Efficient Mining of Regional Movement Patterns in Semantic Trajectories

Dong-Wan Choi[†] Jian Pei[‡] Thomas Heinis[†]

[†]Imperial College London, London, UK
[‡]Simon Fraser University, Burnaby, Canada
[†]{d.choi, t.heinis}@imperial.ac.uk, [‡]jpei@cs.sfu.ca

ABSTRACT

Semantic trajectory pattern mining is becoming more and more important with the rapidly growing volumes of semantically rich trajectory data. Extracting sequential patterns in semantic trajectories plays a key role in understanding semantic behaviour of human movement, which can widely be used in many applications such as location-based advertising, road capacity optimisation, and urban planning. However, most of existing works on semantic trajectory pattern mining focus on the entire spatial area, leading to missing some locally significant patterns within a region. Based on this motivation, this paper studies a *regional semantic trajectory pattern mining* problem, aiming at identifying all the regional sequential patterns in semantic trajectories. Specifically, we propose a new density scheme to quantify the frequency of a particular pattern in space, and thereby formulate a new mining problem of finding all the regions in which such a pattern densely occurs. For the proposed problem, we develop an efficient mining algorithm, called *RegMiner* (Regional Semantic Trajectory Pattern Miner), which effectively reveals movement patterns that are locally frequent in such a region but not necessarily dominant in the entire space. Our empirical study using real trajectory data shows that *RegMiner* finds many interesting local patterns that are hard to find by a state-of-the-art global pattern mining scheme, and it also runs *several orders of magnitude* faster than the global pattern mining algorithm.

1. INTRODUCTION

People may have different movement patterns in different regions. For example, people in a downtown area may have a movement pattern from work places (e.g., office) to entertainment spots (e.g., pub), but a different pattern, e.g., from attractions (e.g., museum) to accommodations (e.g., hotel) will appear in a touristic area. Understanding this regional human movement behaviour can help us improve the quality of location-based services, focus on specific regions for advertising, and help determining where to place a new service facility.

Consider one is trying to find the best region to open a new coffee shop. One option could be to choose a region with few competitors (i.e., other coffee shops), or they can select a region with many customer places (e.g., offices). The optimal region, however, will be found by answering the following question:

"which region is expected to have a maximum number of customer visits at some coffee shop?"

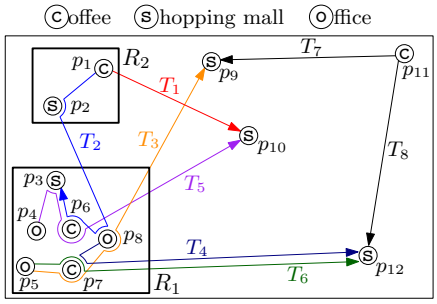
Nowadays, we can answer this kind of question thanks to the prevalence of trajectory data enriched with semantic information, called *semantic trajectory* [2], where each point of a trajectory not only represents a particular location but also has a semantic category such as coffee shop. Semantic trajectory data has become highly available with the increasing volume of geo-tagged data from many location-sharing services such as Foursquare and Facebook Places.

Several approaches to mine sequential patterns in semantic trajectories [2, 28, 29, 14] were recently proposed. They commonly deal with the entire spatial region as a target area to find globally frequent patterns by following the conventional sequential pattern mining model [1]. Unfortunately, these schemes cannot immediately answer the aforementioned question as they very likely ignore local patterns frequently occurring within a relatively small region.

Motivated by this challenge, we study the new mining problem of finding all *regional movement patterns* in semantic trajectories. Figure 1 demonstrates the aim of our problem that differs from those of existing mining schemes. For example, the pattern $\langle \mathsf{coffee} \rightarrow \mathsf{shopping\ mall} \rangle$ is the most frequent movement pattern in the entire space. This, however, does not implies that the pattern is locally frequent in any particular region of the area. In region R_1, for instance, $\langle \mathsf{coffee} \rightarrow \mathsf{shopping\ mall} \rangle$ occurs only once at $\langle p_6, p_3 \rangle$. Indeed, the most frequent pattern in R_1 is $\langle \mathsf{office} \rightarrow \mathsf{coffee} \rangle$ with 5 occurrences. We use the notion *pRegion* to refer to a region like R_1 where a particular pattern is locally frequent. Given a set of semantic trajectories, our goal is to find all the pRegions as well as their corresponding movement patterns.

The challenge of our problem is twofold. First, the problem itself is not easy to formally define. We need to clarify how a region is defined and to what extent a pattern is frequent in a region so that we can identify pRegions. It does obviously not work to pick an arbitrary region and check whether it is associated with a frequent movement pattern.

Furthermore, finding regional patterns essentially implies that we have to consider every occurrence of a pattern, which



trajectory	sequence of POIs	sequence of categories
T_1	$\langle p_1, p_{10}\rangle$	$\langle \mathtt{c}, \mathtt{s}\rangle$
T_2	$\langle p_1, p_2, p_8, p_6, p_3\rangle$	$\langle \mathtt{c}, \mathtt{s}, \mathtt{o}, \mathtt{c}, \mathtt{s}\rangle$
T_3	$\langle p_5, p_7, p_8, p_9\rangle$	$\langle \mathtt{o}, \mathtt{c}, \mathtt{o}, \mathtt{s}\rangle$
T_4	$\langle p_8, p_7, p_{12}\rangle$	$\langle \mathtt{o}, \mathtt{c}, \mathtt{s}\rangle$
T_5	$\langle p_4, p_3, p_6, p_{10}\rangle$	$\langle \mathtt{o}, \mathtt{s}, \mathtt{c}, \mathtt{s}\rangle$
T_6	$\langle p_5, p_7, p_{12}\rangle$	$\langle \mathtt{o}, \mathtt{c}, \mathtt{s}\rangle$
T_7	$\langle p_{11}, p_9\rangle$	$\langle \mathtt{c}, \mathtt{s}\rangle$
T_8	$\langle p_{11}, p_{12}\rangle$	$\langle \mathtt{c}, \mathtt{s}\rangle$

Figure 1: An example of semantic trajectories (in different colors), $T_1, T_2, ..., T_8$, where $p_1, p_2, ..., p_{12}$ are places of interest (POIs) and c, s, and o are categories of POIs representing coffee shop, shopping mall, and office, respectively.

is quite different from the classic sequential pattern mining model [1] that counts the number of pattern occurrences only *across* sequences. To simply illustrate, consider trajectory T_2 in Figure 1. If we take only one of two occurrences of $\langle \mathtt{c}, \mathtt{s}\rangle$ in T_2, either $\langle p_1, p_2\rangle$ or $\langle p_6, p_3\rangle$ must be discarded. This may lead to underestimating the local frequency of $\langle \mathtt{c}, \mathtt{s}\rangle$ within either R_1 or R_2.

This naturally connects us to the second challenge of our problem, that is, the problem is computationally hard. The problem of enumerating maximal frequent subsequences is already known as NP-hard [1, 26, 27]. In our problem, we have to consider all occurrences of a pattern as well as the spatial location where the pattern occurs, which makes the problem even more challenging. Without a carefully designed model, it is not feasible to analyse a large-scale semantic trajectory data set.

To tackle the two challenges above, our main approach is to formulate this pattern mining problem as a *pattern-based* and *density-based* clustering problem. We first introduce a new density model, called *pDensity*, that quantifies the pattern frequency in space. We then define the pRegion with respect to a pattern as a set of places of interest (POIs) constituting a cluster of *instances* of the pattern according to our pDensity metric. To obtain all the pRegions, we find all the pDensity-based clusters for each pattern. Furthermore, we propose a new type of patterns in sequences, called *compact sequential pattern*, to avoid considering excessive occurrences of a pattern.

In addition to the problem formulation, we devise an efficient mining algorithm, called *RegMiner*, to find all the pattern-based clusters using our pDensity measurement. The essential idea of *RegMiner* is starting with promising category sequences that are frequent enough to be able to have some regional patterns and then performing the density-based clustering for each such category sequence using our spatial indexing scheme.

In summary, the contributions of this paper are as follows:

- We propose a novel mining problem of finding all the regional sequential patterns in semantic trajectories. This is the first study on understanding regional movement behaviour using semantic trajectories.
- We devise the *RegMiner* algorithm for our mining problem. *RegMiner* thoroughly and efficiently finds all the regional patterns in semantic trajectories.
- Moreover, we propose a new sequential pattern scheme in sequences, called *compact sequential pattern*, together with its solution algorithm.
- A thorough experimental study is performed using real trajectory data sets. Experimental results show that our problem setting can reveal many local patterns that are hard to be discovered by a global pattern mining scheme. Further the experiments show that *RegMiner* is also highly efficient, compared to a state-of-the-art global pattern mining algorithm in semantic trajectories.

The rest of the paper is organized as follows. Section 2 reviews the existing work related to our problem. Section 3 discusses our pRegion model and the formal setting of the proposed problem. Section 4 presents the *RegMiner* algorithm. All experimental results and case studies are reported in Section 5, and we conclude in Section 6.

2. RELATED WORK

Trajectory pattern mining has been one of the most popular topics in the data mining community, and hence is studied extensively. The conventional mining scheme of trajectory data mainly aims at finding some spatio-temporal patterns from a set of raw GPS records. Those spatio-temporal patterns include frequent routes [4, 10, 5, 19], clusters of common sub-trajectories [15, 17], frequently co-locating moving objects [11, 13, 16, 22, 30], to name a few, each of which itself has produced a branch of works as a sub topic of trajectory pattern mining. A systematic review on all those sub topics is beyond the scope of this paper, and full survey articles are available [31, 9]. In the rest of this section, we focus on the topic of semantic trajectory pattern mining and regional co-location pattern mining that are closely related to our mining model.

Semantic Trajectory Pattern Mining. With the high availability of semantic trajectory data, there are a few recent studies [2, 28, 29, 14] on mining sequential patterns in semantic trajectories. Alvares *et al.* first proposed the semantic trajectory data model, and presented a preprocessing method to integrate semantic information with raw trajectories [2]. They also discussed a simple solution using some traditional sequence mining methods to mine frequent sequences of important places without considering the spatial aspect. Ying *et al.* specified this idea for the purpose of location prediction by mining frequent sequences of semantic categories [28]. Zhang *et al.* combined this problem of mining semantic category sequences with the problem of mining frequent spatial routes by dealing with groups of similar POIs rather than individual POIs [29]. By extending the semantic trajectory data model, Kim *et al.* studied a topical trajectory pattern mining problem where the goal is not only finding transition patterns but also extracting latent topics behind geo-tagged text messages [14].

All of this group of works focus on extracting globally frequent patterns in the entire data space, and none of them addresses the problem of mining regional sequential patterns in semantic trajectories.

Regional Co-location Pattern Mining. Our problem is also related to the recent topic of mining regional co-location patterns in spatio-temporal data, called the *RCP* problem. This problem was first introduced by Eick *et al.* [7] and further studied by others [20, 23, 18]. A motivating example of RCP is finding all the regions where bars or pubs are co-located with crime locations to see whether there is a correlation between alcohol and crime. The ultimate goal of the RCP problem is broadly similar to our mining problem in the sense that it is also about discovering regions along with their regional co-location patterns. A significant difference, however, is that RCP deals with a set of independent spatio-temporal or spatial points rather than a set of spatio-temporal sequences, i.e., trajectories. This line of work consequently cannot address our problem that should essentially consider the sequential aspect of spatio-temporal trajectories.

3. THE pREGION MODEL

In this section, we introduce our *pRegion* model including several new notions to define the regional movement pattern in semantic trajectories, and formulate our mining problem.

3.1 Basic Settings

Consider a set of semantic trajectories, denoted by $\mathcal{T} = \{T_1, T_2, ..., T_{|\mathcal{T}|}\}$. Let $\mathcal{P} = \{p_1, p_2, ..., p_{|\mathcal{P}|}\}$ be a set of *POIs*, and let $\mathcal{C} = \{\mathtt{c}_1, \mathtt{c}_2, ..., \mathtt{c}_{|\mathcal{C}|}\}$ be a set of *semantic categories*, where each $p \in \mathcal{P}$ is a 2D point associated with a category $\mathtt{c} \in \mathcal{C}$, denoted by $\gamma(p) = \mathtt{c}$. Following the existing definitions [29, 14], we define the *semantic trajectory* as follows:

Definition 1. (*Semantic trajectory*) A *semantic trajectory* T is a sequence of pairs of POI and a timestamp, denoted by $T = \langle (p_1, t_1), ..., (p_{\ell(T)}, t_{\ell(T)}) \rangle$, where $\ell(\cdot)$ denotes the *length* of a sequence, which is the number of elements.

For brevity, we use a simpler notation for a trajectory like $T = \langle p_1, ..., p_{\ell(T)} \rangle$ omitting timestamps as long as the context is clear. For the sequence of categories corresponding to T, we denote it as $\gamma(T) = \langle \gamma(p_1), ..., \gamma(p_{\ell(T)}) \rangle$.

To represent a semantic movement pattern in trajectories, we use the notion *category sequence* or just *pattern* interchangeably in the paper.

Definition 2. (*Category sequence*) A *category sequence* is a sequence of categories, denoted by $S = \langle \mathtt{c}_1, \mathtt{c}_2, ..., \mathtt{c}_{\ell(\mathtt{C})} \rangle$, such that each $\mathtt{c}_i \in \mathcal{C}$ for $i \in [1, \ell(S)]$.

We further say that a trajectory *covers* a category sequence or a pattern, described as:

Definition 3. Let $T = \langle p_1, ..., p_{\ell(T)} \rangle$ be a semantic trajectory and $S = \langle \mathtt{c}_1, \mathtt{c}_2, ..., \mathtt{c}_{\ell(S)} \rangle$ be a category sequence. Then, we say that T *covers* S iff. the category sequence of T, i.e., $\gamma(T)$, is a *super sequence* of S, denoted by $\gamma(T) \sqsupseteq S$, meaning that there exist integers $1 \leq j_1 < j_2 < \cdots < j_{\ell(S)} \leq \ell(t)$ such that $\gamma(p_{j_1}) = \mathtt{c}_1, \gamma(p_{c_2}) = \mathtt{c}_2, ..., \gamma(p_{j_{\ell(S)}}) = \mathtt{c}_{\ell(S)}$.

Example 1. Consider $T_3 = \langle p_5, p_7, p_8, p_9 \rangle$ in Figure 1. Its length $\ell(T_3)$ is 4 and its category sequence $\gamma(T_3)$ is $\langle \mathtt{o}, \mathtt{c}, \mathtt{o}, \mathtt{s} \rangle$. Given a category sequence $S = \langle \mathtt{c}, \mathtt{s} \rangle$, we say that T_3 covers S since $\gamma(T_3) \sqsupseteq S$.

Table 1: List of frequent symbols

Symbol	Description
$\mathcal{T} = \{T_1, T_2, ...\}$	the set of semantic trajectories
$\mathcal{P} = \{p_1, p_2, ...\}$	the set of POIs
$\mathcal{C} = \{\mathtt{c}_1, \mathtt{c}_2, ...\}$	the set of categories
$T = \langle p_1, p_2, ... \rangle$	a semantic trajectory
$\gamma(T) = \langle \gamma(p_1), \gamma(p_2), ... \rangle$	the category sequence of T
$S = \langle \mathtt{c}_1, \mathtt{c}_2, .. \rangle$	a category sequence or a pattern
$r = T[s:e]$	a pRoute within T
$\ell(T), \ell(S),$ or $\ell(r)$	the length of T, S, or r
$w(r)$	the weight of r
$\mathcal{I}(S)$	the set of all pRoutes of S
$sup(S)$	the support of S
$Nh(r)$	the neighbourhood of r
$\mathcal{N}(r)$	the set of neighbour pRoutes of r
$\lambda(r', r)$	the contribution ratio of r' to r
$\delta(r, S)$	the pDensity of r w.r.t. S
ϵ	the neighbourhood size
σ	the pDensity threshold
Δt	the maximum transition time

Table 1 summarizes the frequently used symbols throughout the paper.

3.2 Problem Formulation

The goal of our approach is to retrieve all regions in which a particular movement pattern densely appears in space, i.e., retrieving all *pRegion*s. Toward this goal, we first consider what a *region* really means in semantic trajectories. It is straightforward that a region in space is meaningful only when it contains a particular set of POIs. Thus, we can discretely define the region to be a set of POIs rather than an area of arbitrary shape in space.

In addition, a pRegion is not just a set of POIs, but should also be associated with a particular pattern that occurs frequently within the pRegion. Intuitively, the set of POIs defining a pRegion with respect to a pattern should satisfy the following criteria: (1) All the POIs should participate in covering the pattern, meaning that some of them together belong to an occurrence of the pattern in a trajectory. (2) There should be many occurrences of the pattern in trajectories passing some of those POIs. (3) All the POIs should be reasonably clustered in space.

Example 2. Based on the above criteria of a pRegion, the set of POIs in R_1 shown in Figure 1, i.e., $\{p_3, p_4, p_5, p_6, p_7, p_8\}$, can be understood as a pRegion with respect to pattern $\langle \mathtt{o}, \mathtt{c} \rangle$. All POIs of R_1 participate in covering $\langle \mathtt{o}, \mathtt{c} \rangle$. Also, there are 5 occurrences of $\langle \mathtt{o}, \mathtt{c} \rangle$ within the set of these POIs, which can be seen to be '*frequent*' given a threshold. Finally, the POIs are spatially clustered as none of the POIs is an outlier in space.

3.2.1 pRoutes: Compact Sequential Patterns in Semantic Trajectories

In order to define the pRegion, we first clarify the set of POIs that can cover a particular pattern. To this end, we need to examine where the pattern occurs in trajectories. As mentioned in the introduction, it is inevitable to consider multiple occurrences of a pattern *within a trajectory* as well as *across trajectories* for the purpose of evaluating some regions. Thus, we cannot just take one *instance* of the pattern per trajectory as we do in the conventional sequential pattern mining model [1] because every instance has its

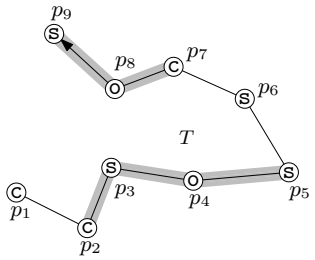


Figure 2: Trajectory $T = \langle p_1, ..., p_9 \rangle$ and its two pRoutes of pattern $S = \langle \mathtt{c}, \mathtt{o}, \mathtt{s} \rangle$ shown in gray

own importance where it occurs. At the same time, however, it is neither feasible nor semantically meaningful to take the complete set of all the pattern occurrences.

Consider the trajectory $T = \langle p_1, ..., p_9 \rangle$ and its category sequence $\gamma(T) = \langle \mathtt{c}, \mathtt{c}, \mathtt{s}, \mathtt{o}, \mathtt{s}, \mathtt{s}, \mathtt{c}, \mathtt{o}, \mathtt{s} \rangle$ in Figure 2. The total number of occurrences of a pattern $S = \langle \mathtt{c}, \mathtt{o}, \mathtt{s} \rangle$ in $\gamma(T)$ is 9, but considering all 9 instances is obviously excessive in the sense that there are only two $\mathtt{o}$'s in $\gamma(T)$. Instead, it would be more appropriate to take only two of all instances of S in T. The question is which two occurrences should be selected as *significant* ones in the context of movement patterns.

Our answer is to take pattern instances that are as *compact* as possible, which we call *compact sequential patterns*. In the case above, we choose two instances of S, namely $\langle p_2, p_3, p_4, p_5 \rangle$ and $\langle p_7, p_8, p_9 \rangle$, that are the most compact among 9 instances. The underlying principle is that the more compactly the subsequence covers S, the better it reflects the movement behaviour represented by S. Intuitively, a direct movement pattern from one category to another category is most important, and a pattern is less significant with respect to the transition between those two categories when there are multiple stops in the middle.

To refer to an instance of such a compact sequential pattern, we use the notion *pRoute* defined as follows:

Definition 4. (*pRoute*) Given a trajectory T and a pattern S, a contiguous subsequence from the s-th POI to the e-th POI of T, denoted by $T[s : e]$, covering S is a *pRoute* of S in T iff. (1) there is no other $T[s' : e']$ covering S such that $T[s' : e'] \sqsubseteq T[s : e]$, i.e., $s' \geq s$ and $e' \leq e$, and (2) the transition time between every two consecutive POIs of $T[s : e]$ is at most Δt, i.e., for $(p_i, t_i) \in T$ and $(p_{i+1}, t_{i+1}) \in T$ such that $i \in [s, e-1]$, $t_{i+1} - t_i \leq \Delta t$, where Δt is the maximum transition time.

Note that we discard every transition whose time gap is too large to be considered as a *true movement*. We denote the set of all pRoutes of S in $\mathcal{T}$ as $\mathcal{I}(S)$. For brevity, hereafter, we also use the symbol r to refer to an individual pRoute, i.e., $r = T[s : e]$.

Of course, not all pRoutes have the same amount of importance with respect to their category pattern. In the example above, pRoute $T[7 : 9]$ should be considered more significant than $T[2 : 5]$ as the former more tightly covers the pattern $\langle \mathtt{c}, \mathtt{o}, \mathtt{s} \rangle$. Intuitively, the fewer POIs a pRoute uses to cover the pattern, the more significant it is. To quantify the significance of each pRoute, we define the *weight* of each pRoute as follows:

Definition 5. (*weight of pRoute*) Let r be a pRoute of a pattern S. Then, the *weight* of r, denoted by $w(r)$ is:

$$w(r) = \frac{1}{1 + \ell(r) - \ell(S)}$$

It is not difficult to see that $w(r)$ is at most 1 for the tightest r, and gets smaller when it contains more stops in the middle. We define the *support* of S, denoted by $sup(S)$, as the total weight of pRoutes in $\mathcal{I}(S)$, that is:

$$sup(S) = \sum_{r \in \mathcal{I}(S)} w(r).$$

Example 3. Consider trajectories and a pattern $S = \langle \mathtt{o}, \mathtt{s} \rangle$ in Figure 1. $T_3[3 : 4]$ is a pRoute of S, but $T_3[1 : 4]$ is not since $T_3[3 : 4] \sqsubseteq T_3[1 : 4]$. The set of all pRoutes of S, i.e., $\mathcal{I}(S) = \{T_2[3 : 5], T_3[3 : 4], T_4[1 : 3], T_5[1 : 2], T_6[1 : 3]\}$ and its support, i.e., $sup(S)$ is $1/2 + 1 + 1/2 + 1 + 1/2 = 3.5$.

Based on our version of the support definition, the compact sequential pattern nicely satisfies the *Apriori property*, which is essential to design an efficient sequential pattern mining algorithm.

THEOREM 1. *Given two patterns S and S^+, if $S \sqsubseteq S^+$, then $sup(S) \geq sup(S^+)$.*

PROOF. First of all, the weight of each pRoute cannot increase as the pRoute grows. Also, every pRoute of S^+ also covers S as $S \sqsubseteq S^+$. Therefore, for each pRoute $r^+ \in \mathcal{I}(S^+)$, there should be at least one (or more) compact sequential pattern instance of S, i.e., pRoute of S, within r^+, implying that $|\mathcal{I}(S)| \geq |\mathcal{I}(S^+)|$. □

3.2.2 pDensity: Semantic Pattern Density Model

For a pRegion, we now quantify to what extent the pattern corresponding to the pRegion frequently occurs as well as to what extent the POIs of the pRegion are spatially close to each other. To this end, we introduce a new density model in terms of both space and pattern frequency, called *pDensity*.

To define pDensity, we extend the idea of DBSCAN [8]. The essential idea of DBSCAN is to focus on the *ϵ-neighbourhood* area for each data point, where the ϵ-neighbourhood is defined as an ϵ-radius circle centered at the point as shown in Figure 3(a). DBSCAN quantifies the density basically for each data point by counting all the points residing in the ϵ-neighbourhood of the point, and the point is considered to be *dense enough* if its ϵ-neighbourhood contains at least a certain number of points.

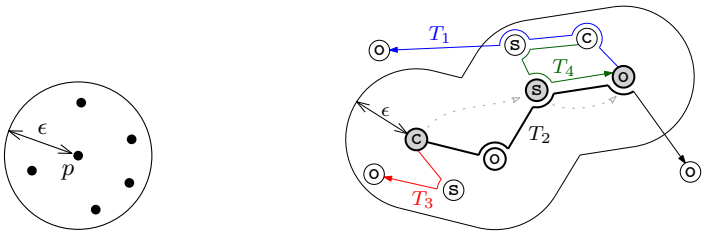
In our problem, the pDensity can be quantified by counting instances of a particular pattern, instead of counting data points distributed in space. Therefore, the pDensity should be defined for each instance of the pattern, that is, a pRoute, not for each single POI.

For each pRoute of a pattern, we define the following neighbourhood area:

Definition 6. (*ϵ-neighbourhood of pRoute*) Given a pattern S, a pRoute r of S, and a distance parameter ϵ, the *ϵ-neighbourhood* of r, denoted by $Nh(r)$, is the bounding area within distance ϵ from the polyline connecting all POIs of r.

Figure 3(b) gives an example showing the ϵ-neighbourhood of pRoute $T_2[1 : 4]$ of pattern $\langle \mathtt{c}, \mathtt{s}, \mathtt{o} \rangle$ together with other trajectories therein, where all pRoutes of $\langle \mathtt{c}, \mathtt{s}, \mathtt{o} \rangle$ are underlined in the table.

Within the ϵ-neighbourhood of each pRoute r of a pattern S, we basically count all the pRoutes of S to measure the pDensity of r. However, counting pRoutes in the neighbourhood is not as simple as counting points in DBSCAN



trajectory	sequence of categories
T_1	$\langle$o, c, s, o$\rangle$
T_2	$\langle$c, o, s, o, o$\rangle$
T_3	$\langle$c, s, o$\rangle$
T_4	$\langle$c, s, s, o$\rangle$

(a) ϵ-neighbourhood of point p in DBSCAN

(b) ϵ-neighbourhood of pRoute $T_2[1:4]$ of pattern $\langle \mathtt{c}, \mathtt{s}, \mathtt{o} \rangle$, where four trajectories (in different colors), $T_1, ..., T_4$, are in the neighbourhood

Figure 3: Different definitions of the neighbourhood in DBSCAN and in our pDensity model

because there could be many pRoutes that are not fully inside the neighbourhood such as $T_1[2:4]$ in Figure 3(b). It is not desired to discard all such pRoutes in that they still somehow contribute to the pDensity of a nearby pRoute. To determine which pRoutes should be considered, we define the following *neighbour* relationship between pRoutes.

Definition 7. (*neighbour pRoute*) Let r and r' be two pRoutes of pattern S. Then, we say that r' is a *neighbour pRoute* of r, denoted by $r' \in \mathcal{N}(r)$, iff. there is at least one POI of r' inside $Nh(r)$.[1]

Thus, we do not consider any pRoutes just passing through the neighbourhood without any stops as well as being too far away from the neighbourhood. This is based on the intuition that one has to at least *visit* a region during his or her short trip matching with the corresponding pattern in order for the trip to be somehow relevant to the region. To compute the pDensity of r, we only take these neighbour pRoutes in $\mathcal{N}(r)$.

Once again, not all neighbour pRoutes have the same amount of importance. To quantify how much a pRoute contributes to the pDensity of another pRoute having it as a neighbour, we define the *contribution ratio* of each neighbour pRoute as follows:

Definition 8. (*contribution ratio*) Let r and r' be two pRoutes of pattern S such that $r' \in \mathcal{N}(r)$. Then, the *contribution ratio* of r' to the pDensity of r, denoted by $\lambda(r', r)$, is:

$$\lambda(r', r) = \frac{\ell(r' \sqcap Nh(r))}{\ell(r')},$$

where $\ell(r' \sqcap Nh(r))$ denotes the number of POIs of r' residing in the $Nh(r)$.

Finally, we are ready to define the pDensity as follows:

Definition 9. (*pDensity*) Let r be a pRoute of pattern S. Then, the *pDensity* of r with respect to S, denoted by $\delta(r, S)$, is computed by:

$$\delta(r, S) = \frac{\ell(S)}{\ell(r)} \cdot \sum_{r' \in \mathcal{N}(r)} \lambda(r', r) \cdot w(r').$$

The first term $\frac{\ell(S)}{\ell(r)}$ is to give a *penalty* to a pRoute that is longer than its corresponding pattern. By this penalty, a longer pRoute should include more neighbour pRoutes in its larger neighbourhood region in order to have a pDensity value equal to that of a shorter one.

[1]Note that this neighbour relationship between pRoutes is reflexive, but not symmetric, i.e., $r' \in \mathcal{N}(r) \not\Rightarrow r \in \mathcal{N}(r')$. Thus, we can find a case where some POIs of r' is inside $Nh(r)$, but no POIs of r reside in $Nh(r')$.

Example 4. In Figure 3(b), let us compute the pDensity of $T_2[1:4]$ with respect to pattern $\langle \mathtt{c}, \mathtt{s}, \mathtt{o} \rangle$, i.e., $\delta(T_2[1:4], \langle \mathtt{c}, \mathtt{s}, \mathtt{o} \rangle)$. There are four pRoutes in $\mathcal{N}(T_2[1:4])$, namely $T_1[2:4]$, $T_2[1:4]$, $T_3[1:3]$, and $T_4[1:4]$, all of which cover $\langle \mathtt{c}, \mathtt{s}, \mathtt{o} \rangle$ in a compact manner.

The weight of each pRoute is as follows: $w(T_1[2:4]) = 1$, $w(T_2[1:4]) = 1/2$, $w(T_3[1:3]) = 1$, and $w(T_1[1:4]) = 1/2$.

The contribution ratio of pRoute $T_1[2:4]$ to $\delta(T_2[1:4], \langle \mathtt{c}, \mathtt{s}, \mathtt{o} \rangle)$, i.e., $\lambda(T_1[2:4], T_2[1:4])$, is $2/3$, $\lambda(T_3[1:3], T_2[1:4])$ is 1, $\lambda(T_4[1:4], T_2[1:4])$ is 1, and obviously $\lambda(T_2[1:4], T_2[1:4])$ is also 1. Therefore, $\delta(T_2[1:4], \langle \mathtt{c}, \mathtt{s}, \mathtt{o} \rangle) = 3/4 \cdot (2/3 \cdot 1 + 1 \cdot 1/2 + 1 \cdot 1 + 1 \cdot 1/2) = 2$.

Semantically, this is identical to the case where two length-3 pRoutes of $\langle \mathtt{c}, \mathtt{s}, \mathtt{o} \rangle$ are fully inside the neighbourhood region.

3.2.3 pDensity-based Clustering Scheme

Based on the pDensity defined in the previous subsection, we now formalize a density-based clustering scheme using our pDensity measurement, and finally define the pRegion.

As for the clustering scheme, the overall idea is adopted from DBSCAN [8] yet with a different density metric. Similar to DBSCAN, we have two main parameters, namely ϵ and σ, where ϵ bounds the size of the neighbourhood of each pRoute and σ is the density threshold to determine whether a pRoute is dense enough in terms of the pDensity. We first define the *dense pRoute* as follows:

Definition 10. (*dense pRoute*) Given a density threshold σ and a pRoute r of pattern S, r is a *dense pRoute* iff. $\delta(r, S) \geq \sigma$.

Unlike the DBSCAN model dealing with points, our model dealing with polylines needs a more strict version of clustering scheme to prevent the *free-riding* effect. Thus, we do not want to end up with too large clusters loosely connected by many lengthy polylines. For this purpose, we define *pDensity-reachable* and *pDensity-connected* in a more rigid way than their counterparts are defined in DBSCAN.

Definition 11. (*pDensity-reachable*) Given a dense pRoute r, a pRoute r' is *pDensity-reachable* from r iff. (1) r' is also a dense pRoute, and (2) either $r' \in \mathcal{N}(r)$ or $r' \in \mathcal{N}(r'')$ such that r'' is *pDensity-reachable* from r.

Definition 12. (*pDensity-connected*) Two dense pRoutes r_1 and r_2 are *pDensity-connected* iff. there exists a dense pRoute r' such that both r_1 and r_2 are pDensity-reachable from r'.

Note that we connect only dense pRoutes and discard all non-dense pRoutes as noises. The pDensity-connectivity is

still an equivalence relation, and therefore can partition a set of pRoutes into multiple closed subsets.

Based on the pDensity-connectivity, a pRegion corresponds to a pDensity-based cluster of pRoutes, which is formally defined as follows:

Definition 13. (*pRegion*) Given a set $\mathcal{T}$ of semantic trajectories and a set $\mathcal{P}$ of underlying POIs of $\mathcal{T}$, a *pRegion* $R \subseteq \mathcal{P}$ with respect to pattern S is a set of POIs contained in a *maximal closed set* $M \subseteq \mathcal{I}(S)$ of pRoutes such that any two pRoutes in M are pDensity-connected.

3.2.4 Problem Statement

Combining all definitions together, our final problem statement is as follows:

Definition 14. (*Problem Definition*) Given a set $\mathcal{T}$ of semantic trajectories, and parameters ϵ, σ, and Δt, the *problem of mining regional semantic trajectory patterns* is to find all the pRegions with respect to ϵ, σ, and Δt.

Each pRegion is automatically associated with a particular pattern as we only take pRoutes of the same pattern when performing the pDensity-based clustering.

4. ALGORITHM FOR MINING pREGIONS

In this section, we present the algorithm *RegMiner* that efficiently mines all the pRegions, given a semantic trajectory data set.

4.1 Solution Strategy

Finding all the pRegions takes two essential tasks. We need to (1) systematically enumerate every subsequence of semantic categories for the completeness of the result, and (2) perform pDensity-based clustering over the set of pRoutes corresponding to each category sequence.

The first task involves a huge set of candidate category sequences. Instead of checking all possible sequences, we can focus on promising ones by the following quick observation.

LEMMA 1. *Let S be a category sequence, i.e., a pattern. Based on the compact sequential pattern model in Section 3.2.1, if $sup(S) < \sigma$, then there is neither any dense pRoute nor any pRegion with respect to S.*

PROOF. This is obvious since, for any $r \in \mathcal{I}(S)$, $\delta(r, S)$ is at most $\sum_{r \in \mathcal{I}(S)} w(r) = sup(S)$ by the definition. □

By Lemma 1, it suffices to consider only compact sequential patterns whose supports are at least σ.

The next task is, for each of the promising category sequences identified by the first task, to actually compute all the pRegions with respect to the pattern. Let S be such a category sequence, and then we first have to find the set $\mathcal{I}(S)$ of all the pRoutes of S. Once we have $\mathcal{I}(S)$, all the pRegions with respect to S can be obtained by performing the pDensity-based clustering on $\mathcal{I}(S)$.

Based upon two tasks explained above, the pseudocode of *RegMiner* is outlined in Algorithm 1. Before performing two main tasks, *RegMiner* first splits each trajectory into multiple sub-trajectories so that each sub-trajectory can only contain transitions within a time period less than the maximum transition time (Line 1). By doing so, we no longer have to consider the timestamp of each trajectory in the rest steps.

ALGORITHM 1: *RegMiner* ($\mathcal{T}$, $\mathcal{P}$, $\mathcal{C}$, ϵ, σ, Δt)

Input: $\mathcal{T}$:= the set of semantic trajectories, $\mathcal{P}$:= the set of POIs, $\mathcal{C}$:= the set of categories, ϵ := the neighbourhood size parameter, σ := the pDensity threshold, Δt := the maximum transition time between two POIs
Output: $\mathcal{R}$:= the set of pairs of a pRegion and a pattern

```
1 T_Δt ← Split each T ∈ T into non-overlapping contiguous
    sub-trajectories s.t. the transition time between every two
    consecutive POIs of each sub-trajectory is at most Δt;
2 S_σ ← Find all frequent category sequences whose support is at
    least σ in T_Δt;
3 foreach category sequence S ∈ S_σ do
4     I(S) ← Find all pRoutes of S;
5     M ← Perform pDensity-based clustering on I(S);
6     foreach pRoute cluster M ∈ M do
7         P ← {all POIs in M};
8         Insert the pair (P, S) into R;
9 return R;
```

4.2 Algorithm RegMiner

We now present the algorithm *RegMiner* in detail by discussing each step of Algorithm 1.

4.2.1 Mining Compact Sequential Patterns

To find promising category sequences, we need to solve the following mining problem.

Definition 15. (*Compact sequential pattern mining*) Given a set $\mathcal{T}$ of trajectories and a support threshold σ, extract all the compact sequential patterns in the set of category sequences of trajectories in $\mathcal{T}$ whose supports are at least σ.

To address this sub problem, we employ the idea of *instance growth*, which was firstly introduced in the algorithm *GSgrow* for the problem of *mining frequent repetitive gapped subsequences* [6].

Repetitive gapped subsequences basically indicate *non-overlapping* instances of a pattern *within and across* sequences. More specifically, two subsequences of a sequence are said to be overlapping with respect to a pattern if they share at least one common item that is used to cover the pattern at the same position. For example, in the case of a category sequence $\gamma(T) = \langle \mathtt{c}, \mathtt{c}, \mathtt{s}, \mathtt{o}, \mathtt{s}, \mathtt{s}, \mathtt{c}, \mathtt{o}, \mathtt{s} \rangle$ in Figure 2, we can even say that $T[1:9]$ and $T[2:3]$ are non-overlapping with respect to pattern $\langle \mathtt{c}, \mathtt{s} \rangle$ since $T[1:9]$ can use the first $\mathtt{c}$ and the ninth $\mathtt{s}$ to cover the pattern, none of which are not shared by $T[2:3]$ at the same position.

As explained in Section 3.2.1, these non-overlapping instances differ from instances of our compact sequential pattern, that is, pRoutes. In the context of semantic trajectories, it is pointless to consider all the non-overlapping instances, considering that $T[1:9]$ becomes meaningless by taking $T[2:3]$ or $T[7:9]$ in the example above. Therefore, we need to modify the *GSgrow* algorithm so that all the compact sequential patterns, instead of repetitive gapped sequences, can be found.

Basically, the idea of instance growth is an extension of the *pattern growth* strategy used in the *PrefixSpan* algorithm [21]. We thus start with a short frequent pattern and grow this pattern until no further frequent pattern can be generated using the pattern as a prefix by the Apriori property. When growing a pattern, *GSgrow* extends each instance of the pattern as long as the instance can be extended without overlapping other instances previously extended.

Similarly, our strategy is also to grow each instance of the pattern. In the meantime, we also check whether any instances previously extended can become *non-compact* ones by the instance being extended, and delete those non-compact instances. By accessing instances within a trajectory in the ascending order of their starting positions (i.e., s's in $T[s, e]$'s), the entire process can be done without any additional overhead, compared to *PrefixSpan* and *GSgrow*.

Furthermore, to eventually obtain pRegions for each promising pattern S, we also need to compute $\mathcal{I}(S)$ (see Lines 3 in Algorithm 1). This task can also be done during the process of mining compact sequential patterns in a piggyback manner.

The following example presents how our algorithm, called *CompactGrow*, works for mining compact sequential patterns, and also shows how to retrieve all pRoutes for each pattern.

Example 5. (*CompactGrow*) For the set $\mathcal{T}$ of semantic trajectories shown in Figure 1 with $\sigma = 3$, compact sequential patterns whose support is at least σ can be mined as follows:

Step 1: Find length-1 sequential patterns. For the length-1 patterns, every instance itself is compact. Therefore, we just scan $\mathcal{T}$ and count all occurrences of each category instead of counting all sequences containing each item in *PrefixSpan*. The results are:

$$sup(\langle \mathtt{c} \rangle) = 9,\ \ sup(\langle \mathtt{s} \rangle) = 10,\ \ sup(\langle \mathtt{o} \rangle) = 6,$$

all of which are frequent with regard to $\sigma = 3$.

Also, we maintain the set of pRoutes for each length-1 pattern. To take $\langle \mathtt{o} \rangle$ as an example, the set of corresponding pRoutes, i.e., $\mathcal{I}(\langle \mathtt{o} \rangle)$, is:

$$\{T_2[3:3],\ T_3[1:1],\ T_3[3:3],\ T_4[1:1],\ T_5[1:1],\ T_6[1:1]\}$$

This information can be obtained by one scan of $\mathcal{T}$.

Step 2: Grow pRoutes for each length-1 pattern. Next, for each length-1 pattern found in Step 1, we extend each pRoute of the length-1 pattern to be the pRoute of a length-2 pattern having the length-1 pattern as a prefix. More specifically, for each pRoute $T_i[s, e]$ of pattern $\langle \mathtt{o} \rangle$ to be a pRoute of pattern $\langle \mathtt{o}, \mathtt{s} \rangle$, we find the leftmost position of category $\mathtt{s}$ after the e-th POI in $\gamma(T_i)$. For all pRoutes within the same trajectory like $T_i[s_1, e_1], T_i[s_2, e_2], ...$, we do this in the ascending order of their starting positions. To illustrate, consider pRoutes of $\langle \mathtt{o} \rangle$ in T_3, namely $T_3[1:1]$ and $T_3[3:3]$. We first access $T_3[1:1]$ and extend it to be $T_3[1:4]$ as the leftmost position having $\mathtt{s}$ is 4 in T_3, and then we also extend $T_3[3:3]$ to be $T_3[3:4]$. Importantly, at the moment of extending $T_3[3:3]$ to be $T_3[3:4]$, we check whether the preceding pRoute already extended, i.e., $T_3[1:4]$, including the currently extended pRoute, i.e., $T_3[3:4]$. In this case, since $T_3[3:4] \sqsubseteq T_3[1:4]$, we discard $T_3[1:4]$ from the set of pRoutes of $\langle \mathtt{o}, \mathtt{s} \rangle$. The resulting set $\mathcal{I}(\langle \mathtt{o}, \mathtt{s} \rangle)$ is:

$$\{T_2[3:5],\ T_3[3:4],\ T_4[1:3],\ T_5[1:2],\ T_6[1:3]\}$$

Also, the support of $\langle \mathtt{o}, \mathtt{s} \rangle$ is 3.5, still larger than σ.

Step 3: Keep growing pRoutes until the corresponding pattern is not frequent. For the set of pRoutes of each length-l pattern, we recursively do the same process, that is, growing each pRoute to be a pRoute of a length-$(l+1)$ pattern having the length-l pattern as a prefix. We safely stop this process when the support of the length-l pattern is smaller than σ since any pattern extended from the length-l pattern cannot have a larger support value, thanks to Theorem 1.

For example, since $sup(\langle \mathtt{s}, \mathtt{o} \rangle) = w(\{T_2[2:3]\}) = 1 < \sigma$, we no longer have to extend $\langle \mathtt{s}, \mathtt{o} \rangle$.

ALGORITHM 2: *CompactGrow* ($\mathcal{T}$, σ)

Input: $\mathcal{T}$ and $\mathcal{C}$:= sets of semantic trajectories and categories, σ := the support threshold
Output: the set of all compact sequential patterns having at least σ support together with their corresponding sets of pRoutes

1 $\mathcal{C}_\sigma \leftarrow$ frequent categories in $\mathcal{C}$ with at least σ occurrences in $\mathcal{T}$;
2 **foreach** $\mathtt{c} \in \mathcal{C}_\sigma$ **do**
3 $S \leftarrow \langle \mathtt{c} \rangle$;
4 $\mathcal{I}(S) \leftarrow \{T_i[s:s] | T_i \in \mathcal{T} \wedge$ s-th position of $\gamma(T_i)$ is $\mathtt{c}\}$;
5 Output S and $\mathcal{I}(S)$;
6 *PrefixSpan* (S, $\mathcal{I}(S)$);
7 **Subroutine** *PrefixSpan* (S, $\mathcal{I}(S)$)
  Input: S := a pattern, $\mathcal{I}(S)$:= the set of pRoutes of S
8 **foreach** $\mathtt{c} \in \mathcal{C}_\sigma$ **do**
9 $S^+ \leftarrow$ append $\mathtt{c}$ to S;
10 $\mathcal{I}(S^+) \leftarrow$ *RouteGrow* ($\mathtt{c}$); $sup(S^+) = \sum_{t \in \mathcal{I}(S^+)} w(t)$;
11 **if** $sup(S^+) \geq \sigma$ **then**
12 Output S^+ and $\mathcal{I}(S^+)$;
13 *PrefixSpan* (S^+, $\mathcal{I}(S^+)$);
14 **Subroutine** *RouteGrow* ($\mathtt{c}$)
  Input: $\mathtt{c}$:= a category to be appended
  Output: $\mathcal{I}(S^+)$:= the set of pRoutes of the pattern S^+ of appending $\mathtt{c}$ to S
15 $\mathcal{I}(S^+) \leftarrow \emptyset$; $s_{prev} \leftarrow 0$; $e_{prev} \leftarrow 0$;
16 **foreach** $T_i[s:e] \in \mathcal{I}(S)$ *in the ascending order of* s **do**
17 $e' \leftarrow$ the leftmost position of $\mathtt{c}$ in $\gamma(T_i)$ after e-th POI;
18 **if** $\exists$ *such* e' **then**
19 **if** $T_i[s_{prev}, e_{prev}] \sqsupseteq T_i[s, e']$ **then**
20 Delete $T_i[s_{prev}, e_{prev}]$ from $\mathcal{I}(S^+)$;
21 Insert $T_i[s, e']$ into $\mathcal{I}(S^+)$;
22 $s_{prev} \leftarrow s$, $e_{prev} \leftarrow e'$;
23 **return** $\mathcal{I}(S^+)$;

Algorithm 2 presents the details of our *CompactGrow* algorithm. We first identify frequent length-1 patterns and recursively call *PrefixSpan* for further growing patterns (Lines 1–6). When we grow pattern S by appending each category $\mathtt{c}$ to S, we invoke *RouteGrow* to obtain the set of all the pRoutes of the extended pattern, denoted by S^+ (Lines 8–10). Whenever we grow each pRoute, if the previously extended pRoute includes the pRoute currently extended, we delete the previous one as it is no longer compact (Lines 19–20). Note that we do not have to check all the previous pRoutes by accessing all the pRoutes in the same trajectory in the ascending order of their starting positions. If the size of the resulting set $\mathcal{I}(S^+)$, that is, the support of S^+, is not less than σ, we output S^+ together with $\mathcal{I}(S^+)$ as one of the frequent compact sequential patterns (Lines 11–13).

As mentioned earlier, the entire processing cost of *CompactGrow* is no more than that of *PrefixSpan* that is known as one of the most efficient algorithms for extracting sequential patterns in sequences.

4.2.2 pDensity-based Clustering on pRoutes

After identifying promising category sequences and their corresponding pRoutes, we perform the pDensity-based clustering on pRoutes of each of those sequences to find all the corresponding pRegions, which yields the following sub problem.

Definition 16. (*pDensity-based clustering problem*) Given the set $\mathcal{I}(S)$ of all pRoutes of a pattern S, and parameters ϵ and σ, find all the pDensity-based clusters on $\mathcal{I}(S)$ with respect to ϵ and σ.

Inspired by DBSCAN [8], this clustering problem can be solved by the following steps.

1. Pick the pRoute r_0 with the shortest length among *unvisited* pRoutes in $\mathcal{I}(S)$ since it is likely to have a high pDensity value due to a small penalty.
2. Mark r_0 as *visited* and compute $\delta(r_0, S)$ by retrieving all neighbour pRoutes of r_0, i.e., $\mathcal{N}(r_0)$.
3. If $\delta(r_0, S) \geq \sigma$, then construct a new pDensity-based cluster M_0 containing r_0. Otherwise, discard r_0 as a noise and go to Step 1.
4. Expand M_0 by checking all neighbour pRoutes in $\mathcal{N}(r_0)$ as follows: for each $r_i \in \mathcal{N}(r_0)$ marked as *unvisited*, mark r_i as *visited* and add r_i into M_0 if $\delta(r_i, S) \geq \sigma$ by retrieving all pRoutes in $\mathcal{N}(r_i)$.
5. If $\delta(r_i, S) \geq \sigma$, recursively repeat Step 4 with $\mathcal{N}(r_i)$ until M_0 can no longer be expanded.
6. Go to Step 1 if there still exists a pRoute marked as *unvisited* in $\mathcal{I}(S)$.

The overall performance of this procedure heavily depends on the task of retrieving neighbour pRoutes. To accelerate this task, we employ a spatial index structure, such as the R-tree [12], in the following manner.

We basically build a spatial index on the set of all POIs contained in any pRoutes in $\mathcal{I}(S)$, denoted by $\mathcal{P}(S)$. For each $p \in \mathcal{P}(S)$, we construct an inverted list of all the pRoutes containing p, and associate the inverted list with p. Then, for each $r \in \mathcal{I}(S)$, a range query on $\mathcal{P}(S)$ with $Nh(r)$ gives us all POIs, augmented with their inverted lists, that are inside $Nh(r)$. To compute the contribution ratio of each neighbour pRoute of r, we can count how many times each pRoute appears in all the inverted lists of POIs being retrieved. By doing so, we can retrieve all neighbour pRoutes of r together with their contribution ratios.

The construction of inverted lists entails $O(\sum_{r \in \mathcal{I}(S)} \ell(r))$ time. In the case of the *priority R-tree* [3], the indexing time is $O(|\mathcal{P}(S)| \log |\mathcal{P}(S)|)$ and the window range query time is $O(\sqrt{|\mathcal{P}(S)|})$, which is known as the best possible efficiency for a linear space index structure [3]. Therefore, by means of this type of spatial index, we can find the complete set of all the pRegions with respect to S in $O(|\mathcal{I}(S)|\sqrt{|\mathcal{P}(S)|} + |\mathcal{P}(S)| \log |\mathcal{P}(S)| + \sum_{r \in \mathcal{I}(S)} \ell(r))$ time since we issue at most $O(|\mathcal{I}(S)|)$ range queries for the retrieval of neighbour pRoutes.

5. EXPERIMENTS

In this section, we experimentally evaluate the performance of *RegMiner* with the following two objectives. First, we study interesting cases on regional human movement behaviours discovered by *RegMiner* using real data sets. Next, we quantitatively compare the quality, the effectiveness, and the efficiency of *RegMiner* to its competitors.

Our *RegMiner* algorithm is implemented in Java and all the experiments are conducted on a PC running Linux (Ubuntu 16.04) equipped with an Intel Core i7 CPU 3.4GHz and 16GB memory.

Table 2: The statistics of data sets

Dataset	UK	NE
Number of POIs	54,278	168,625
Number of trajectories	4,893	13,489
Number of categories	414	427
Avg. transition time (mins)	5232.63	4417.55

5.1 Data Sets

We use two real data sets in our experiments, namely UK and NE. Both of UK and NE are selected from the data set of world-wide Foursquare check-ins released by Yang *et al.* [24, 25]. Among more than 30 million check-ins and 3 million POIs over 77 countries in the data set, we take check-ins and POIs whose GPS coordinates are within the regions of the United Kingdom and the Northeastern United States to generate the data sets, UK and NE, respectively.

With the set of selected check-ins and POIs, we group and sort those check-ins by users, and thereby construct semantic trajectories for each of the data set. When we construct trajectories, we filter out noise by combining every two consecutive check-ins whose categories and locations are the same. To take trajectory $T = \langle p_1, p_2, p_3, p_4, p_5 \rangle$ and $\gamma(T) = \langle \mathtt{c}, \mathtt{s}, \mathtt{s}, \mathtt{s}, \mathtt{o} \rangle$ as example, we shrink T to be $\langle p_1, p_2, p_5 \rangle$ if p_2, p_3, and p_4 lie in exactly the same location. The statistics of the two resulting data sets are shown in Table 2.

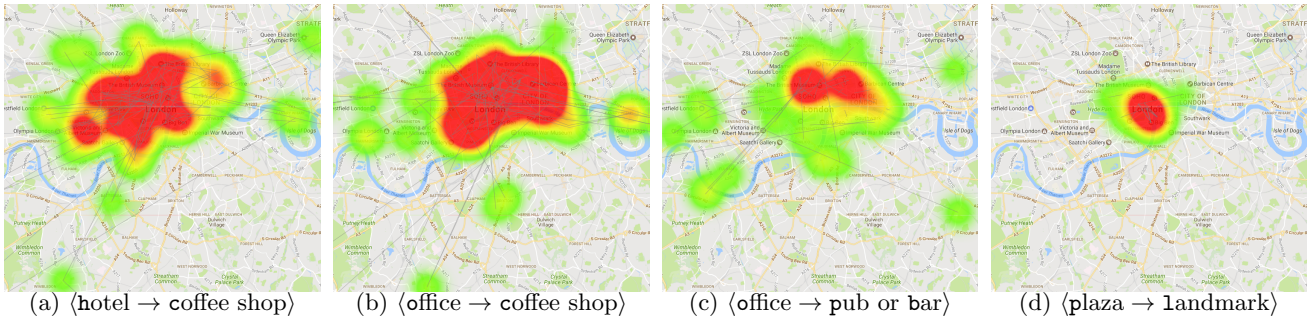
5.2 Case Study in London

Using the UK data set, we apply the *RegMiner* algorithm to discover underlying regional movement patterns, particularly focusing on the area of London. We set the default parameter values as follows: $\Delta t = 24$ (hours), $\sigma = 30$, and $\epsilon = 5$ (km). These default values are determined by considering both the statistics of the UK data set and their semantic meanings. 5 km for ϵ is a reasonably good value to quantify how close two POIs should be to be within a neighbourhood region. This is not too big but not too small compared to the average length of 33 London boroughs, 6.95 km [2]. For Δt, our default value, 24 hours, does not allow us to take movements with transition time longer than a day. Again, this value is not too long semantically yet not too small compared to the average transition time of the UK data set, 87.21 hours. The default value of σ, 30, is determined by our experimental results. Note that we thoroughly analyse the effect of varying all the parameter values in Section 5.3.

We found 184 pRegions in total for the entire area of UK, including 143 length-2 patterns, 27 length-3 patterns, and 14 length-4 or longer patterns. Among all the resulting pRegions in UK, we show some interesting regional patterns appearing in London as presented in Figure 4. For the effective presentation reflecting different density values of pRoutes, we draw a heat map using the Google Map API[3], where the

[2]https://data.london.gov.uk

[3]https://developers.google.com/maps/



(a) ⟨hotel → coffee shop⟩ (b) ⟨office → coffee shop⟩ (c) ⟨office → pub or bar⟩ (d) ⟨plaza → landmark⟩

Figure 4: Illustrating movement patterns and their pRegions in London

Table 3: The statistics of illustrating movement patterns in London

Pattern	# pRoutes	Support	Avg. pDensity	# dense pRoutes	# pRegions
⟨hotel → coffee shop⟩	134	91.91	32.24	78	1
⟨office → coffee shop⟩	226	169.56	57.85	163	2
⟨office → pub⟩	152	95.12	16.50	37	1
⟨office → bar⟩	92	60.16	11.92	1	1
⟨home → grocery store⟩	274	156.94	5.66	0	0
⟨plaza → landmark⟩	52	33.50	21.55	21	1

density of each pRoute is assigned to the weight values of the relevant POIs. The detailed statistics for the regional patterns are summarized in Table 3.

One of the most representative regional movement patterns in London is the transition from a hotel to a coffee shop (shown in Figure 4(a)). This pattern lies in a quite broad area especially concentrating on the central area yet covering many famous attractions as well. The region of the transition from an office to a coffee shop (shown in Figure 4(b)) shows a similar coverage, but it is more focused on the right center of the city where Soho and City of London are located. This is sensible in that there are many companies and work places located in this area. However, the neighbourhood of Kensington containing many attractions like Kensington Palace and Hyde Park is not densely covered by the pattern ⟨office → coffee shop⟩ unlike the area of ⟨hotel → coffee shop⟩.

Compared to ⟨office → coffee shop⟩, another interesting movement pattern is the pattern ⟨office → pub or bar⟩ (shown in Figure 4(c)). We can first observe that the overall density of ⟨office → pub or bar⟩ is much lower than that of ⟨office → coffee shop⟩, yet is relatively more concentrated on the area of Soho. Also, it seems that people are willing to cross the River Thames when they go to a pub or a bar after work as the dense pRoutes of ⟨office → pub or bar⟩ are uniformly distributed across the river. This is apparently not the case in ⟨office → coffee shop⟩ whose pRegion is strongly biased to the northern part of the river. Due to a similar reason, there is no pRegion with respect to ⟨office → pub or bar → office⟩, but we found a pRegion of ⟨office → coffee shop → office⟩ that occupies a small part of the pRegion of ⟨office → coffee shop⟩.

We also observe that a globally frequent pattern does not necessarily lead to its regional patterns. For example, the pattern ⟨home → grocery store⟩ is highly frequent in the entire space as its support value is 156.94, but it does not yield any pRegions. This is intuitive in the sense that, even though there are a good number of residences and grocery stores, transitions between them are scattered all around the city without involving any regional behaviours.

On the contrary, some globally insignificant patterns occasionally have regional patterns. For instance, the support of ⟨plaza → landmark⟩ (shown in Figure 4(d)) is only 33.5 barely over $\sigma = 30$, but it shows quite dense regional patterns. We found that the relevant region is the heart of London for tourists, where Big Ben, London Eye, and Buckingham Palace are located.

5.3 Quantitative Study

We now quantitatively study whether our *RegMiner* algorithm is effective and efficient particularly compared to the existing global pattern mining scheme in semantic trajectories as well as a simple mining method using a grid structure. We also examine how various parameter values affect the performance of *RegMiner*.

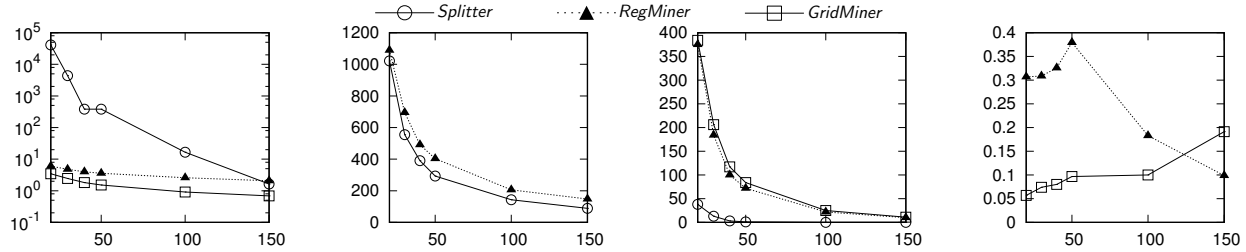
5.3.1 Baselines

GridMiner. For quality comparison, we implement a grid-based regional pattern mining algorithm, called *GridMiner*, where all *dense* cells with high pattern frequency are returned. Since it is too naïve to use a grid structure with a fixed granularity, we apply a quadtree-based space partitioning scheme to *GridMiner*. *GridMiner* takes the entire space as a cell at the beginning, and check if there is any dense pattern with respect to the cell. If so, the cell is recursively divided into 4 sub cells (i.e., quadrants) and do the same test for each sub cell.

The issue is how to estimate the density of each cell with respect to a pattern S. Intuitively, we can imagine that each cell consists of multiple ϵ-neighbourhood regions, and the area of each ϵ-neighbourhood region can be modelled as $c \cdot \pi\epsilon^2 \cdot \ell(S)$ for some constant factor c. Then, we can estimate the density of a cell for pattern S as:

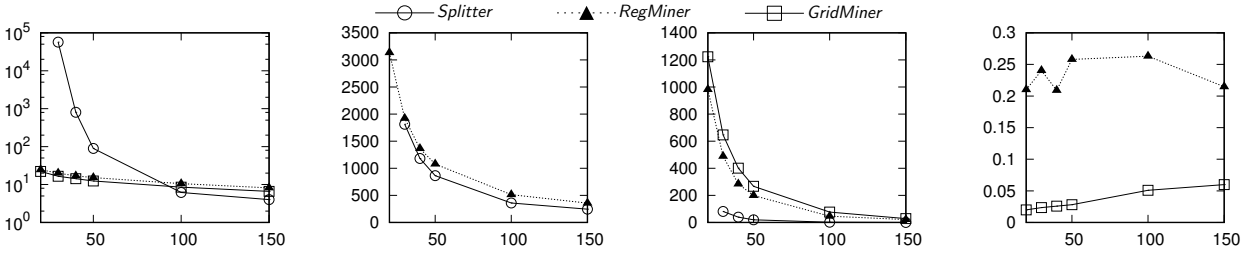
$$sup(S) \times \frac{c \cdot \pi\epsilon^2 \cdot \ell(S)}{\text{the area of the cell}}.$$

Given a parameter value c, *GridMiner* finds all the cells and their corresponding patterns whose estimated density values are not less than σ. The case of $c = 1$ means we consider each neighbourhood region as the compact union



(a) Execution time (secs) (b) Number of global patterns (c) Number of local patterns (d) Average cirDensity

Figure 5: Experimental results using UK with varying the pattern frequency threshold (σ) from 20 to 150



(a) Execution time (secs) (b) Number of global patterns (c) Number of local patterns (d) Average cirDensity

Figure 6: Experimental results using NE with varying the pattern frequency threshold (σ) from 20 to 150

of only $\ell(S)$ ϵ-radius circles. This strict c value makes *GridMiner* extremely ineffective to the point that no patterns are found in most cases. When c gets larger, we also include some intermediate areas for a transition between two POIs, leading to more patterns returned. Thus, there is a trade-off between the effectiveness and the quality controlled by the c value. We set c to 600 in all experiments, which is the value making the effectiveness of *GridMiner* (i.e., the number of output patterns) close to that of *RegMiner* with default values of the other parameters.

***Splitter*.** We take another competitor, *Splitter* proposed by Zhang *et al.* [29], by employing the authors' implementation in Java. The *Splitter* algorithm is the state-of-the-art algorithm for finding globally frequent sequential patterns in semantic trajectories.

Even though *Splitter* is not designed for finding regional patterns, it interestingly considers regional properties in the sense that it groups POIs of the same category that are spatially close to each other. More specifically, *Splitter* finds all sequential patterns of such groups of POIs, called *fine-grained sequential patterns*. Therefore, it is worthwhile testing whether *Splitter* can be adapted to mine regional patterns using a small support threshold.

5.3.2 Quantitative Metrics

Quality. For quality comparison between *RegMiner* and *GridMiner*, we devise a new quality metric, called *cirDensity*, which is intuitively the ratio of the number of pattern occurrences to the area of the pRegion. To approximate the area of an arbitrary shaped pRegion, we first compute the *smallest enclosing circle* (*SEC*) for the set of POIs in the pRegion, and enlarge the diameter of the SEC by ϵ. Then the cirDensity of a pRegion R is defined as:

$$\frac{\text{the weighted number of pattern occurrences in } R}{\text{the area of the } \epsilon\text{-enlarged SEC of } R \text{ (km}^2\text{)}}.$$

Note that the cirDensity actually evaluates our proposed density metric (i.e., pDensity) that is always optimized by *RegMiner*.

Efficiency and Effectiveness. For the efficiency test, we measure the running time, and we follow the rules of *Splitter* to evaluate the effectiveness of all competing algorithms, that is, the more patterns, the more effective [29]. Note that this rule is well matched with *RegMiner* and *GridMiner* as we also want our resulting set as complete as possible [29].

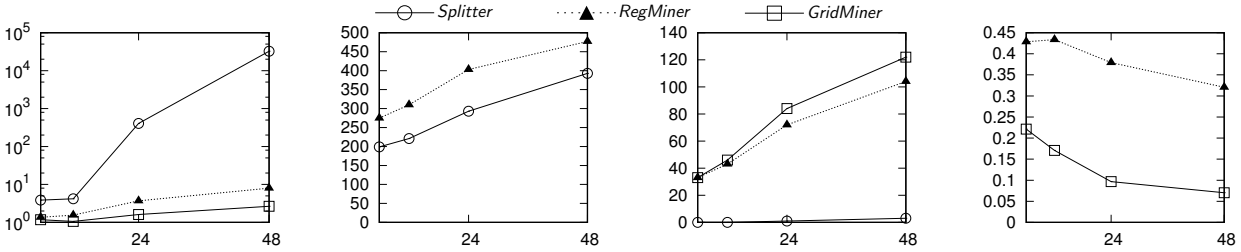
5.3.3 Comparison on Quality, Efficiency, and Effectiveness

Effect of the pattern frequency threshold (σ). We first vary the pattern frequency threshold (σ) with default values of the other parameters. Figures 5 and 6 show the experimental results using UK and NE, respectively.

To give similar conditions to *RegMiner* and *Splitter*, we use the fact that both *RegMiner* or *Splitter* first find globally frequent patterns regardless of the spatial aspect in its own scheme, and then process the set of all instances (occurrences) for each globally frequent pattern to finally discover its local (regional) patterns. Even though the meaning of the support threshold in the two algorithms are not exactly the same, we observe that both algorithms retrieve a similar number of global patterns, i.e., frequent category sequences, as shown in Figures 5(b) and 6(b). Thus, it can be seen that both algorithms process a similar amount of workload to ultimately find their final local patterns.

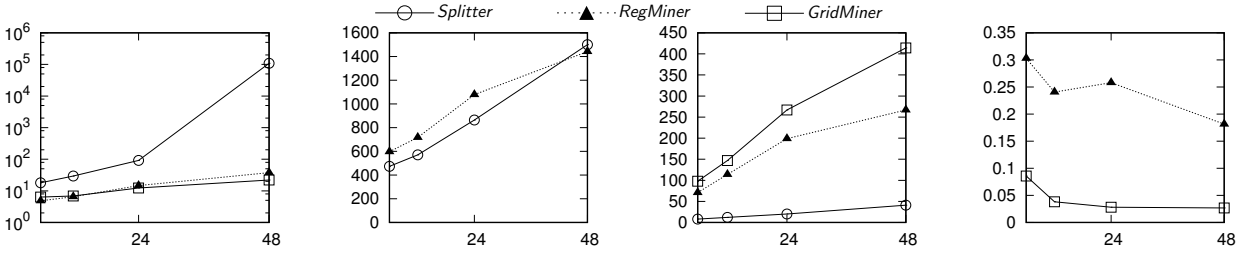
In both of the results using UK and NE, it is remarkably observed that *RegMiner* deals with even a larger set of global patterns yet runs a lot faster than *Splitter* does. Furthermore, *RegMiner* effectively discovers much more regional patterns, using a reasonably small support threshold such as 20. *Splitter* fails to capture any local patterns in most of the cases using thresholds larger than 50. This is probably due to the fact that *Splitter* counts only one occurrence for each pattern, which turns out to be inappropriate for mining regional patterns as explained in Section 3.2.1.

When trying *Splitter* to return many local patterns by decreasing the support threshold, we observe that *Splitter* gets



(a) Execution time (secs) (b) Number of global patterns (c) Number of local patterns (d) Average cirDensity

Figure 7: Experimental results using UK with varying the maximum transition time (Δt hours) from 6 to 48



(a) Execution time (secs) (b) Number of global patterns (c) Number of local patterns (d) Average cirDensity

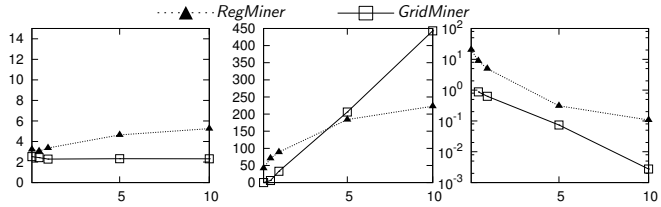
Figure 8: Experimental results using NE with varying the maximum transition time (Δt hours) from 6 to 48

a few orders of magnitude slower than *RegMiner*, as shown in Figures 5(a) and 6(a). In the case of $\sigma = 20$ using the NE data set *Splitter* does not even terminate in several days. The main reason is that *Splitter* checks every occurrence of a pattern in a trajectory even though it counts only one of them. As explained in Section 3.2.1, the entire search space of considering all possible occurrences could be prohibitively huge particularly for long patterns. By considering only the reasonably significant occurrences, what we call pRoutes, *RegMiner* achieves the fairly high efficiency as well as the effectiveness.

In terms of the efficiency and the effectiveness, both *RegMiner* and *GridMiner* show remarkable performance, compared to *Splitter*. This is because *GridMiner* basically shares the main idea with *RegMiner*, that is, taking only compact sequential patterns, defining the density of a pattern with respect to a region, and returning regional patterns based on our density scheme. The drawback of *GridMiner* lies in the quality comparison. In most cases, *GridMiner* turns out to return less useful regions having a low cirDensity value, as shown in Figures 5(d) and 6(d).

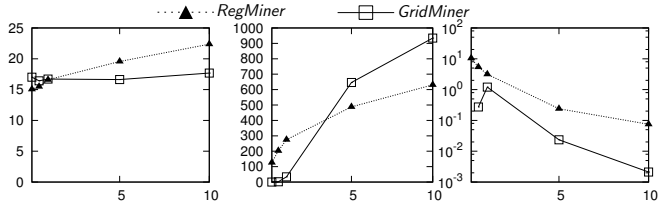
Effect of the maximum transition time (Δt). We conduct the same experiments varying the maximum transition time (Δt). We also set the pattern frequency threshold to 50 by default since it is the largest value enabling *Splitter* to return at least some local patterns (i.e., the most favourable value for speeding up *Splitter*). The corresponding results are shown in Figures 7 and 8.

Once again, *RegMiner* and *GridMiner* are superior to *Splitter* in terms of both the efficiency and the effectiveness for the goal of mining regional patterns. Overall, all the algorithms clearly show the increasing trend of execution time and the number of patterns when increasing Δt values. This is because a larger Δt value allows more transitions between POIs to be processed, leading to longer global patterns and consequently longer final local patterns. Apparently, we can observe that dealing with longer patterns



(a) Execution time (secs) (b) Number of local patterns (c) Average cirDensity

Figure 9: Experimental results using UK with varying the neighbourhood size (ϵ kilometres) from 0.1 to 10



(a) Execution time (secs) (b) Number of local patterns (c) Average cirDensity

Figure 10: Experimental results using NE with varying the neighbourhood size (ϵ kilometres) from 0.1 to 10

severely slows down *Splitter*, but *RegMiner* scales quite well even with larger Δt values, as shown in Figures 7(a) and 8(a). *GridMiner* again exposes its fundamental weakness at the quality. In every case, the cirDensity of *GridMiner* is a few times smaller than that of *RegMiner*, as shown in Figures 7(d) and 8(d).

Effect of the Neighbourhood Size (ϵ). We finally examine how the size of the neighbourhood region (ϵ) affects the performance of our algorithms, *RegMiner* and *GridMiner*. As observed in Figures 9(a), 10(a), 9(b), and 10(b), the lager the ϵ value, the longer the execution time and

the more pRegions we can find, in both UK and NE. The numbers of global patterns obtained by *RegMiner* using UK and NE remain 695 and 1922, respectively. It appears that the efficiency and the effectiveness of *GridMiner* are slightly better than those of *RegMiner*. However, it is important to note that the quality of *GridMiner* becomes drastically low when ϵ gets large, as shown in Figures 9(c) and 10(c) where the cirDensity values on y-axis are in log scale.

6. CONCLUSION

Understanding human movement behaviour using semantic trajectories attracted the data mining community in recent years. In this paper, we provide the first mining tool for discovering significant regions showing particular movement patterns. In order to effectively and efficiently grasp regional movement patterns, we model this problem as a practically solvable one by introducing new sub problems like compact sequential pattern mining and pDensity-based clustering, and thereby devise an efficient mining algorithm, *RegMiner*. We apply our algorithm to real data sets, and gain some geographic insights on regional movement behaviour in urban areas such as London. The extensive quantitative experimental results also show that our algorithm is not only more effective with good quality but also far more efficient than a state-of-the-art global mining algorithm on semantic trajectories.

Although this work mainly deals with trajectories, we believe that our regional pattern mining framework can be applied to many different types of sequential data by adopting a different neighbour relationship between pattern instances other than the spatial closeness. For example, given the assumption that we can collect a set of users' web browsing histories, mining regional patterns in the network of web pages can help us to find some interesting groups of web pages that are semantically similar and close to each other in terms of a network distance metric.

7. REFERENCES

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, pages 3–14, 1995.

[2] L. O. Alvares, V. Bogorny, B. Kuijpers, J. A. F. de Macêdo, B. Moelans, and A. A. Vaisman. A model for enriching trajectories with semantic geographical information. In *ACM GIS*, number 22, 2007.

[3] L. Arge, M. de Berg, H. J. Haverkort, and K. Yi. The priority r-tree: A practically efficient and worst-case optimal r-tree. *ACM TALG*, 4(1), 2008.

[4] H. Cao, N. Mamoulis, and D. W. Cheung. Mining frequent spatio-temporal sequential patterns. In *ICDM*, pages 82–89, 2005.

[5] Z. Chen, H. T. Shen, and X. Zhou. Discovering popular routes from trajectories. In *ICDE*, pages 900–911, 2011.

[6] B. Ding, D. Lo, J. Han, and S. Khoo. Efficient mining of closed repetitive gapped subsequences from a sequence database. In *ICDE*, pages 1024–1035, 2009.

[7] C. F. Eick, R. Parmar, W. Ding, T. F. Stepinski, and J. Nicot. Finding regional co-location patterns for sets of continuous variables in spatial datasets. In *SIGSPATIAL*, page 30, 2008.

[8] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*, pages 226–231, 1996.

[9] Z. Feng and Y. Zhu. A survey on trajectory data mining: Techniques and applications. *IEEE Access*, 4:2056–2067, 2016.

[10] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi. Trajectory pattern mining. In *SIGKDD*, pages 330–339, 2007.

[11] J. Gudmundsson and M. J. van Kreveld. Computing longest duration flocks in trajectory data. In *SIGSPATIAL*, pages 35–42, 2006.

[12] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[13] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *PVLDB*, 1(1):1068–1080, 2008.

[14] Y. Kim, J. Han, and C. Yuan. TOPTRAC: topical trajectory pattern mining. In *SIGKDD*, pages 587–596, 2015.

[15] J. Lee, J. Han, and K. Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD*, pages 593–604, 2007.

[16] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *PVLDB*, 3(1):723–734, 2010.

[17] Z. Li, J. Lee, X. Li, and J. Han. Incremental clustering for trajectories. In *DASFAA*, pages 32–46, 2010.

[18] B. Liu, L. Chen, C. Liu, C. Zhang, and W. Qiu. RCP mining: Towards the summarization of spatial co-location patterns. In *SSTD*, pages 451–469, 2015.

[19] W. Luo, H. Tan, L. Chen, and L. M. Ni. Finding time period-based most frequent path in big trajectory data. In *SIGMOD*, pages 713–724, 2013.

[20] P. Mohan, S. Shekhar, J. A. Shine, J. P. Rogers, Z. Jiang, and N. Wayant. A neighborhood graph based approach to regional co-location pattern discovery: a summary of results. In *SIGSPATIAL*, pages 122–132, 2011.

[21] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *ICDE*, pages 215–224, 2001.

[22] L. A. Tang, Y. Zheng, J. Yuan, J. Han, A. Leung, W. Peng, and T. F. L. Porta. A framework of traveling companion discovery on trajectory data streams. *ACM TIST*, 5(1):3:1–3:34, 2013.

[23] S. Wang, Y. Huang, and X. S. Wang. Regional co-locations of arbitrary shapes. In *SSTD*, pages 19–37, 2013.

[24] D. Yang, D. Zhang, L. Chen, and B. Qu. Nationtelescope: Monitoring and visualizing large-scale collective behavior in lbsns. *Journal of Network and Computer Applications*, 55:170–180, 2015.

[25] D. Yang, D. Zhang, and B. Qu. Participatory cultural mapping based on collective behavior in location based social networks. *ACM TIST*, 2015.

[26] G. Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *SIGKDD*, pages 344–353, 2004.

[27] G. Yang. Computational aspects of mining maximal frequent patterns. *Theor. Comput. Sci.*, 362(1-3):63–85, 2006.

[28] J. J. Ying, W. Lee, T. Weng, and V. S. Tseng. Semantic trajectory mining for location prediction. In *SIGSPATIAL*, pages 34–43, 2011.

[29] C. Zhang, J. Han, L. Shou, J. Lu, and T. F. L. Porta. Splitter: Mining fine-grained sequential patterns in semantic trajectories. *PVLDB*, 7(9):769–780, 2014.

[30] K. Zheng, Y. Zheng, N. J. Yuan, S. Shang, and X. Zhou. Online discovery of gathering patterns over trajectories. *IEEE TKDE*, 26(8):1974–1988, 2014.

[31] Y. Zheng. Trajectory data mining: An overview. *ACM TIST*, 6(3):29:1–29:41, 2015.