steps. In principle, the first step creates a prefix partition tree from *SDB* where infrequent items are filtered out. The tree is used in later steps in replacement of *SDB*. Step 2 enumerates all candidate patterns from the tree. After that, non-closed patterns are pruned in backward and forward direction in Step 3 and 4 respectively. Details are explained below with an example illustrated in Figure 1.

4.1.1. STEP 1: Filtering and partitioning of sequences

The objective of the first step of BP-CCSM is to create partitions for input sequences and exclude infrequent items so that patterns can be examined efficiently without scanning *SDB* in later steps, which is realized by employing a frequent pattern tree (FP-tree) proposed by Han *et al.* (2004).

The FP-tree constructed from *SDB* consists of two parts. The first part is a tree which has a root node *root* with item-prefix subtrees as children. The second part is an header table $\mathcal{HT}$ that stores the count of each item i in *SDB* denoted by $\mathcal{HT}(i).count$ and a linked-list connecting all the nodes storing i in the tree, denoted by $\mathcal{HT}(i).nodelist$.

The following definitions of node and tree are applicable to both FP-tree and other types of trees created in later steps. A node n stores two attributes: item id i and count c, denoted by $n.item = i$ and $n.count = c$. A tree of nodes is denoted by $\mathcal{T}$. A sequence $s = \langle i_1, i_2, ..., i_m \rangle$ is inserted into $\mathcal{T}$ as a path starting from *root*, denoted by

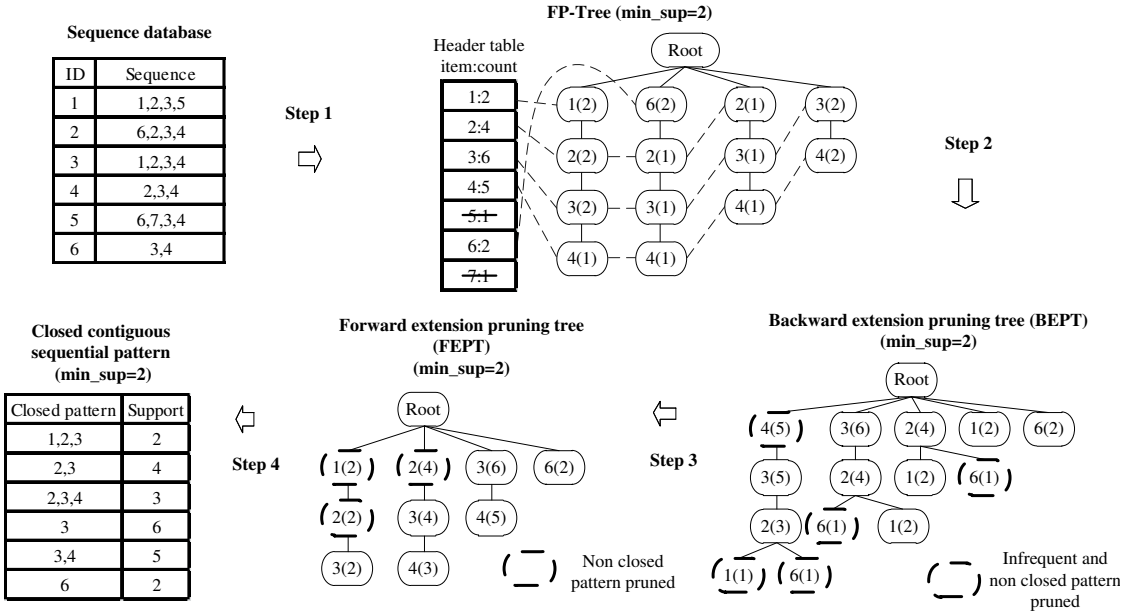


Figure 1. Illustration of BP-CCSM algorithm. Infrequent items are strike-through in the header table. Inside a node in the trees, the number is the item id and the count is wrapped in parenthesis. In BEPT, any pattern at a node that has a count smaller than *min_sup* (infrequent) or has a child with the same count (non-closed) is pruned. In FEPT, any pattern at a node that has a child with the same count (non-closed) is pruned.

$root \rightarrow n_1 \rightarrow \cdots \rightarrow n_m$ where $n_j.item = i_j, 1 \leq j \leq m$ and the sign $\rightarrow$ denotes that the latter node is a child of the former. Let $\pi(n_m)$ denote the sequence of items from $root$ to n_m, namely $\pi(n_m) = \langle i_1, i_2, ..., i_m \rangle$. During the insertion of sequences, the count of a node n_m is maintained either as the sum or the maximum of the counts of all the inserted sequences that have $\pi(n_m)$ as a prefix, denoted by *sum count maintenance* and *maximum count maintenance* respectively.

Based on the above definitions, an FP-tree is constructed through two scans of *SDB* described below:

S-1.1 In the first scan of *SDB*, record the count of each item i in a sequence $s \in SDB$ in the table $\mathcal{HT}$.

S-1.2 In the second scan, insert each sequence $s \in SDB$ into an FP-tree with *sum count maintenance*. Whenever an infrequent item i' is encountered in s, i.e., $\mathcal{HT}(i').count < min\_supp$, discard it and insert the remainder of s into the tree starting from *root*. Whenever a new node is created to store item i, append it to the corresponding linked list stored in $\mathcal{HT}(i).nodelist$.

An example is illustrated in Figure 1 where the FP-tree is created with $min\_sup = 2$ and sequences are inserted in the ascending order of ID. In the first scan, two infrequent items are identified as 5 and 7 with a support of 1. In the second scan, when $\langle 6, 7, 3, 4 \rangle$ is inserted into the FP-tree, the infrequent item 7 is discarded. In detail, after item 6 is inserted to the position $root \rightarrow 6$, item 7 is detected as infrequent and discarded according to the count statistics in $\mathcal{HT}$. The remaining subsequence $\langle 3, 4 \rangle$ is reinserted from *root*, which creates a new path $root \rightarrow 3 \rightarrow 4$ with count of 1. The next sequence inserted is $\langle 3, 4 \rangle$ thus the counts of node 3 and 4 on the path $root \rightarrow 3 \rightarrow 4$ are incremented by 1 according to the sum count maintenance adopted by FP-tree.

A node n in FP-tree indicates that there are $n.count$ number of *contiguous subsequences*

of frequent items in *SDB* that have $\pi(n)$ as a prefix. For instance, the node 4(2) on the path $root \rightarrow 3 \rightarrow 4$ represents that there are 2 *contiguous subsequences of frequent items* in *SDB* that have the prefix of $\langle 3, 4\rangle$, which is counted from sequences $\langle 6, 7, 3, 4\rangle$ with 7 discarded and $\langle 3, 4\rangle$ in *SDB*. It is important to note that a frequent pattern is always a *contiguous subsequence of frequent items* but the reverse is not true. For example, in Figure 1, $\langle 6, 2, 3\rangle$ is a subsequence of frequent items but not a frequent pattern.

4.1.2. STEP 2: Enumerating candidate contiguous sequential patterns

STEP 2 of BP-CCSM enumerates all candidate contiguous sequential patterns from the FP-tree constructed previously. The term *candidate* is used because a pattern constituted by frequent items is not guaranteed to be a frequent pattern. The enumerated patterns are inserted in a reversely way into a tree which is later used for backforward extension pruning hence it is called BEPT. That tree can be regarded as a suffix partition of candidate contiguous sequential patterns in *SDB*. Details are described below:

S-2 For each frequent item i in $\mathcal{HT}$, visit each node n stored in the linked-list $\mathcal{HT}(i).nodelist$, insert the sequence $\pi(n)$ with count of $n.count$ **in a reverse order** into BEPT using the *sum count maintetance*.

As shown in Figure 1, the node 3(1) on the path $root \rightarrow 2 \rightarrow 3$ in the FP-tree is inserted in BEPT as $root \rightarrow 3 \rightarrow 2$. Since the sum of count is maintained at each node and all nodes are examined in FP-tree, finally all the occurrences of $\langle 2, 3\rangle$ in FP-tree are inserted to that path $root \rightarrow 3 \rightarrow 2$ in BEPT with their counts accumulated. Finally node 2 on the path has a count of 4.

Different from FP-tree, a node n in BEPT indicates that a *contiguous sequential* pattern as a **reverse** of $\pi(n)$ exists in *SDB* with support of $n.count$. In the BEPT shown in Figure 1, the node 2(4) along the path $root \rightarrow 3 \rightarrow 2$ corresponds to the pattern $\langle 2, 3\rangle(4)$ in *SDB*.

4.1.3. STEP 3: Support and backward extension based pruning of patterns

STEP 3 of BP-CCSM prunes patterns if either it is infrequent in *SDB* or it is not closed in the backward direction. The pruning is performed on BEPT as follows:

S-3.1 For each node n in BEPT, if (a) $n.count < min\_supp$ or (b) if there exists a child node n' of n where $n'.count = n.count$, then prune n. It is equivalent to prune the corresponding pattern as a reverse of $\pi(n)$;

S-3.2 For any the non-pruned node n in BEPT, insert $\pi(n)$ in **a reverse order** into a tree named forward-extension pruning tree (FEPT) using the *maximum count maintenance*.

In STEP 3.1, condition (a) implies that $\pi(n)$ is infrequent while condition (b) prunes $\pi(n)$ based on the following observation. In BEPT if n has a child node n' with the same support $n.count = n'.count$, then $\pi(n')$ represents a *minimal sized backward-extension* of the pattern $\pi(n)$ and $supp(\pi(n')) = supp(\pi(n))$, thus $\pi(n)$ is not closed and pruned. In the BEPT shown in Figure 1, the node 4(1) on the path $root \rightarrow 6 \rightarrow 2 \rightarrow 3 \rightarrow 4$ is pruned because the corresponding pattern $\langle 6, 2, 3, 4\rangle(1)$ is infrequent. The node 4(5) on the path $root \rightarrow 4$ is pruned since its child node 3(5) has the same count. Equivalently, the pattern $\langle 4\rangle(5)$ is pruned because it has a minimal sized backward-extension pattern with the same support as $\langle 3, 4\rangle(5)$.

Since each sequence in BEPT is inserted in a reverse order into FEPT with *maximum count maintenance*, a node n in FEPT indicates that a contiguous sequential pattern as

Table 1. Summary of data structures used by BP-CCSM

Data structure	Information stored	Count maintenance	Partition order
FP-tree	Contiguous subsequences of frequent items	sum	prefix
BEPT	All contiguous sequential patterns of frequent items	sum	suffix
FEPT	Contiguous sequential patterns not pruned in BEPT	maximum	prefix

$\langle\pi(n)\rangle(n.count)$ exists in *SDB*. In the FEPT shown in Figure 1, the node 3(3) along the path $root \rightarrow 2 \rightarrow 3 \rightarrow 4$ indicates that the support of pattern $\langle 2,3,4\rangle$ is 3.

The reason for the *maximum count maintenance* adopted by FEPT in STEP 3.2 is that the inserted sequences are patterns and it is incorrect to sum the supports of two overlapping patterns as they account for overlapping subsets in *SDB*. For example, when inserting the two patterns $p_a = \langle 2,3,4\rangle(3)$ and $p_b = \langle 2,3\rangle(4)$ respectively from node 2(3) on $root \rightarrow 4 \rightarrow 3 \rightarrow 2$ and 2(4) on $root \rightarrow 3 \rightarrow 2$ from BEPT into FEPT, the correct support of pattern $\langle 2,3\rangle$ in FEPT is $\max(supp(p_a), supp(p_b)) = 4$ rather than $supp(p_a) + supp(p_b) = 7$.

4.1.4. STEP 4: Forward extension based pruning of patterns

STEP 4 of BP-CCSM essentially uses a similar technique as STEP 3 with the difference that only forward pruning is performed without checking infrequent patterns which have been pruned in STEP 3. Detailed steps are given below:

S-4 Iterate through all nodes in FEPT, whenever a node n has a child n' where $n.count = n'.count$, prune n ; otherwise export $\pi(n)$ as a pattern with support of $n.count$.

This pruning is based on the observation that in FEPT when $n'.count = n.count$, $\pi(n')$ is a minimal sized forward-extension of pattern $\pi(n)$ with the same support indicating that $\pi(n)$ is not closed and should be pruned. For example, in the FEPT in Figure 1, $\langle 1,2\rangle(2)$ is pruned because of $\langle 1,2,3\rangle(2)$. Finally all CCSPs are exported.

4.1.5. Summary of BP-CCSM

The data structures used in BP-CCSM are summarized in Table 1. The count maintenance adopted by a tree depends on whether the counts of inserted sequences can be summed up (e.g., trajectories) or not (e.g.,patterns). In BP-CCSM, BEPT is a suffix tree while FP-tree and FEPT are prefix trees, which matches with the two reverse insertions from FP-tree to BEPT and from BEPT to FEPT. Finally CCSPs are exported in the same sequential order as *SDB*.

It is important to note that in implementation BEPT may consume a lot of memory since it contains all contiguous sequential patterns of frequent items in *SDB*. When a long sequence with M distinct frequent items is inserted into FP-tree as a path with M nodes. For every node n in that path, $\pi(n)$ is inserted into BEPT in a reverse order and a new branch is created. Totally, at most $0.5M(M+1)$ nodes are created in BEPT corresponding to that path. Based on the above observation, the number of nodes in BEPT is $O(M^2)$ if the number of nodes in the input FP-tree is M. To address the memory consumption problem, each branch under the *root* in BEPT can be created and evaluated individually in STEP 2 instead of creating a single large BEPT. Since each branch is evaluated independently in STEP 3.1, the same result will be obtained finally.

4.1.6. Proof of correctness

The following lemmas are used in the proof of correctness of BP-CCSM.

Lemma 4.1: *If a contiguous sequential pattern p is constituted by frequent items in SDB, regardless of being closed or frequent, there exists a node n in BEPT where $\pi(n)$ is the reverse of p and $n.count = supp(p)$.*

Proof: Assume $p = \langle e'_1, e'_2, \ldots, e'_n \rangle$ where all the items are frequent, all the occurrences of p in *SDB* are explicitly present in FP-tree as a path $e'_1 \rightarrow e'_2 \rightarrow \cdots \rightarrow e'_n$ which can be distributed in different branches. For each node n' in FP-tree, $\pi(n')$ is inserted in a reverse order into BEPT. Therefore, there exists a path in BEPT as $root \rightarrow e'_n \rightarrow \cdots \rightarrow e'_1$. Because *sum count maintenance* is used by BEPT, all the occurrences of p are finally summed up as the support of p. □

Lemma 4.2: *If a contiguous sequential pattern p is infrequent or non-closed in SDB, BP-CCSM prunes p.*

Proof: When p is infrequent, according to Lemma 4.1, p is present on BEPT and pruned by *min_sup*. When p is non-closed, the case is discussed below. Let the sign $\cdot$ denote an arbitrary item. One of the following 3 conditions must be satisfied:

- C1: There exists a pattern p_b that is a minimum backward extension of p with the same support, denoted by $p_b = \langle \cdot, p \rangle$ and $supp(p_b) = supp(p)$. According to Lemma 4.1, both p and p_b are present in BEPT and the node that corresponds to p_b is a child of the node that corresponds to p with the same count. Therefore p is pruned in STEP 3.1;
- C2: There exists a pattern p_f that is a minimum forward extension of p with the same support, denoted by $p_f = \langle p, \cdot \rangle$ and $supp(p') = supp(p)$. If p_f is not pruned in STEP 3.1, then p_f is inserted into FEPT where the node of p_f is a child of the node corresponding to p with the same support thus p is pruned in STEP 4. If p_f is pruned in STEP 3.1, it implies that the node of p_f in BEPT has a child with the same count, which corresponds to a pattern $p_{bf} = \langle \cdot, p_f \rangle = \langle \cdot, p, \cdot \rangle$ and $supp(p_{bf}) = supp(p)$, which is covered in C3;
- C3: There exists a pattern p_{bf} that has both a minimum backward and forward extension of p with the same support, denoted by $p_{bf} = \langle \cdot, p, \cdot \rangle$ and $supp(p_{bf}) = supp(p)$. Then the pattern $p_b = \langle \cdot, p \rangle$ is also presented in the data with the same support of p because $supp(p_{bf}) \leq supp(p_b) \leq supp(p)$, which is covered in C1 where p is pruned.

Therefore, no matter which one of the three conditions is met, p is pruned. □

Lemma 4.3: *If a contiguous sequential pattern p is frequent and closed in SDB, BP-CCSM outputs p.*

Proof: According to Lemma 4.1, p will be present in *BEPT* and not pruned since it is frequent and closed. Then in FEPT, p is not pruned neither. Therefore, BP-CCSM outputs p. □

Based on Lemma 4.2 and 4.3, the correctness of BP-CCSM is proved .

4.1.7. Pattern filtering with BP-CCSM

From the last step of BP-CCSM shown in Figure 1 it can be observed that FEPT can be treated as a prefix partition of all frequent CCSPs in *SDB*. The property enables efficient pattern filtering on the fly. Algorithm 1 explains the pattern filtering by length

Algorithm 1: Pattern filtering by length and support

Input :
$fept$: an FEPT tree generated in the last step of BP-CCSM
L_{min}, L_{max}: pattern length range
S_{min}, S_{max}: pattern support range
Output:
FPS: filtered pattern set

```
/* Initialize an empty stack and a filtered pattern set */
1  Stack S=∅
2  FPS=∅
3  S.push(fept.root)
4  while S is not empty do
5      Node n := S.pop()
6      L_n = |π(n)| // Pattern length at n
7      if L_min ≤ L_n ≤ L_max and S_min ≤ n.count ≤ S_max then
8          if n is closed then
9              FPS.insert({π(n), n.count}) // a filtered pattern is found
10         S.push_children(n)
11     else if L < L_min or n.count ≥ S_min then
12         S.push_children(n)
13 return FPS
```

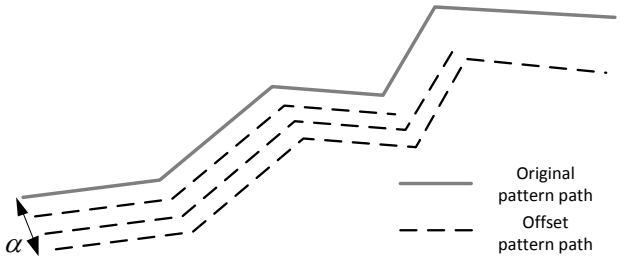


Figure 2. Illustration of the dynamic offset map

and support. The condition check from line 7 to 11 is based on the observation that if n' is a child node of n in FEPT then $L_{n'} > L_n$ and $n'.count \leq n.count$. Therefore, specific set of nodes can be pruned without inserting into the node stack.

4.2. *Visual exploration of trajectory sequential patterns*

4.2.1. Visualization of CCSP

Two types of maps are designed for the visualization of CCSP. The first one is a traditional traffic map where each edge e is shown as a polyline with color encoded by the number of trajectories passing e. The statistics can be obtained easily from FEPT created in the last step of BP-CCSM by collecting the maximum count for each item in the tree. However, patterns traversing multiple edges are hidden in the traffic map. For instance, it can show that there are 100, 200 and 300 trajectories passing the edge with ID of 1, 2, 3 respectively but the fact that only 50 trajectories pass the path of

$\langle 1, 2, 3 \rangle$ is not visible. Therefore, the traffic map is useful in presenting an overview of the movement recorded in a large number of trajectories.

The second type of map is called *dynamic offset map*, where each pattern is drawn as a polyline with a dynamic offset δ from its original geometry, as illustrated in Figure 2. Given a pattern set PS, let L^{PS}_{max} and L^{PS}_{min} denote the maximum and minimum pattern length in it. The geometry of a pattern p is offset by δ

$$\delta = \frac{length(p) - L^{PS}_{min}}{L^{PS}_{max} - L^{PS}_{min}} \times \alpha$$

where α is a parameter controlling the offset magnitude. The design of offset guarantees that δ increases with the pattern length so that longer patterns tend to be placed at the outside. Consequently, the overlapping of p with its super patterns can be alleviated. More concrete examples of dynamic offset map can be found in Section 5.4.2.

Compared with the traditional traffic map, dynamic offset map can display the complete path of a pattern together with its support. However, the offset transformation introduces extra computation. The dynamic offset map requires rendering a set of complete trajectories as opposed to a set of short edges visualized in the traffic map. In practice, it is useful in obtaining a detailed view of a small set of patterns.

As a summary, the traffic map is effective in presenting an overview of the movement whereas dynamic offset map is applicable to visualizing a small set of patterns according to user's interest, which can be preferably used with trajectory and pattern filtering.

4.2.2. Web based visual analytics

Based on the above two types of maps, a system called Sequential Pattern Explorer for Trajectories (SPET) is built with the architecture demonstrated in Figure 3. The modules are explained below:

- Trajectory partitioning: The original map matched trajectory set TR is partitioned according to the temporal attribute of each trajectory, e.g., day-of-week (*dow*) and hour-of-day (*hod*).
- Trajectory filtering and aggregation: This module collects trajectories according to user specified spatial and temporal predicates, e.g., a region where taxi trips depart from on Monday 6:00 - 10:00.
- Pattern mining and filtering: The module employs BP-CCSM to mine the trajectories collected in the previous step. The patterns generated can be further filtered by length and support.
- Web based visual analytics: This module provides interface for the user to access the functionalities of SPET. Details are presented below.

The user interface of the visual analytics is demonstrated in Figure 4. Controllers are placed to the left and bottom of a map where the user can specify *dow*, *hod* and *min_sup*. Patterns are displayed on the interactive map where the user can draw a rectangle to specify the filtering region, as highlighted in purple dashed style.

In the visual analytics, traffic map is loaded by default where the legend shows a percentage of the traffic count or pattern support relative to the number of filtered trajectories so that movement in various periods can be compared. The default classification scheme of pattern support is 4 classes in logarithm scale decorated with colors from green to red. It is based on the logarithm distribution followed by pattern support shown in Figure 5 (d). A setting panel is provided where classfication scheme, map type, pattern filtering and offset parameters can be adjusted, as depicted in the right of Figure 4.

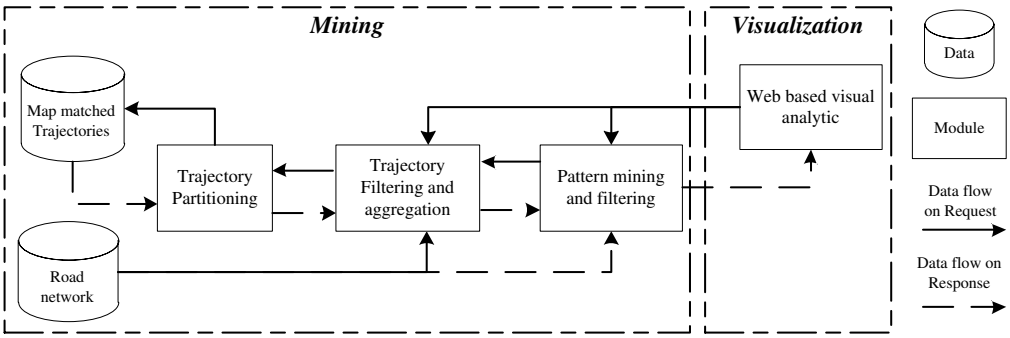


Figure 3. Conceptional architecture of SPET

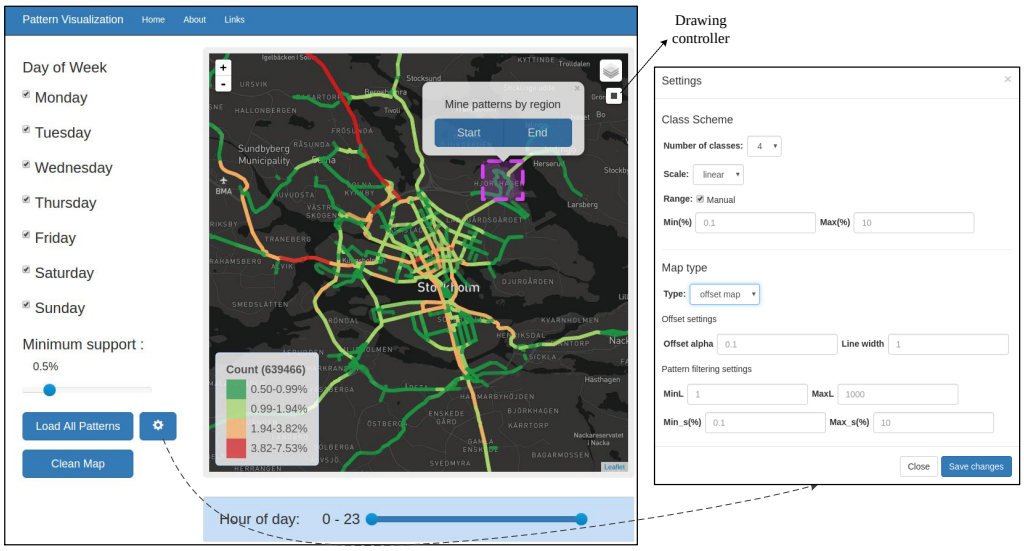
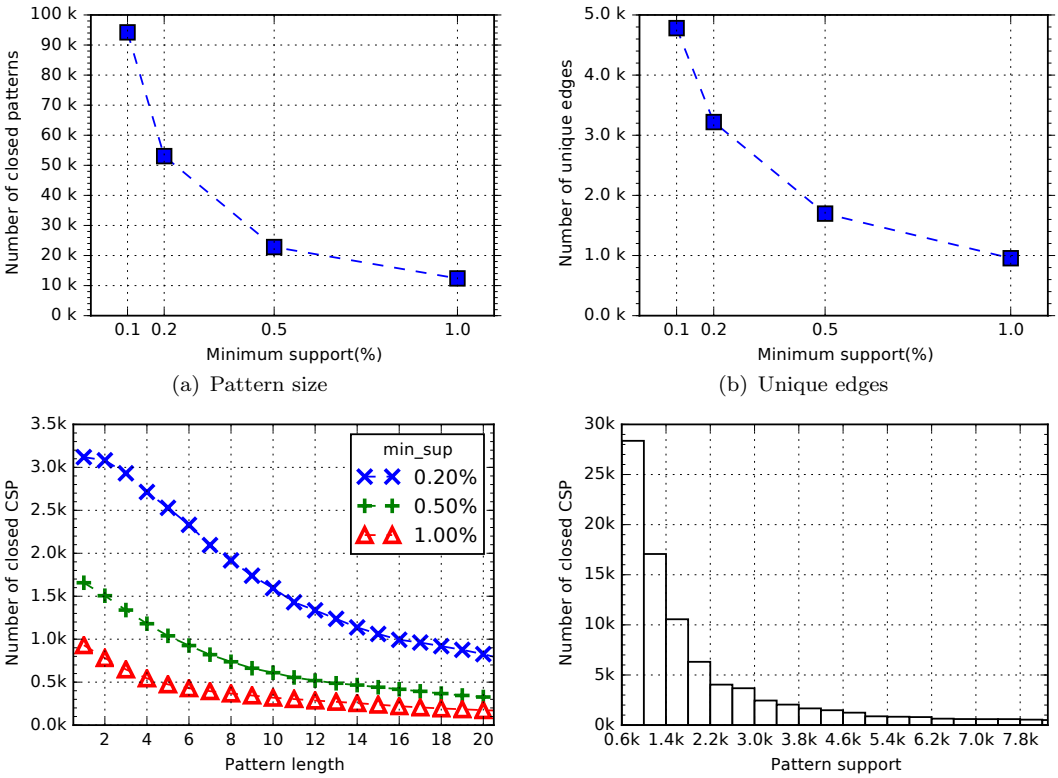


Figure 4. User interface of the web based visual analytics.

5. Case study

5.1. *Data description and study area*

In this case study, the taxi GPS data originally collected in the Mobile Millennium Project (Allström *et al.* 2011) is used for evaluation of the algorithm and visual analytics system. It is collected from about 1,500 vehicles in Stockholm, Sweden, which covers one month period from 2013-03-01 to 2013-03-31 and contains about 13 million records. Each record stores 5 fields including vehicle ID, timestamp, latitude, longitude and a hired state, which is a boolean value indicating whether a taxi is occupied or not. A taxi trip is extracted as a polyline formed by a sequence of consecutive records with true hired states. In total 639,466 trajectories are extracted, which are map matched to a road network containing 23,921 nodes and 57,928 directed edges using an algorithm similar to Krumm (2011). About 0.5% of the trajectories containing repeated edges are filtered out. Since the case study is concentrated on evaluating the mining algorithm and visual analytics system, which is influenced little by the map matching quality, we assume that all the trajectories are perfectly matched to the road network. After map matching, the raw trajectories are transformed into sequences of edge IDs. The number of edges contained in a trajectory ranges from 1 to 509 with an average value of 38.9.



(a) Pattern size (b) Unique edges

(c) Pattern length (d) Pattern support

Figure 5. Statistics of the patterns mined from the one-month trajectories shown by pattern size (a), unique edges (b), pattern length (c) and support (d). In (d), the histogram shows the result mined with $min\_sup = 0.1\%$ (639 in absolute value).

The number of unique edges traversed by the trajectories is 30,214.

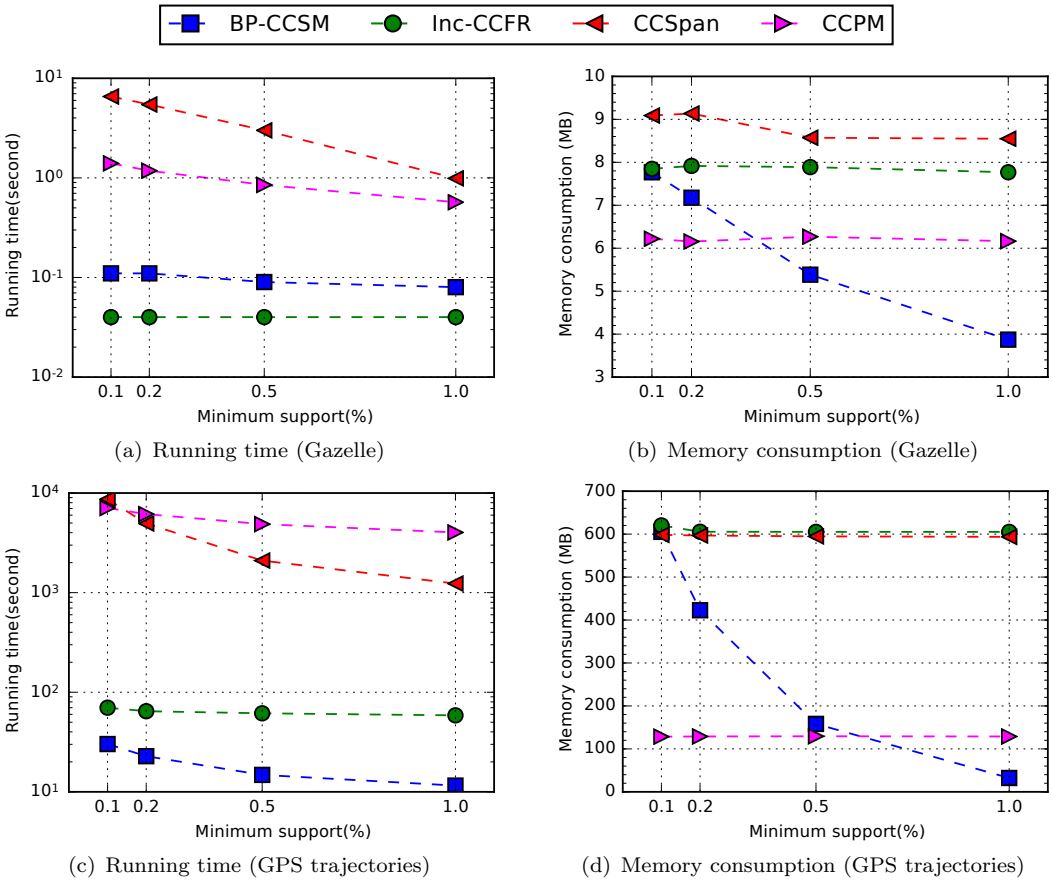
5.2. *Experiment setup*

The algorithm is implemented in C++ and all the experiments are performed on a desktop computer running Ubuntu 14.04 operating system with Intel Quad Core CPU 3.00 GHz and 4GB RAM. The experiments are described in two parts: pattern mining and pattern visual exploration.

5.3. *Pattern mining experiments*

In the first experiment, all the trajectories in the one month period are mined using BP-CCSM with four *min_sup* tested 0.1%, 0.2%, 0.5% and 1.0% relative to the total number of trajectories. The results are displayed in Figure 5. It can be observed that both the number of patterns and unique edges in them reduce substantially as *min_sup* increases. It is expected because a higher *min_sup* will filter out more infrequent items in the first step of BP-CCSM. Additionally, Figure 5(c) shows that pattern length follows a consistent trend where there are more patterns with a shorter length. On the other hand, pattern support fulfills a logarithm distribution as shown in Figure 5 (d).

Regarding performance, BP-CCSM is compared with three other algorithms: Inc-



(a) Running time (Gazelle)

(b) Memory consumption (Gazelle)

(c) Running time (GPS trajectories)

(d) Memory consumption (GPS trajectories)

Figure 6. Performance comparison of BP-CCSM with Inc-CCFR, CCSpan and CCPM with two datasets: Gazelle (a,b) and GPS Trajectories (c,d). The Gazelle dataset contains 59,601 sequences formed by 497 items with average length of 2.5 and maximum length of 267. The GPS dataset contains 639,466 sequences formed by 30,214 items with average length of 38.9 and maximum length of 509.

CCFR (Bachmann *et al.* 2013), CCSpan (Zhang *et al.* 2015) and CCPM (Abboud *et al.* 2017), which are all implemented in C/C++ and tested in the same environment[1]. In addition to the GPS trajectories, another public dataset Gazelle BMS1 is tested in this experiment, which contains sequences representing click streams of an e-commerce and is available at the SPMF website[2]. It is used in KDD Cup-2000 competition as well as some previous work for performance evaluation (Zhang *et al.* 2015, Abboud *et al.* 2017). The dataset contains 59,601 sequences formed by 497 items with an average length of 2.5 and maximum length of 267. None of them contains repeated items.

When tested on the small Gazelle dataset as shown in Figure 6 (a) and (b), BP-CCSM outperforms CCSpan and CCPM with the speed increased by 60 and 10 times respectively. BP-CCSM doubles the running time of Inc-CCFR but both of them take a rather short time less than 0.1 seconds. That difference can be omitted in practice. It could be explained by the observation that Gazelle dataset is small and sequences are

[1]The CCSpan and CCPM are implemented in C++ by the authors of this paper based on the original articles with some optimizations in sequence comparison and pattern output. Therefore, higher performance and lower memory consumption are reported here compared with the experiments in the original papers.
[2]http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

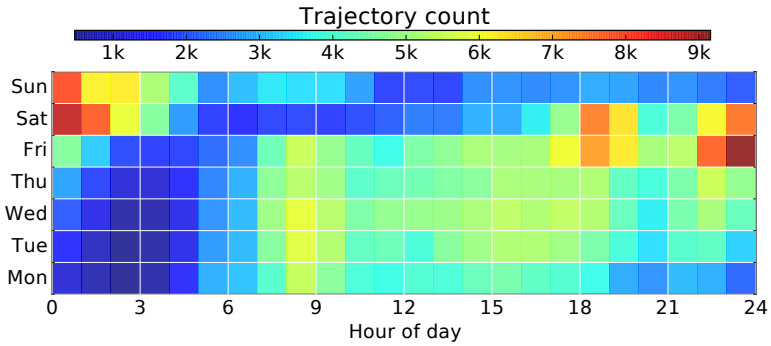


Figure 7. Distribution of trajectories with respect to day-of-week and hour-of-day.

short. In that case, creating three partition trees may take a longer time than iterating through the sequences. On the other hand, BP-CCSM consumes less memory as *min_sup* increases while the other three algorithms exhibit no significant change. The reason is that with a higher *min_sup*, more items become infrequent so that smaller trees are created and processed in BP-CCSM.

When tested on the much larger GPS trajectory dataset, both the running time and memory rise sharply compared with Gazelle, as displayed in Figure 6 (c) and (d). Among the four algorithms, BP-CCSM is the fastest one whose speed is about 2-5 times of Inc-CCFR, 300 times of CCSpan and CCPM. CCSpan and Inc-CCFR consume consistent memory about 600 megabytes (MB), which is 5 times of CCPM. As *min_sup* increases from 0.1% from 1.0%, the memory of BP-CCSM decreases rapidly from 600 MB to 30MB.

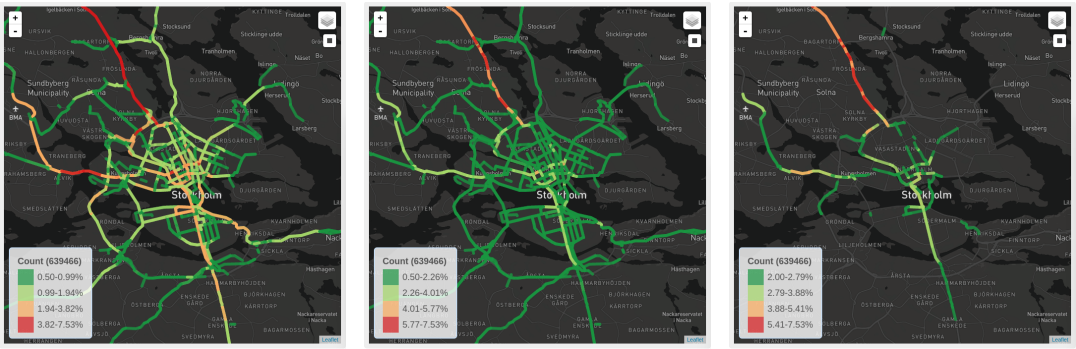
Based on the above experiments, we conclude that BP-CCSM outperforms the other three algorithms considering speed, memory consumption and scalability to data size. It is also important to be aware that both BP-CCSM and Inc-CCFR mine the pattern support as the number of occurrences instead of the number of sequences containing a pattern, as discussed in Section 4.1. Therefore, when there exist a large number of sequences containing repeated items, e.g., DNA sequences, the two types of supports can be quite different and CCSpan or CCPM should be used.

5.4. *Pattern visual exploration experiments*

In the pattern visual exploration experiments, all the trajectories are firstly partitioned by day-of-week and hour-of-day into $7 \times 24 = 168$ subsets. The distribution of trajectories in these subsets is shown in Figure 7 where three clusters can be identified. The first cluster has the highest density above 6,000 trips per hour, which covers the periods from Friday and Saturday night to the morning of the next day. The second cluster with 4,000 to 6,000 trips observed per hour is formed by the periods from Monday to Thursday 7:00 to 24:00. The rest periods constitute the third cluster with a density below 2,000 trips per hour. The distribution of trajectories guides the selection of weekday and weekend periods as well as morning and afternoon rush hour in the visual exploration experiments. The traffic map and dynamic offset map in SPET are separately presented below.

5.4.1. Traffic map visualization

Three scenarios are evaluated with the traffic map in SPET. In the first scenario, patterns mined with various *min_sup* are visualized with neither spatial nor temporal predicate. The results are displayed in Figure 8. From Figure 8 (a) and (b) it can be



(a) (b) (c)

Figure 8. Pattern exploration examples by varying *min_sup* and classification scheme mined with 168 time windows: (a) *min_sup*=0.5% with logarithm scale, (b) *min_sup*=0.5% with equal interval scale (c) *min_sup*=2.0% with logarithm scale.

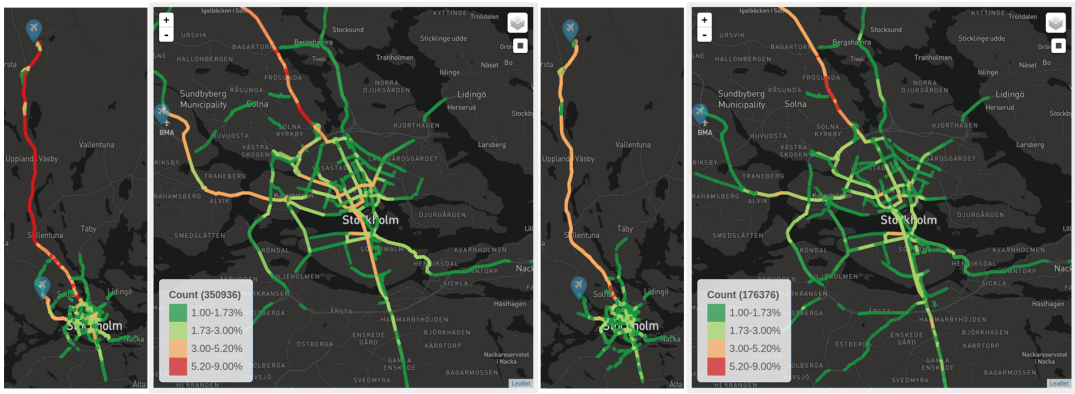
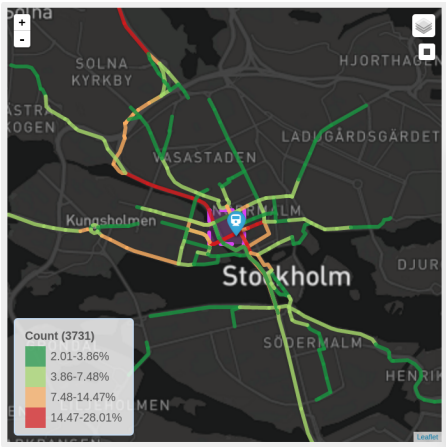


Figure 9. Distribution of trajectory patterns on weekdays, i.e., 0:00 - 24:00 from Monday to Thursday (left) and weekends, i.e., 0:00 - 24:00 on Saturday and Sunday (right) mined with the one-month trajectories and *min_sup*=1%. In the zoomed-out view, two airports are marked where the one at the top is Arlanda and the middle is Bromma. The same class range is specified.

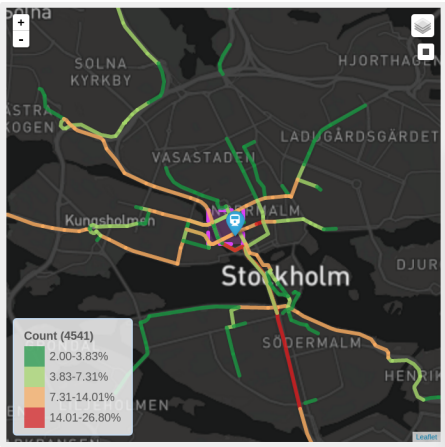
observed that logarithm scale is more appropriate for identifying variations in pattern supports than equal interval scale. It can be explained by the distribution of pattern supports in Figure 5(d). On the other hand, dominant patterns get highlighted by increasing *min_sup* as depicted in Figure 8 (c).

The second scenario compares the distribution of weekday and weekend patterns. Based on Figure 7, weekday periods are selected as 0:00 - 24:00 from Monday to Thursday and weekend as 0:00 - 24:00 on Saturday and Sunday. The comparison is presented in Figure 9. On both weekdays and weekends, the patterns with the highest support appear on the highway connecting city center of Stockholm with the Arlanda airport at the north, where the patterns on weekdays have a higher relative support. Some differences can also be captured close to the Bromma airport and in some suburb areas.

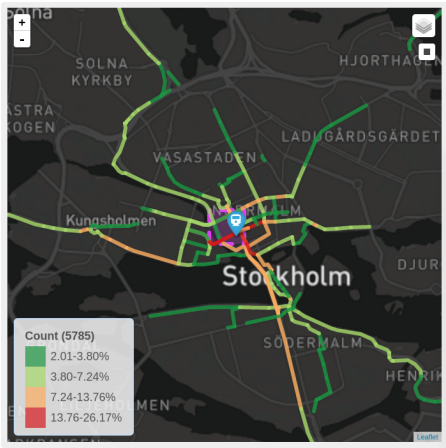
The third scenario explores patterns with spatial and temporal predicates defined as filtering trajectories departing from and arriving at the train station in the morning and evening rush hour, as shown in Figure 10. The morning and evening rush hour are identified from Figure 7 as 7:00 - 10:00 and 16:00 - 19:00 respectively. Figure 10 (a) shows that in the morning rush hour a majority of trips departing from the train station move towards the north of the city. At the same time, more trips arrives at the train



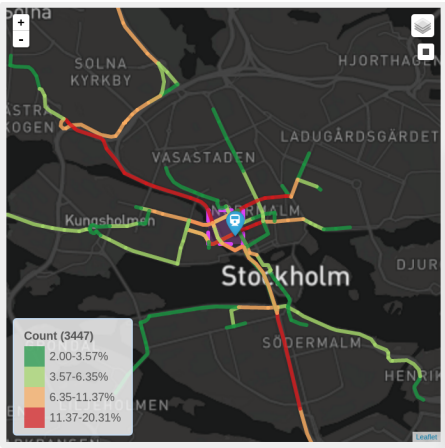
(a) Depart from train station P1



(b) Arrive at train station P1



(c) Depart from train station P2



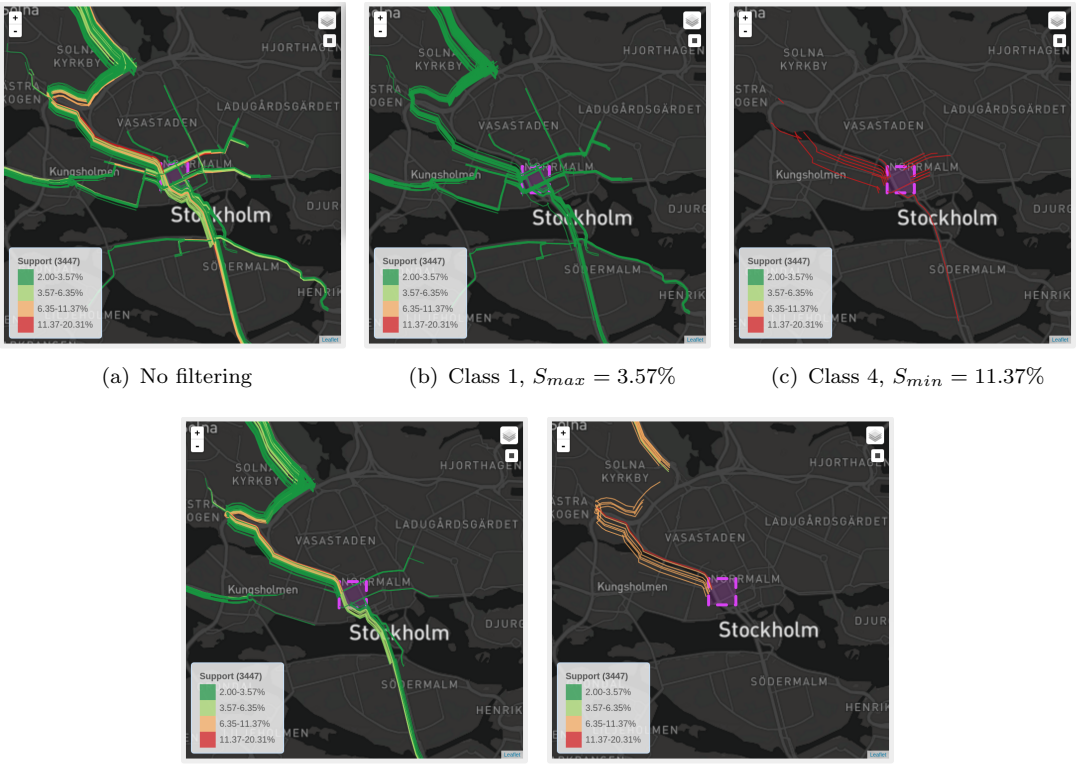
(d) Arrive at train station P2

Figure 10. Distribution of trajectory patterns in the trips departing from and arriving at the train station in the morning rush hour 7:00 - 10:00 (P1) and afternoon rush hour 16:00 - 19:00 (P2). The patterns are mined from the one-month trajectories with $min\_sup = 2\%$ relative to the number of filtered trajectories. The location of the train station is shown with a marker and the query region is displayed as a dashed purple rectangle.

station from the south than other directions, as shown in Figure 10 (b). The movement in the afternoon rush hour follows a reverse trend as displayed in Figure 10 (c) and (d). By comparing the geometries of patterns in different periods, it reveals that two sets of routes are followed by the trips starting from and arriving at the train station respectively.

5.4.2. Dynamic offset map visualization

Dynamic offset map is explored with the same set of predicates used in Figure 10(d). The results are demonstrated in Figure 11. In (a) where no pattern filter is applied, the visual cluttering problem can still be serious but the complete paths of the patterns are visible. In (b), only patterns in Class 1 are drawn by setting the support filter as $S_{max} = 3.57\%$. Similarly in (c), only patterns in Class 4 are visualized with $S_{min} = 11.37\%$. As can be observed, dynamic offset map gives a clear view of the filtered patterns with little cluttering. In addition to the support filter, pattern length filter can also be applied as demonstrated in (d) and (e), where short patterns are filtered out.



(a) No filtering (b) Class 1, $S_{max} = 3.57\%$ (c) Class 4, $S_{min} = 11.37\%$

(d) $L_{min} = 10$ (e) $L_{min} = 10$ and $S_{min} = 6.35\%$

Figure 11. Dynamic offset map visualization with different kinds of pattern filters. The patterns are mined with $min\_sup = 2\%$ from trajectories arriving at the train station in the afternoon rush hour 16:00-19:00, which are the same predicates used in Figure 10 (d).

Based on the maps displayed from Figure 8 to Figure 11, the usage of SPET is summarized below. Initially, the user can select the traffic map to get an overview of the movement recorded, e.g., identifying hot-spot regions or routes. After that, a rectangle can be drawn to concentrate on patterns in the trajectories relevant with a specific area, e.g., departing from or arriving at the train station or airport. To further explore the patterns, user can switch to the dynamic offset map to get details of the movement information on specific routes filtered by pattern support or length. These patterns precisely reveal route choices together with their spatial and temporal variations, which can contribute to travel demand modeling and transportation planning.

Compared with other approaches such as heatmap or transparent trajectories (Willems *et al.* 2009), the advantages of SPET are described below. With backend of BP-CCSM, sequential patterns are visualized instead of raw trajectories, which is scalable to data size. From Figure 5(a) it can be observed that the pattern set size (10,000 to 100,000) is much smaller than the trajectory set size (639,000). In the meantime, noisy and infrequent trajectories can be filtered out. Quantitative information of patterns, which is generally not available in a map with transparent trajectories, is presented to the user in SPET. Equipped with the two types of maps, both overview and details of movement can be inspected and various visual exploration tasks can be accomplished.

6. Conclusions and Future Work

The paper proposed BP-CCSM, an efficient Bidirectional Pruning based Closed Contiguous Sequential pattern Mining algorithm. By employing three trees to create partitions of input sequences and candidate patterns, closeness can be checked by comparing nodes in a tree as opposed to performing a large number of sequence comparisons. A visual analytics system called Sequential Pattern Explorer for Trajectories was built with backend of BP-CCSM for pattern mining. Two types of maps were designed in SPET where a conventional traffic map provided an overview of a large number of patterns and a dynamic offset map displayed detailed information for user-filtered patterns. Experiments on a small public clickstream dataset and a large taxi GPS trajectory dataset demonstrate that BP-CCSM considerably outperforms three state-of-the-art similar mining algorithms in terms of running time and memory consumption. SPET supports efficient visual exploration of spatial and temporal variations in closed contiguous sequential patterns from a massive collection of trajectories.

Although this paper presented a case study using floating car data, the algorithm and visual analytics system can be readily applied to other type of GPS observations recorded from movement on a road network, e.g., bicycles and travelers. As for unconstrained movement, e.g., migration of birds, some insight could be gained by performing a different sequence transformation such as matching trajectories into a grid space.

In the future, we will generalize BP-CCSM to mine patterns from sequences containing repeated items. Both spatial and temporal variations in the patterns are currently explored by manually specifying different sets of predicates. Automatic detection of these variations will be investigated. The interactivity of SPET could also be improved by designing more advanced pattern filtering functions and the visual cluttering problem can be further addressed by employing 3D visualization approaches.

Acknowledgements

The authors would like to thank the three anonymous referees for their constructive comments that greatly improved the quality of this paper. The first author is supported by Chinese Scholarship Council. Acknowledgement is given to Clas Rydergren for access to the taxi GPS dataset, which was collected by the Info24 company within the Mobile Millennium Project. The authors are also grateful to Christian Borgelt and Jingsong Zhang for providing the source code of the algorithm Inc-CCFR and CCSpan respectively.

References

Abboud, Y., Boyer, A., and Brun, A., 2017. *In*: *CCPM: A Scalable and Noise-Resistant Closed Contiguous Sequential Patterns Mining Algorithm.*, 147–162 Cham: Springer International Publishing.

Afshar, X.Y.H.R., 2003. CloSpanMining Closed Sequential Patterns in Large Datasets. *SIAM Intl Conf.Data Mining*, 166–177.

Allström, A., *et al.*, 2011. Mobile Millennium Stockholm. *In*: *2nd International Conference on Models and Technologies for Intelligent Transportation Systems. Leuven, Belgium.*

Andrienko, G., Andrienko, N., and Wrobel, S., 2007. Visual analytics tools for analysis

of movement data. *ACM SIGKDD Explorations Newsletter (ACM Digital Library)*, 9 (2), 38–46.

Andrienko, N. and Andrienko, G., 2013. Visual Analytics of Movement: An Overview of Methods, Tools and Procedures. *Information Visualization*, 12 (1), 3–24.

Bachmann, A., Borgelt, C., and Gidófalvi, G., 2013. Incremental Frequent Route Based Trajectory Prediction. *In*: *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '13, Orlando, FL, USA New York, NY, USA: ACM, 49:49–49:54.

Cao, H., Mamoulis, N., and Cheung, D.W., 2005. Mining frequent spatio-temporal sequential patterns. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 82–89.

Chen, B.Y., *et al.*, 2014. Map-matching algorithm for large-scale low-frequency floating car data. *International Journal of Geographical Information Science*, 28 (1), 22–38.

Ferreira, N., *et al.*, 2013. Visual exploration of big spatio-temporal urban data: A study of New York city taxi trips. *IEEE Transactions on Visualization and Computer Graphics*, 19 (12), 2149–2158.

Fournier-Viger, P., *et al.*, 2014. *In*: *Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information.*, 40–52 Cham: Springer International Publishing.

Giannotti, F., *et al.*, 2011. Unveiling the Complexity of Human Mobility by Querying and Mining Massive Trajectory Data. *The VLDB Journal*, 20 (5), 695–719.

Giannotti, F., *et al.*, 2007. Trajectory pattern mining. *In*: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07* New York, New York, USA: ACM Press, p. 330.

Gomariz, A., *et al.*, 2013. *In*: *ClaSP: An Efficient Algorithm for Mining Frequent Closed Sequences.*, 50–61 Berlin, Heidelberg: Springer Berlin Heidelberg.

Guo, D., *et al.*, 2012. Discovering Spatial Patterns in Origin-Destination Mobility Data. *Transactions in GIS*, 16 (3), 411–429.

Guo, H., *et al.*, 2011. TripVista: Triple Perspective Visual Trajectory Analytics and its application on microscopic traffic data at a road intersection. *In*: *2011 IEEE Pacific Visualization Symposium*, Mar.. IEEE, 163–170.

Han, J., *et al.*, 2004. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Mining and Knowledge Discovery*, 8 (1), 53–87.

Kraak, M.j., 2003. The Space-Time Cube Revisited from a Geovisualization Perspective. *21st International Cartographic Conference (ICC)*, (August), 10–16.

Krumm, J., 2011. Trajectory Analysis for Driving. *Computing with Spatial Trajectories.* New York, NY: Springer New York, 213–241.

Li, B., *et al.*, 2011a. Hunting or waiting? Discovering passenger-finding strategies from a large-scale real-world taxi dataset. *In*: *2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, March., 63–68.

Li, C., *et al.*, 2012a. Efficient Mining of Gap-Constrained Subsequences and Its Various Applications. *ACM Trans. Knowl. Discov. Data*, 6 (1), 2:1–2:39.

Li, Q., Zhang, T., and Yu, Y., 2011b. Using Cloud Computing to Process Intensive Floating Car Data for Urban Traffic Surveillance. *Int. J. Geogr. Inf. Sci.*, 25 (8), 1303–1322.

Li, X., *et al.*, 2012b. Prediction of urban human mobility using large-scale taxi traces and its applications. *Frontiers of Computer Science in China*, 6 (1), 111–121.

Mabroukeh, N.R. and Ezeife, C.I., 2010. A Taxonomy of Sequential Pattern Mining Algorithms. *ACM Comput. Surv.*, 43 (1), 3:1–3:41.

Rahmani, M., Koutsopoulos, H.N., and Ranganathan, A., 2010. Requirements and potential of GPS-based floating car data for traffic management: Stockholm case study. *In*: *13th International IEEE Conference on Intelligent Transportation Systems*, Sept., 730–735.

Tominski, C., *et al.*, 2012. Stacking-Based Visualization of Trajectory Attribute Data. *IEEE Transactions on Visualization and Computer Graphics*, 18 (12), 2565–2574.

Wang, J. and Han, J., 2004. BIDE: efficient mining of frequent closed sequences. *Proceedings. 20th International Conference on Data Engineering*, 79–90.

Willems, N., van de Wetering, H., and van Wijk, J.J., 2009. Visualization of Vessel Movements. *In*: *Proceedings of the 11th Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'09, Berlin, Germany Chichester, UK: The Eurographs Association; John Wiley; Sons, Ltd., 959–966.

Ye, Y., *et al.*, 2009. Mining Individual Life Pattern Based on Location History. *In*: *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware* IEEE, 1–10.

Yildirimoglu, M. and Geroliminis, N., 2013. Experienced travel time prediction for congested freeways. *Transportation Research Part B: Methodological*, 53, 45 – 63.

Zhang, J., Wang, Y., and Yang, D., 2015. CCSpan: Mining closed contiguous sequential patterns. *Knowledge-Based Systems*, 89, 1–13.