Mining Frequent Spatio-temporal Sequential Patterns

Huiping Cao, Nikos Mamoulis, and David W. Cheung
Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
{*hpcao, nikos, dcheung*}*@cs.hku.hk*

Abstract

Many applications track the movement of mobile objects, which can be represented as sequences of timestamped locations. Given such a spatio-temporal series, we study the problem of discovering sequential patterns, which are routes frequently followed by the object. Sequential pattern mining algorithms for transaction data are not directly applicable for this setting. The challenges to address are (i) the fuzziness of locations in patterns, and (ii) the identification of non-explicit pattern instances. In this paper, we define pattern elements as spatial regions around frequent line segments. Our method first transforms the original sequence into a list of sequence segments, and detects frequent regions in a heuristic way. Then, we propose algorithms to find patterns by employing a newly proposed substring tree structure and improving Apriori technique. A performance evaluation demonstrates the effectiveness and efficiency of our approach.

1 Introduction

The movement of an object (i.e., trajectory) can be described by a sequence of spatial locations sampled at consecutive timestamps (e.g., with the use of Global Positioning System (GPS) devices). Parts of the object routes are often repeated in the archived history of locations. For instance, buses move along series of streets repeatedly, people go to and return from work following more or less the same routes, etc. The movement routes of most objects (e.g., private cars) are not predefined. Even for objects (e.g., buses) with pre-scheduled paths, the routes may not be repeated with same frequency due to different schedule in weekends or some special days. We are interested in finding frequently repeated paths, i.e., *spatio-temporal sequential patterns*, from a long spatio-temporal sequence. These patterns could help to analyze/predict the past/future movement of the object, support approximate query on the original data, and so on. However, they cannot be obtained straightforwardly by eliminating the noisy movement because of the *large* volume of the spatio-temporal data.

Discovery of sequential patterns from transactional databases has attracted lots of interest since Agrawal et al. introduced the problem [1]. In such a database, each transaction contains a set of items bought by some customer in one time, and a transaction sequence is a list of transactions ordered by time. For example, $\langle(a,b),(a,c),(b)\rangle$ is a sequence containing three transactions (a,b), (a,c) and (b). Given a collection of transaction sequences, the problem is to find ordered lists of itemsets appearing with high frequency. E.g., $\langle(b),(a),(b)\rangle$ is a pattern supported by the above sequence.

Unfortunately, pattern discovery techniques in transactional databases are not readily applicable for finding sequential patterns in spatio-temporal data. First, the elements in a transactional pattern are items that explicitly appear in pattern instances. On the other hand, location coordinates in a spatio-temporal series are real numbers, which do not repeat themselves *exactly* in every pattern instance. Second, the patterns are discovered from explicitly defined sets of sequences, like $\langle(a,b),(a,c),(b)\rangle$, in the previous example. Thus, a transaction list only contributes 0 or 1 to the support of a pattern, depending on whether the pattern appears or not in the specific sequence-set. In our setting, however, we detect frequent patterns from *one* long spatio-temporal sequence, without predefined segmentation of the data. The challenge is to identify the segments that contribute to a pattern, without allowing them to overlap with each other.

To summarize, the main contributions of this paper are: (i) We propose a model for spatio-temporal sequential patterns mining, based on appropriate definitions for pattern elements and pattern instances. (ii) We present an effective method for extracting pattern elements. (iii) We provide efficient pattern mining algorithms for discovering longer patterns. The remainder of the paper is organized as follows. Section 2 reviews the related literature. The formal definition of spatio-temporal sequential pattern is given in Section 3. Section 4 presents our solutions in detail. An ex-

perimental evaluation about the effectiveness and efficiency of our approach is presented in Section 5. Finally, Section 6 concludes this paper.

2 Related work

Our work is most related to pattern discovery from sequential data, which include time series, event sequences, and spatio-temporal trajectories.

Mannila et al. [10] investigated the discovery of frequent *episodes* from event sequences. An episodes is a (partially or totally) ordered list of events, thus is a variant of sequential pattern. A fixed sliding window w is used to extract segments (i.e., subsequences) in the event series, and the contribution of every segment to each candidate episode's frequency is counted. The segments supporting one episode may overlap, which is reasonable since episodes try to capture the appearing order of instantaneous events. However, this methodology may not get satisfactory results in finding spatio-temporal patterns, for several reasons. First, the window limits the length of the patterns. Second, pattern supports may not be counted correctly. E.g., the object's movement is $aabbcdefg$, where each character a, b, etc. corresponds to a spatial region. The occurrence of the pattern abc should be 1, since the object moves from a to c, once. However, if w is 5, pattern abc has support 4 due to the contribution of 4 segments (a_b_c, $\_ab\_c$, $a\_\_bc$, and $\_a\_bc$). Third, as opposed to well-defined categorical values for event instances, object locations do not repeat themselves exactly in pattern instances, for they are usually ordinal and inexact. Yang et al. investigated mining long sequential patterns in [13], also dealing with event series with noise.

Previous work on detecting patterns from time-series (e.g, [2, 7]) converted the problem to finding subsequences in lists of categorical data (e.g., event sequences), by preprocessing the original sequence to a string. A window w of fixed size is slided along the sequence, and a subsequence with length w is extracted for *every position*. In [2], the subsequences are clustered based on their shapes, and each cluster is given an id. In [7], some features are extracted from each subsequence (e.g., the slope of the best-fitting line of the sub-series, the mean of the signal, etc.). The feature space is divided into groups of similar values, and every subsequence is converted to a group-id. The raw sequence is then transformed to a string of cluster-ids or group-ids. The use of the window may over-count the patterns due to the reason explained above. In addition, since w is *fixed*, the extracted subsequences have the same length, which may affect the resultant patterns. Furthermore, for spatio-temporal data, even when we extract the subsequences using a sliding window and get simple features from these segments, we cannot directly group these features using methods in [2] and [7]. The cluster-based approach ([2]) has been discredited by [8]. The way to group the subsequence features ([7]) may be effective for time-series with 1-dimension values. For more complex spatio-temporal data, if we directly apply this method, i.e., split the features into groups, we may miss the information about the spatial proximity of segments, which is essential for grouping.

The first study on finding frequent sequential patterns from spatio-temporal data is [11]. The raw data here is not a long sequence, but lists of spatial locations. After discretizing the locations to *pre-defined* spatial decomposition, the process is intrinsically similar to that in transactional databases.

[9] addresses the problem of discovering *periodic* patterns in spatio-temporal data, which is a generalization of mining periodic patterns in event sequences. Given a *period* T, in the case of spatio-temporal data, a periodic pattern is a (not necessarily contiguous) sequence of spatial regions, which appears frequently every T timestamps and describes the object movement (e.g., a bus moves from district a to district b and then to c with high probability, every three hours). The contribution of [9] is that it does not treat spatio-temporal series as event sequences, by merely replacing each location by a predefined region enclosing it, but automatically discovers the regions that form the patterns. This method, although effective for its purpose, relies on a fixed T (i.e., the patterns repeat themselves every regular time periods). In addition, it is prone to distortions/shiftings of the pattern instances, i.e., periodic segments where the pattern does not appear in the same positions as in the pattern definition do not contribute to the pattern's support.

3 Spatio-temporal sequential patterns

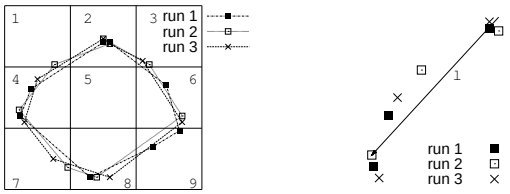
A **spatio-temporal sequence** S is a list of locations, (x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n), where t_i represents the timestamp of location (x_i, y_i) $(1 \leq i \leq n)$. Figure 1 illustrates the movement of an object which repeats a similar route in three runs. We are interested in movement patterns repeated frequently in such a series. This section first motivates our solution, then formally defines the problem.

3.1 Motivation

Locations are not repeated *exactly* in every instance of a movement pattern. Our idea is to summarize a series of spatial locations to that of spatial regions.

A naive method is to use a regular grid (or some predefined spatial decomposition) to divide the space into regions by taking a user-defined parameter G,an approximate number that each axis will be split to. Then, the locations series can become a sequence of grid-ids utilizing a transformation approach. The first method, Grid I, converts each location to the id of the cell it falls in. E.g., the raw series in Figure 1a, can be transformed to the cell-id sequence $c_2c_4c_8c_9c_6c_2 \ldots c_3$. Although intuitive, this method has two problems. First, we lose the information on how the object moves inside a cell, if the space decomposition is

coarse. The patterns may not be very descriptive. Second, for two instances of a pattern, the locations may not fall into the same cell (i.e., two adjacent locations appear in neighboring cells). We may miss some frequent patterns, whose instances are divided between different grid-based patterns. The first problem could be alleviated by decreasing G, however, this would increase the chances of missing patterns due to the second problem. An alternative conversion technique adds the ids of cells that intersect with the line segments connecting consecutive locations to the transformed sequence. In the example of Figure 1a, Grid II converts the sequence for the first run to $c_2c_1c_4c_7c_8c_9c_6c_3c_2$. Nevertheless, by this improvement, the new series may be significantly longer than the original one, which may already be extremely long, like spatio-temporal sequences usually are.



(a) (b)

Figure 1. Object Movement

Thus, we need a better way to abstract the trajectory. Motivated by line simplification techniques ([3]), we represent segments of the spatio-temporal series by directed line segments. Figure 1b shows that the line segment l summarizes the first three points in each of the three runs with little error. In this way, not only do we compress the original data, decreasing the mining effort, but also the derived line segments (which approximately describe movement) provide initial seeds for defining the spatial regions, which could be expanded later by merging similar and close segments.

3.2 Problem definition

A **segment** s_{ij} in a spatio-temporal sequence S $(1 \leq i < j \leq n)$ is a contiguous subsequence of S, starting from (x_i, y_i, t_i) and ending at (x_j, y_j, t_j). Given s_{ij}, we define its **representative line segment** $\vec{l_{ij}}$ with starting point (x_i, y_i) and ending point (x_j, y_j). Let ϵ be a distance error threshold, s_{ij} **complies with** $\vec{l_{ij}}$ with respect to ϵ and is denoted as $s_{ij} \propto_\epsilon \vec{l_{ij}}$, if dist$((x_k, y_k), \vec{l_{ij}}) \leq \epsilon$ for all $k (i \leq k \leq j)$, where dist$((x_k, y_k), \vec{l})$ is the distance between (x_k, y_k) and line segment $\vec{l}$. When $s_{ij} \propto_\epsilon \vec{l_{ij}}$, each point $(x_k, y_k), i \leq k \leq j$, in s_{ij} can be projected to a point (x'_k, y'_k) on $\vec{l_{ij}}$. (x'_k, y'_k) implicitly denotes the projection of (x_k, y_k) to $\vec{l_{ij}}$. Figure 2a illustrates a segment s_{ij} complying with $\vec{l_{ij}}$ and shows the projection (x'_k, y'_k) of point (x_k, y_k) on $\vec{l_{ij}}$. A **segmental decomposition** S^s of S is defined by a list of consecutive segments that constitute S. Formally, $S^s = s_{k_0k_1}s_{k_1k_2} \dots s_{k_{m-1}k_m}$, $k_0 = 1, k_m = n, m < n$, where $s_{k_ik_{i+1}} \propto_\epsilon \vec{l_{k_ik_{i+1}}}$ for all i, To simplify notation, we use $s_0s_1 \dots s_{m-1}$ to denote S^s.

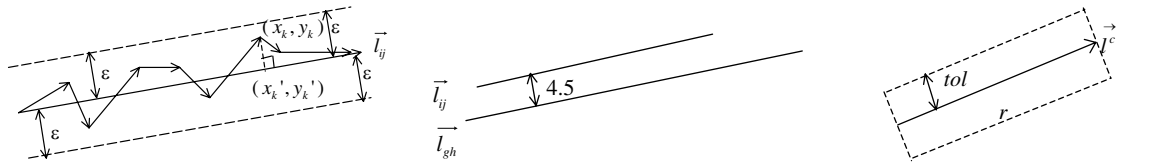
Let $\vec{l}$ represent a directed line segment, $\vec{l}.angle$ and $\vec{l}.len$ be its slope angle and length respectively. Two line segments $\vec{l_{ij}}$ and $\vec{l_{gh}}$ representing segments s_{ij} and s_{gh} are **similar**, denoted by $\vec{l_{ij}} \sim \vec{l_{gh}}$, with respect to angle difference threshold θ and length factor f $(0 \leq f \leq 1)$ if:
(i) $|\vec{l_{ij}}.angle - \vec{l_{gh}}.angle| \leq \theta$ and
(ii) $|\vec{l_{ij}}.len - \vec{l_{gh}}.len| \leq f \times max(\vec{l_{ij}}.len, \vec{l_{gh}}.len)$ If $\vec{l_{ij}} \sim \vec{l_{gh}}$, s_{ij} and s_{gh} are also treated as similar to each other. Note that similarity is symmetric. The location information of segments is not considered in defining similarity, since we use it when defining the segments' closeness.

Line segment $\vec{l_{ij}}$ is **close** to $\vec{l_{gh}}$ if for $\forall (x'_k, y'_k) \in \vec{l_{ij}}$, dist$((x'_k, y'_k), \vec{l_{gh}}) \leq \epsilon$. When $\vec{l_{ij}}$ is close to $\vec{l_{gh}}$, we also say that the segment s_{ij} is close to the segment s_{gh}, where $s_{ij} \propto_\epsilon \vec{l_{ij}}$ and $s_{gh} \propto_\epsilon \vec{l_{gh}}$. As opposed to similarity, closeness is asymmetric. Figure 2b shows an example. Let $\vec{l_{ij}}$ is parallel to $\vec{l_{gh}}$ and $\epsilon = 5.0$. The distance between these two parallel line segments is 4.5. Observe that $\vec{l_{ij}}$ is close to $\vec{l_{gh}}$ because the distance from each point in $\vec{l_{ij}}$ to $\vec{l_{gh}}$ is less than 5.0. However, $\vec{l_{gh}}$ is not close to $\vec{l_{ij}}$ for the point in the right upper part has distance to $\vec{l_{ij}}$ bigger than 5.0.

Let L be a set of segments from sequence S^s. The **mean line segment** for L, $\vec{l^c}$, is a line segment that best fits all the points in L with the minimum sum of squared errors (SSE). In other words, if $PSet$ contains all the points of the segments in L, the mean line segment $\vec{l^c}$ is such that $\sum_{p \in PSet} dist(p, \vec{l^c}) \leq \sum_{p \in PSet} dist(p, \vec{l})\ \forall \vec{l} \neq \vec{l^c}$.

Let tol be the average orthogonal distance of all the points in L to $\vec{l^c}$. A spatial pattern element is a rectangular spatial region r_L with four sides determined by $(\vec{l^c}, tol)$ as following: (1) two sides of r's that are parallel to $\vec{l^c}$, have the same length as $\vec{l^c}$, and their distances to $\vec{l^c}$ are tol; (2) the other two vertical sides have length $2 \cdot tol$, and their midpoints are the two end points of $\vec{l^c}$. We refer to $\vec{l^c}$ as the **central line segment** of region r_L. We say that region r_L contains k segments or k segments contribute to r_L if L consists of k segments. Figure 2c visualizes this definition. A **spatio-temporal sequential pattern** P is an ordered sequence of pattern elements: $r_1r_2 \dots r_q$, $(1 \leq q \leq m)$. The **length** of pattern P is the number of regions in it.

A contiguous subsequence of S^s, $s_is_{i+1} \dots s_{i+q-1}$, is a **pattern instance** for P: $r_1r_2 \dots r_q$ if $\forall j (1 \leq j \leq q)$, if the representative line segment for segment s_{i+j-1} is *similar* and *close* to the central line segment of region r_j. A pattern's instances cannot overlap in time (the pattern may be over-counted like that in [10] otherwise), i.e., if two con-



(a) Segment complies with $\vec{l_{ij}}$ (b) Example for closeness (c) Region r determined by $(\vec{l^c}, tol)$

Figure 2. Example of definitions

tinuous subsequences of S^s, $s_i \dots s_j$ and $s_g \dots s_h$, are two instances for pattern P, either $j < g$ or $h < i$. Given patterns P': $r'_1 r'_2 \dots r'_i$ and P: $r_1 r_2 \dots r_j$, P' is a **subpattern** of P if $i \leq j$ and $\exists k, (1 \leq k \leq j-i+1)$ such that $r'_1 = r_k$, $r'_2 = r_{k+1}$, …, $r'_i = r_{k+i-1}$. P is a **superpattern** of P'.

The **support** of a pattern P is the number of instances supporting P. Given a support threshold min_sup, P is **frequent** if its support exceeds min_sup. Since a pattern with same frequency to one of its supersets is redundant, we focus on detecting *closed* frequent patterns [4], for which every proper subpattern has equal frequency. The mining **problem** is to find frequent patterns from a long spatio-temporal sequence S with respect to a support threshold min_sup, and subject to a segmenting distance error threshold ϵ, a similarity parameter θ and a length factor f. The parameter values depend on the application domain, or can be tuned as part of the mining process [2]. In using the raw data to discover patterns, we discuss how to set the parameters in Section 5.1 more applicably.

4 Solution

In this section, we describe how to discover frequent singular patterns, i.e., frequent spatial regions (Section 4.1) and longer closed patterns (Section 4.2).

4.1 Discovering frequent singular patterns

The segmentation (line simplification) algorithm ([3, 5, 6]) is used to convert the locations series to segments sequences so that each raw sequence segment could be abstracted by a line segment. Our idea is to transform S to S^s using such a technique, and take the segments obtained as seed for the desired spatial regions, whose central line segments best fit the points of segments in the regions. The DP (Douglas-Peucker) algorithm [3] is a classical top down approach for this problem. [6] provides an online algorithm in splitting a sequence to segments with quite good quality. Since it is important to keep the internal movement inside a region, we need to capture the sharp turn of the movement in the transformation. We employ DP method because it has been proved to be the best algorithm in choosing splitting points [12]. In brief, DP algorithm recursively decomposes S: $\{p_1, \dots, p_n\}$ to a series of line segments $l_1, \dots l_m$, $m \leq n$, each of which, l_i, simplifies a subsequence S_{l_i}, such that the perpendicular distance from every point in S_{l_i} to l_i is at most ϵ. For efficiency purpose, DP's improved version ([5]) could be adopted.

Discovering *frequent singular patterns* from S^s is a hard problem, since in the worst-case, all combinations of segments in S^s have to be considered as candidate. To expedite the process, we employ a heuristic, *Growing*. Let $Segs$ be a set initially containing all the segments in S^s. *Growing* works as follows. It selects the segment s with median length, i.e., the median of the lengths of the segments in $Segs$, as seed for the initial spatial region r. Then, r is grown by merging other segments in $Segs$ through *filtering* and *verification* steps, described later. Next, for the set of remaining segments not merged to r, the segment s' with median length in it is selected as seed for growing. Finally, the overall algorithm terminates after all segments (i) have been assigned to a region (as initial seeds or to the region of another seed), or (ii) have been found not to belong to any frequent region and marked as outliers. Selecting the segment with median length as seed could help to absorb short segments with less error, compared to taking segment with longer length as seed. Meanwhile, it could prevent generating regions with too fine granularity, which could happen when shorter length segment is used as seed. *Growing* is deterministic in using this seed selection procedure.

The filtering process checks two conditions. First, for each s_i in $Segs$ the angle difference $diff\_a_i$ between $\vec{l_s}$ and s_i is computed, and s_i is treated as *candidate* if $diff\_a_i$ is less than θ. All the candidate segments are put into a set C. Second, the minimum distance from every segment in C to $\vec{l_s}$ is computed and all segments whose minimum distances to $\vec{l_s}$ is larger than $f \cdot \vec{l_s}.len$ are pruned. The remaining segments in C will be used for verification.

The filtering step computes the minimum distance between segments, but it does not consider the length difference (second condition of similarity), between each $\vec{l_{s_i}} \in C$ and $\vec{l_s}$, and the exact spatial distances of segments in C to $\vec{l_s}$ (closeness condition). In the verification step, Algorithm 1 (shown below) merges the segments in C to the spatial region r around $\vec{l_s}$, if $s_i \in C$ satisfies the closeness and length difference condition. Otherwise, we extract from s_i the part that satisfies the condition, and merge this part with r. The remaining part of s_i is a new segment and inserted back to

$Segs$ (Line 15) for later processing.

Algorithm 1 Verification($\vec{l_s}$, C, $Segs$, f, min_sup)

```
1: α := l_s.len × f; m:=0;
2: //length check
3: for each segment s_i in C do
4:     intersect s_i with l_s, get s' and l_s';
5:     if (diff(l_s.len, l_s'.len) ≤ α) m++;
6: end for
7: //closeness check
8: while (m ≥ min_sup) do
9:     Get l_c from all intersected points for region r;
10:    Validate all intersected parts from C;
11:    if (all intersected parts are close to l_c) break;
12: end while
13: if (m<min_sup) return;
14: for each segment s_i in C do
15:    Add non-intersected part of s_i to Segs;
16:    Remove s_i from Segs;
17: end for
18: Remove segment that l_s represents from Segs;
```

We explain how we compute the intersected part of s_i and $\vec{l_s}$ in Line 4. Let $\vec{l_{s_i}}$ be the representative line segment for s_i. If all projection points (x'_k, y'_k) in $\vec{l_{s_i}}$ have distance to $\vec{l_s}$ no more than α (Line 1), its related location point (x_k, y_k) in the segment is put into the intersected part s'. The line segment created by mapping each point in s' to $\vec{l_{s_i}}$ is denoted as $\vec{l_{s'}}$. For example, let s_i represent segment (x_{10}, y_{10}, t_{10}), ..., (x_{30}, y_{30}, t_{30}). Assume that the distances from points in $\vec{l_{s_i}}$ to $\vec{l_s}$ are all smaller than α except points from (x'_{10}, y'_{10}) to (x'_{15}, y'_{15}). Then, s' is segment (x_{16}, y_{16}, t_{16}), ..., (x_{30}, y_{30}, t_{30}), and $\vec{l_{s'}}$ represents line segment from (x'_{16}, y'_{16}) to (x'_{30}, y'_{30}) in $\vec{l_{s_i}}$.

4.2 Deriving longer patterns

After finding frequently visited spatial regions, original data S is converted to a series S^R of spatial regions by changing the segments in frequent regions to region ids, and those not in any region to outliers. S^R preserves the motion continuity of the object by showing how it moves among regions. Although each region in S^R is repeated frequently, the concatenation of some regions may not be frequent. E.g., a person living in r_1 often goes to a place r_2 in some days and to region r_3 in other days. r_1, r_2 and r_3 are frequently visited, but the path r_2r_3 is not frequent. This section discusses how to detect the longer frequent patterns.

4.2.1 Level-wise mining

A direct way is to perform level-wise pattern mining. However, this approach suffers from the disadvantage that S^R needs to be scanned many times. We propose solutions to reduce the number of candidates and scans in probing long candidates, based on the following properties we observe.

Property 1 (Connectivity Constraint): *Due to continuity of object movement, a spatial region can only connect to some but not all the others in S^R.* This constraint can help reduce the number of generated candidates, as follows. We first construct a connectivity graph for all the spatial regions in S^R. A directed edge from r_i to r_j is added to the graph if the substring r_ir_j appears in the sequence. The edge weight is the frequency that r_ir_j appears in the sequence. Let $r_1r_2 \ldots r_k$ be a frequent pattern, and r_k only points to r_i and r_j, only two candidates, $r_1r_2 \ldots r_kr_i$ and $r_1r_2 \ldots r_kr_j$ are generated. Further, if the edge weight from r_k to some element, say r_i, is no more than min_sup, we need not generate candidate $r_1r_2 \ldots r_kr_i$.

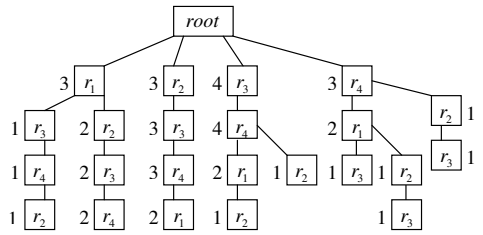
Property 2 (Closeness Property): *Given a pattern P, suppose its last element connects to r_1, r_1 connects to r_2, ..., r_{m-1} connects to r_m, ($m \geq 2$). We can get pattern $P_1 = Pr_1$ (concatenating P and r_1), $P_2 = Pr_1r_2$, ..., $P_m = Pr_1r_2 \ldots r_m$. Obviously, if P_1 and P_m have the same support, any P_i, ($1 < i < m$) also has the same support.* This property helps to generate candidates more efficiently. Let *result* be the frequent patterns at the end of the kth scan and P be a pattern in it with last element r. We can extend P using other patterns in *result* that start with r. For instance, let $P = r_1r_2r_3$, and r_3 only connect to r_4 in the connectivity graph. In addition, assume that *result* contains only one pattern starting from r_3: $P' = r_3r_4r_6r_7$. P can then be extended to candidates $r_1r_2r_3r_4$ (using Property 1), and $r_1r_2r_3r_4r_6r_7$ (using Property 2). If $r_1r_2r_3r_4$ and $r_1r_2r_3r_4r_6r_7$ have the same support after the counting, we only need to consider candidates longer than $r_1r_2r_3r_4r_6r_7$ later, significantly reducing the number of scans.

4.2.2 Mining using the substring tree

We propose a *substring tree* structure to facilitate counting of long substrings with different elements. The substring tree is a rooted directed tree whose root links to multiple substring sub-trees. Each node in a sub-tree consists of pattern element and a counter, which counts the number of substrings (i.e., subsequences of elements) that contribute to the pattern formed by the path from the root to this node. A substring tree example is shown in Figure 3a.

To construct the tree, in scanning S^R, we extract substrings containing distinct elements, and insert them to the tree. In seeing an element r in S^R, we concatenate it to the substrings found so far that do not contain r. Also, if no substring starting with r is found, r is treated as a new substring. We give an example to illustrate the extraction of substrings. Let S^R be $r_1r_2r_3r_4r_1r_3r_4r_2r_3r_4r_1r_2r_3r_4$. Initially, no substring is extracted. When see the first r_1, we create a new substring for it. On seeing the second element r_2, we create a new substring r_2 since no substring starting with r_2 exists. In addition, we concatenate it to the only substring r_1 and get r_1r_2. The process continues until we see the fifth element r_1. There is already a string $r_1r_2r_3r_4$ with r_1 as first element, so $r_1r_2r_3r_4$ is inserted to the tree,

and a new substring starting from r_1 is created. Figure 3a shows the full substring tree for sequence S^R.



(a) Substring tree example

stack	result		stack	result
$\underline{r_3}(4)$			$\underline{r_3r_4}(4)$	
$r_1(3)$			$r_1(3)$	
$r_2(3)$		$\Rightarrow$	$r_2(3)$	
$r_4(3)$			$r_4(3)$	$r_3(4)$
$\underline{r_1}(3)$			$\underline{r_2}(3)$	
$r_2(3)$			$r_4(3)$	
$\Rightarrow$ $r_4(3)$		$\Rightarrow$	$r_3r_4r_1(2)$	$r_1(3)$
$r_3r_4r_1(2)$	$r_3r_4(4)$		$r_1r_2r_3r_4(2)$	$r_3r_4(4)$

(b) Mining patterns from the substring tree

Figure 3. Mining using substring tree

For deriving frequent patterns from the substring tree, we utilize a stack. Each element in the stack comprises of a *pattern*, its *count* and a *level*, indicating whether the pattern has reached a leaf or not. First, we add to the stack the patterns associated with the root's children. Then, we iteratively pop patterns with highest frequency from the stack. If the popped up pattern is not at leaf level and is frequent, we output it, and extend it by concatenating it with its children's elements and push the extended patterns to the stack; otherwise, the pattern is just output (if frequent). In the above example, there are initially four elements in the stack. Figure 3b shows the first several steps for the mining process. Let $min\_sup = 2$. When popping $r_3(4)$ from the initial stack, we output it as result, and extend it to r_3r_4 since it is not at the leaf level. Next, we pop up $r_3r_4(4)$ and delete $r_3(4)$ from the result because its frequency is the same to that of r_3r_4 (definition of closed patterns). This process continues until no pattern exists in the stack. The final closed patterns are $r_3r_4(4)$, $r_1(3)$, $r_2r_3r_4(3)$, $r_4r_1(2)$, $r_3r_4r_1(2)$, $r_1r_2r_3r_4(2)$, $r_2r_3r_4r_1(2)$. The patterns discovered from the substring tree are not the final results because they only contain patterns with distinct elements. We extend the patterns using the level-wise method. The result may contain overlapping patterns like $r_1r_2r_3r_4(2)$ and $r_2r_3r_4r_1(2)$. We report all of them though the pattern space may be large. The reason is that if we output only one of them, say $r_1r_2r_3r_4(2)$, the information that r_4 connects to r_1 (necessary for generating longer patterns) will be missed. Finally, our algorithm outputs frequent *closed* patterns.

5 Experiments

This section evaluates our proposed approach with real and synthetic data. After discussing the way to set the parameters in Section 5.1, we study the effectivesness and efficiency in Section 5.2.

Real datasets: The real data contain tracked bus movements in Patras, Greece. Each sequence is the movement of a bus in a single day. The coordinates in the sequence are in meters following the EGSA84 projection (A Greek coordinate system). Bus locations were sampled every 30 seconds. However, since a vehicle might stop intermittently and the GPS is switched off during that period, the movement in a sequence may not be straightly continuous. The series length varies in the range between 1000 to 7000.

Synthetic data: We also generated long sequences to facilitate the performance study. The generator takes three parameters, $|p|$, n, and m. $|p|$ is the number of line segments constituting circular paths (i.e., patterns) of the movement. n denotes the sequence length. And m represents the number of times that the object repeats the patterns. Obviously, $n > |p| \times m$. The generator first creates circular routes with $|p|$ connected line segments. Then, it generates locations along the routes to simulate the object movement. The actual number of positions for each run is produced by adding/reducing some random values to/from $\lfloor \frac{n}{m} \rfloor$. In every run, the locations for each line segment are approximately the same. The description of the artificial series is given in related experiments.

5.1 Setting the parameters

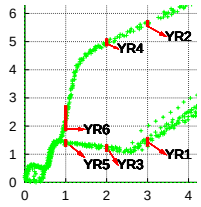
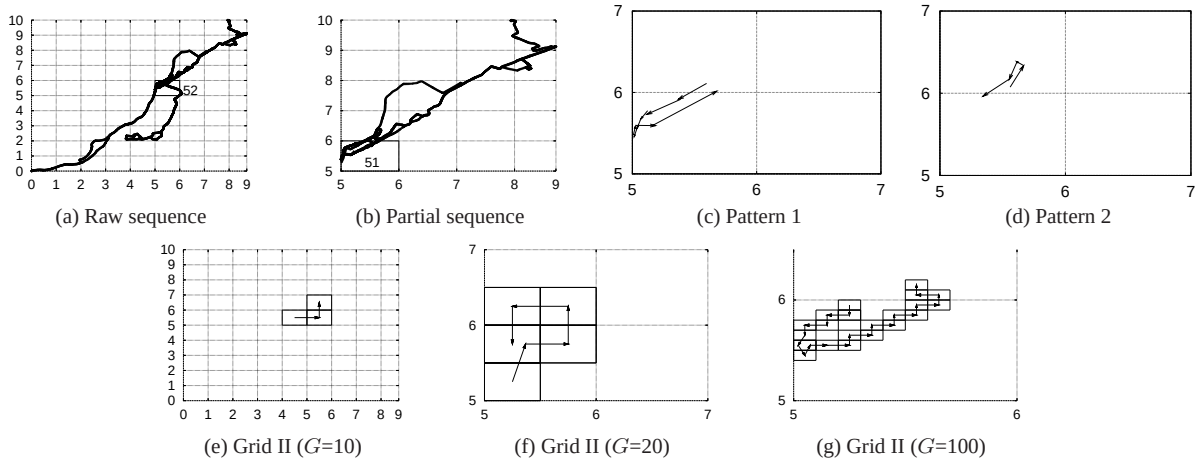


Figure 4. Parameter estimation example

We employ a heuristic based on sampling, to determine the value of parameter ϵ. We choose a random sample from the dataset and keep only the locations, for which the x coordinates are very close to a set of x values, say x_1, x_2, x_3. For each value in the set, we cluster the y coordinates of the sample points and derive dense ranges of y values. For instance, in Figure 4 for x values $x_1 = 1, x_2 = 2, x_3 = 3$, we can identify 6 dense ranges $YR1, YR2, \ldots, YR6$ — denoted by the bold (red in color mode) vertical short line segments.



(a) Raw sequence (b) Partial sequence (c) Pattern 1 (d) Pattern 2

(e) Grid II (G=10) (f) Grid II (G=20) (g) Grid II (G=100)

Figure 5. Raw sequence and patterns discovered

We define ϵ_y as the average length of these y values. Similarly, ϵ_x can be obtained. Finally, we set $\epsilon = \min\{\epsilon_x, \epsilon_y\}$ as smaller ϵ will allow pattern definition at a finer granularity. Experimentally, we found that, for most datasets, by setting ϵ to the estimated value (even vary a little), θ to around 0.3 radians, and f to around 0.2 (20% rule), our algorithm retrieves hidden patterns in the data, i.e., pre-scheduled paths for bus data and patterns generated for synthetic data.

5.2 Effectiveness and efficiency study

We examine the effectiveness of our method taking as input a raw bus movement sequence shown in Figure 5a, which contains 6921 locations. This movement exhibits partial regularity and consists of noise.

For visualization purpose, we show its interesting part in more detail in Figure 5b because the remainder contains noise segments appearing only once. According to the description in Section 5.1, we tune the parameters to $\epsilon = 20$ (map size is 100×100), $f = 0.2$, $\theta = 0.3$ radian, and $min\_sup = 3$. In this movement, the frequently repeated paths are around cell c_{51}. Figure 5c and 5d show the two *longest closed* patterns discovered by our method. For simplicity, only the central line segments for the regions in the patterns are plotted. The arrow of each central line segment shows the movement direction inside that region and the connection of these directed line segments illustrates the movement from one region to another. They are not connected because of the noise movement near the boundary of grid 51 (see Figure 5b). We also plot the results discovered by Grid II since it is more effective than Grid I. When G is 10, the pattern discovered near cell 51 is $c_{50}c_{51}c_{60}$ in Figure 5e (movement from the region of cell c_{50} to cell c_{51} then to cell c_{60}). This is quite coarse, since the movement inside each cell is unknown. The *longest closed* patterns for G=20 and G=100 are shown in Figure 5f and Figure 5g. They improve on accuracy with the increase of G, however, the patterns in the cell above c_{51} (related to pattern in Figure 5d) is still missed. Furthermore, the mining efficiency degrades significantly. Our approach takes about 200ms, while Grid II with $G = 100$ takes about 450ms, which is more than double. In summary, the results show that our method can find hidden sequential patterns effectively. Given proper G, Grid II can also discover coarse movement patterns. However, it suffers from two disadvantages (i) the internal movement in a grid cell cannot be found; (ii) it is less efficient than our method in finding patterns of similar quality.

We used synthetic data to evaluate the *efficiency*. We first analyze the performance of finding frequent singular patterns. The parameters of the data generator were set to $|p| = 20$, $n = 30K$, and $m = 50$ in a map of size=1×1. We set the mining parameters $\epsilon = 0.01$, $f = 0.2$ and $\theta = 0.3$, and vary min_sup. The performance is shown in Table 1a. Num_{P_1} is the number of frequent singular patterns and S^R_{len} is the length of S^R. We observe that the time rises only when the increase of min_sup brings the decrease of Num_{P_1}. It is because the *Growing* method inspects more seeds before it finds satisfactory spatial regions when the resultant Num_{P_1} is smaller. In the worst case, every segment in $Segs$ need to be examined.

Table 1b compares the total time spent by our methods, and the grid methods which use the substring tree for finding longer patterns. The generating parameters are $|p| = 100$, $m = 50$, and $n = 500K$. The substring tree technique slightly outperforms the level-wise method in all cases since it uses most time (about 12s) to find singular frequent pattern and most patterns contain long subpatterns with distinct elements. Their time is nearly constant to min_sup because S^R_{len} is the same for different min_sup. The grid methods

min_sup	Num_{P_1}	S^R_{len}	time (ms)
≤ 82	18	1576	560
83,84	11	1268	600
85	3	423	1130
≥ 86	2	339	1190

(a) Time for discovering singular patterns vs. min_sup

Method	time (s) for different min_sup		
	50	60	70
Level-wise	17.35	17.33	17.32
Substring tree	13.47	13.49	13.49
Grid I ($G = 10$)	42.58	42.56	16.11
Grid II ($G = 10$)	30.52	30.56	30.51
Grid I ($G = 20$)	57.38	33.86	22.00
Grid II ($G = 20$)	345.35	58.56	45.54

(b) Total time vs. various min_sup

Table 1. Efficiency comparison

take longer time, since the transformed cell-ids sequence is much longer (i.e., n or higher) than that of S^R. When we increase G from 10 to 20, the time increases sharply, since the number of cells quadruplicates and the sequence becomes much longer. Sometimes, Grid II may take less time than Grid I (e.g., for G=10 and min_sup=50 and 60). This happens because many cells in the sequence become outliers for this case, thus Grid II discovers shorter patterns (whereas Grid I finds longer ones, since it does not introduce intermediate cells at a sharp movement).

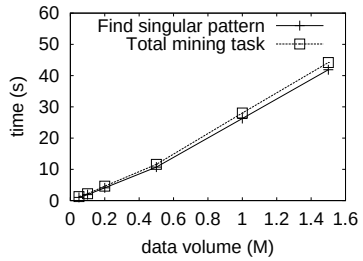


Figure 6. Scalability

Figure 6 tests the scalability of our method in using the substring tree. We generate the datasets, keeping $|p|$ constant (50) and changing n (the number of spatial locations in $\mathcal{S}$) from 50K to 1.5M. The total cost is nearly linear to n, although it includes the cost for sorting the segments lengths and computing angle differences, which is about $O(mlogm)$ where m ($m \ll n$) is the number of segments.

6 Conclusion

In this paper, we modeled the problem of mining sequential patterns from spatio-temporal data by considering both spatial and temporal information. Singular frequent patterns are found effectively, by grouping segments not only by similar shape (like previous work in time-series mining), but also by closeness in space. In addition, we employed special properties of the problem (spatial connectivity, closeness) and a newly proposed substring tree to accelerate search for longer patterns.

Acknowledgements

The authors would like to thank Dieter Pfoser and CTI for providing us with the real bus dataset. Unfortunately, the data are copyrighted and cannot be made publicly available. The work was supported by grant HKU 7142/04E from Hong Kong RGC.

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of Intl. Conf. on Data Engineering*, pages 3–14, 1995.

[2] G. Das, K. I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Proc. of Intl. Conf. on Knowledge Discovery and Data Mining*, pages 16–22, 1998.

[3] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. In *The Canadian Cartographer, Vol.10, No.2*, pages 112–122, 1973.

[4] J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining top-k frequent closed patterns without minimum support. In *Proc. of Intl. Conf. on Data Mining*, pages 211–218, 2002.

[5] J. Hershberger and J. Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. In *Proc. of the 5th Intl. Symposium on Spatial Data Handling(SDH)*, pages 134–143, 1992.

[6] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Proc. of Intl. Conf. on Data Mining*, pages 289–296, 2001.

[7] E. Keogh, S. Lonardi, and B. Chiu. Finding surprising patterns in a time series database in linear time and space. In *Proc. of ACM Knowledge Discovery and Data Mining*, pages 550–556, 2002.

[8] J. Lin, E. Keogh, and W. Truppel. Clustering of streaming time series is meaningless. In *Proc. of the SIGMOD workshop in Data Mining and Knowledge Discovery*, pages 56–65, 2003.

[9] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *Proc. of Intl. Conf. on Knowledge Discovery and Data Mining*, pages 236–245, 2004.

[10] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. In *Data Mining and Knowledge Discovery, Vol. 1*, pages 259–287, 1997.

[11] I. Tsoukatos and D. Gunopulos. Efficient mining of spatiotemporal patterns. In *Proc. of Intl. Symp. on Spatial and Temporal Databases*, pages 425–442, 2001.

[12] E. R. White. Assessment of line-generalization algorithms using characteristic points. *The American Cartographer*, 12(1):17–27, 1985.

[13] J. Yang, W. Wang, P. S. Yu, and J. Han. Mining long sequential patterns in a noisy environment. In *Proc. of SIGMOD conf.*, pages 406–417, 2002.