Splitter: Mining Fine-Grained Sequential Patterns in Semantic Trajectories

Chao Zhang[1] Jiawei Han[1] Lidan Shou[2] Jiajun Lu[1] Thomas La Porta[3]
[1]Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, USA
[2]College of Computer Science, Zhejiang University, China
[3]Dept. of Computer Science and Engineering, Penn State University, University Park, PA, USA
[1]{czhang82, hanj, jlu23}@illinois.edu [2]should@zju.edu.cn [3]tlp@cse.psu.edu

ABSTRACT

Driven by the advance of positioning technology and the popularity of location-sharing services, semantic-enriched trajectory data have become unprecedentedly available. The sequential patterns hidden in such data, when properly defined and extracted, can greatly benefit tasks like targeted advertising and urban planning. Unfortunately, classic sequential pattern mining algorithms developed for transactional data cannot effectively mine patterns in semantic trajectories, mainly because the places in the continuous space cannot be regarded as independent "items". Instead, similar places need to be grouped to collaboratively form frequent sequential patterns. That said, it remains a challenging task to mine what we call *fine-grained sequential patterns*, which must satisfy spatial compactness, semantic consistency and temporal continuity simultaneously. We propose SPLITTER to effectively mine such fine-grained sequential patterns in two steps. In the first step, it retrieves a set of spatially coarse patterns, each attached with a set of trajectory snippets that precisely record the pattern's occurrences in the database. In the second step, SPLITTER breaks each coarse pattern into fine-grained ones in a top-down manner, by progressively detecting dense and compact clusters in a higher-dimensional space spanned by the snippets. SPLITTER uses an effective algorithm called weighted snippet shift to detect such clusters, and leverages a divide-and-conquer strategy to speed up the top-down pattern splitting process. Our experiments on both real and synthetic data sets demonstrate the effectiveness and efficiency of SPLITTER.

1. INTRODUCTION

A *semantic trajectory* [2] is a sequence of timestamped places wherein each place is described by a spatial location as well as a semantic label (*e.g.*, office, park). By virtue of improved positioning accuracy, raw GPS trajectories can be readily linked with external semantic information (*e.g.*, land use data) for enrichment [2, 13]. Meanwhile, location-sharing services like Facebook Places and Foursquare allow people to check-in at different places, each having a spatial location and a semantic category. The check-in sequence of a user is essentially a low-sampling semantic trajectory, millions of which have been collected by each service provider.

Proceedings of the VLDB Endowment, Vol. 7, No. 9

The unprecedented availability of semantic trajectory data opens door to understanding object movement along the spatial, temporal and semantic dimensions simultaneously. Consider the following questions: (1) Where do people working in Manhattan usually go to relax after work? (2) Which restaurants do people prefer after shopping at the Fifth Avenue? (3) Are there any popular sightseeing routes for a one-day trip in Paris? The answers to such questions can greatly benefit a wide spectrum of real-life tasks, such as targeted advertising, urban planning and location prediction.

We answer the above questions by exploring *fine-grained sequential patterns* in semantic trajectories. Given a sequence database $\mathcal{D}$ and a threshold σ, a sequential pattern is typically defined as a subsequence that matches at least σ sequences in $\mathcal{D}$. Semantic trajectory data, however, introduce new challenges to this definition and conventional sequential pattern mining algorithms [1, 10, 16]. To illustrate, Figure 1 shows a semantic trajectory database consisting of 5 objects $\{o_1, o_2, \ldots, o_5\}$ and 12 places $\{p_1, p_2, \ldots, p_{12}\}$. Let $\sigma = 3$. By treating each place p_i $(1 \leq i \leq 12)$ as an independent "item", we fail to find any frequent sequences. However, if we group *similar* places together, interesting patterns may emerge. For instance, let $G_1 = \{p_1, p_2\}, G_2 = \{p_7, p_8\}, G_3 = \{p_9, p_{10}, p_{11}\}$, the sequence $G_1 \rightarrow G_2 \rightarrow G_3$ becomes frequent as it appears in the trajectories of o_1, o_2 and o_4. Each of G_1, G_2 and G_3 contains several places that are spatially close and in the same category. The pattern $G_1 \rightarrow G_2 \rightarrow G_3$ thus clearly reveals a common behavior that people working in area G_1 like to exercise at gym in G_2 after work, and then dine at restaurants in G_3.

The above running example leads to the following observation: to find frequent sequential patterns in semantic trajectories, one should group similar places together. However, while numerous sequential patterns can be formed by adopting different grouping strategies, not all of them are interesting. Specifically, a pattern can reflect movement regularity only when the following conditions are met: (1) *Spatial compactness.* The groups in a pattern should not include places that are too faraway, otherwise the pattern becomes spatially pointless. In the above example, if $G_1 = \{p_1, p_2, p_4, p_5\}$, $G_2 = \{p_3, p_7, p_8, p_{12}\}$, $G_3 = \{p_6, p_9, p_{10}, p_{11}\}$, we still obtain a frequent sequence $G_1 \rightarrow G_2 \rightarrow G_3$. Nonetheless, the sequence offers little insight along the spatial dimension as G_1, G_2 and G_3 are spatially scattered. (2) *Semantic consistency.* The semantics of the places in each group should be consistent. If we put places from different categories into the same group, say $G_2 = \{p_5, p_6, p_7\}$, the semantic meaning of the group becomes obscure. As such, the result patterns become semantically pointless. (3) *Temporal continuity.* The pattern $G_1 \rightarrow G_2 \rightarrow G_3$ in Figure 1 is interesting as both transitions $G_1 \rightarrow G_2$ and $G_2 \rightarrow G_3$ occur in no more than 60 minutes. If the transition time between two consecutive groups is too large, say one year, the pattern becomes temporally pointless.

Object	Semantic Trajectory
o_1	<(p_3, 0), (**p_1**, 10), (**p_7**, 30), (**p_9**, 40)>
o_2	<(p_5, 0), (p_7, 30), (**p_2**, 360), (**p_7**, 400), (**p_{10}**, 420)>
o_3	<(p_3, 0), (p_6, 30)>
o_4	<(p_2, 0), (**p_1**, 120), (p_6, 140), (**p_8**, 150), (**p_{11}**, 180)>
o_5	<(p_{12}, 50), (p_8, 80), (p_{11}, 120), (p_4, 210)>

** Places $p_1 p_2 \cdots p_{12}$ are shown on the right.*

** The timestamps are in minute.*

** Bold elements match the pattern $G_1 \rightarrow G_2 \rightarrow G_3$.*

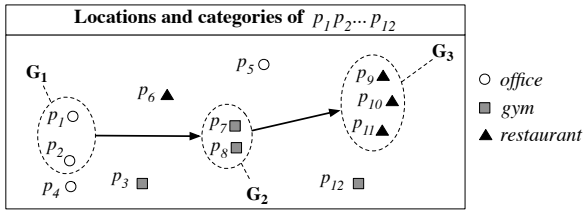


Figure 1: Semantic trajectories of $o_1, o_2, \ldots, o_5$ and an example fine-grained sequential pattern $G_1 \rightarrow G_2 \rightarrow G_3$ ($\sigma = 3$).

We call the patterns satisfying the above three conditions *fine-grained sequential patterns*, and seek to mine them in an effective and efficient manner. Fine-grained sequential patterns are important for various real-life tasks. Let us consider targeted advertising as an example. Suppose the restaurant p_9 in Figure 1 wants to advertise to promote sales. Knowing that many people follow the pattern $G_1 \rightarrow G_2 \rightarrow G_3$, the restaurant can simply advertise around the regions G_1 and G_2 to effectively attract potential customers. As another example, by extracting fine-grained sequential patterns in a city, we can understand how the populace flow. Such an understanding can play a key role in improving the design of transportation systems and road networks.

Despite its importance, mining fine-grained sequential patterns is a non-trivial task. The major challenge is, how to design an effective grouping strategy to ensure the result sequences are frequent and meanwhile fine-grained? A bruteforce solution that enumerates all the possible combinations of places is exponential in nature. Several methods [11, 12, 5] have been proposed for mining sequential patterns in GPS trajectories, but none of them can effectively address our problem either. To handle spatial continuity, all these methods partition the whole space into numerous small grids, and group the places falling inside the same grid (or several neighboring grids). Though simple and efficient, rigid space partitioning is ineffective for mining fine-grained patterns because: (1) It suffers from the sharp boundary problem. That is, the places close to the grid boundaries may be assigned into different groups and thus potential patterns can be lost. (2) It requires a pre-specified granularity for partitioning. For our problem, it is hard to pre-specify a proper granularity as it may be either too coarse to generate compact groups or too fine to discover frequent patterns. (3) Spatial proximity should not be the only criterion for grouping places. For instance, in Figure 1, p_2 is closer to p_4 than p_1, but if we let $G_1 = \{p_2, p_4\}$, the pattern $G_1 \rightarrow G_2 \rightarrow G_3$ becomes infrequent. Hence, the grouping should consider not only spatial proximity, but also the sequential information in the database.

Contributions. We propose SPLITTER, which employs two steps to effectively discover fine-grained sequential patterns. In the first step, SPLITTER groups all the places by category and retrieves a set of coarse patterns from the database. These coarse patterns disregard the spatial compactness constraint, but guarantee semantic consistency and temporal continuity. The discovery of such coarse patterns greatly reduces the search space of fine-grained patterns, because any fine-grained pattern must have one and only one coarse pattern as its *parent* pattern. SPLITTER also attaches each coarse pattern with a set of *trajectory snippets*, which are the place sequences corresponding to the pattern's occurrences in the database.

In the second step, SPLITTER treats each coarse pattern independently and obtains fine-grained patterns by splitting a coarse pattern in a top-down manner. Specifically, SPLITTER splits a coarse pattern by clustering its snippets, and then extracts fine-grained patterns from those *dense* and *compact* snippet clusters. The clusters need to be *dense* to meet the support threshold σ and be *compact* to ensure the patterns' spatial compactness. The key benefit of clustering snippets is that the grouping of places considers not only spatial proximity but also the sequential information encoded in the snippets. The snippet clustering is underpinned by an effective algorithm called *weighted snippet shift*, which allows similar snippets to *shift* to the same stationary point and form compact clusters. For the unqualified snippet clusters, *i.e.*, the clusters that cannot form fine-grained patterns, SPLITTER refines the clustering granularity to discover additional patterns from them. Such a process continues until no more fine-grained patterns exist.

Furthermore, to speed up the top-down pattern splitting process, after each round of clustering, we organize the unqualified snippet clusters into several disjoint *communities* that are mutually faraway. We analytically prove that the further splitting of each community is autonomous. Better still, small communities that cannot exceed support threshold are pruned early on to avoid unnecessary splitting. Therefore, SPLITTER can generate fine-grained patterns in a divide-and-conquer manner with excellent efficiency.

Our contributions can be summarized as follows:

(1) We introduce the problem of mining fine-grained sequential patterns in semantic trajectories. To the best of our knowledge, we are first in attempting to find sequential patterns that reflect fine-grained movement regularity along the spatial, temporal and semantic dimensions simultaneously.

(2) We develop SPLITTER for the proposed problem. SPLITTER does not rely on fixed space partitioning. Instead, it is a data-driven approach, which effectively mines fine-grained sequential patterns with excellent efficiency.

(3) Our extensive experiments on both real and synthetic data sets show that, SPLITTER is flexible to discover fine-grained patterns in various settings, and it outperforms compared methods significantly in terms of both effectiveness and efficiency.

2. PRELIMINARIES

2.1 Problem Description

Let $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ be a set of places and $\mathcal{C}$ be a set of semantic categories. Each place $p \in \mathcal{P}$ is defined as a tuple $(p.loc, p.cat)$. Here, $p.loc$ is a two-dimensional vector representing p's spatial location, and $p.cat \in \mathcal{C}$ is p's category. With these notations, we define *semantic trajectory* as follows.

DEFINITION 1 (SEMANTIC TRAJECTORY). *Given a moving object o, its semantic trajectory is a sequence of timestamped places $\langle (p_1, t_1), (p_2, t_2), \ldots, (p_l, t_l) \rangle$ where $t_i < t_j$ if $i < j$, and each element (p_i, t_i) means o is at place $p_i \in \mathcal{P}$ at time t_i.*

Given a semantic trajectory database $\mathcal{D}$, our goal is to find frequent sequential patterns in $\mathcal{D}$. Due to spatial continuity, similar places need to be grouped to collaboratively form frequent patterns. Below, we introduce the concepts of *G-sequence* and *containment*.

DEFINITION 2 (G-SEQUENCE). *A length-k group sequence (G-sequence) T has the form $T = G_1 \xrightarrow{\Delta t} G_2 \xrightarrow{\Delta t} \cdots \xrightarrow{\Delta t} G_k$, where (1) $G_i \subseteq \mathcal{P}$ $(1 \leq i \leq k)$ is a group of places; and (2) Δt is the maximum transition time between any two consecutive groups.*

DEFINITION 3 (CONTAINMENT). *Given a semantic trajectory $o = \langle (p_1, t_1), (p_2, t_2), \ldots, (p_l, t_l) \rangle$ and a G-sequence $T = G_1 \xrightarrow{\Delta t} G_2 \cdots \xrightarrow{\Delta t} G_k$ $(k \leq l)$, o contains T (denoted as $T \sqsubseteq o$) if there exist integers $1 \leq j_1 < j_2 < \cdots < j_k \leq l$ such that: (1) $\forall 1 \leq i \leq k$, $p_{j_i} \in G_i$; and (2) $\forall 1 \leq i \leq k-1$, $0 < t_{j_{i+1}} - t_{j_i} \leq \Delta t$.*

Note that the matching places $p_{j_1} p_{j_2} \cdots p_{j_k}$ in Definition 3 are not necessarily consecutive in o. For clarity, we also denote a G-sequence $G_1 \xrightarrow{\Delta t} G_2 \cdots \xrightarrow{\Delta t} G_k$ as $G_1 \rightarrow G_2 \cdots \rightarrow G_k$ when the context is clear. Now, we proceed to define *support* and *frequent G-sequence*.

DEFINITION 4 (SUPPORT). *Given a G-sequence T and a semantic trajectory database $\mathcal{D}$, the support of T in $\mathcal{D}$ is the number of trajectories in $\mathcal{D}$ that contain T,* i.e., $Sup(T) = |\{o | o \in \mathcal{D} \wedge T \sqsubseteq o\}|$.

DEFINITION 5 (FREQUENT G-SEQUENCE). *Given a threshold σ, a G-sequence T is frequent in database $\mathcal{D}$ if $Sup(T) \geq \sigma$.*

In the rest of the paper, we use *frequent G-sequence* and *sequential pattern* interchangeably. Note that, even for moderately sized $\mathcal{P}$ and $\mathcal{D}$, there can be numerous frequent G-sequences, as exemplified below.

EXAMPLE 1. *In Figure 1, as $G_1 \rightarrow G_2 \rightarrow G_3$ is frequent, the G-sequences derived by expanding any of G_1, G_2 or G_3 are also frequent. There are five places p_3, p_4, p_5, p_6 and p_{12} that can be used for expansion, and each can be added into any of G_1, G_2 and G_3 or none of them. Hence, we can derive $4^5 - 1$ expanded G-sequences, but none of them are interesting given the presence of $G_1 \rightarrow G_2 \rightarrow G_3$.*

Considering the daunting size and high redundancy of the complete set of frequent G-sequences, it is infeasible to report all of them. Instead, what we want is a set of frequent G-sequences that are *non-overlapping* and *fine-grained*.

DEFINITION 6 (OVERLAPPING RELATIONSHIP). *Given two G-sequences $T_1 = G_1 \xrightarrow{\Delta t} G_2 \xrightarrow{\Delta t} \cdots \xrightarrow{\Delta t} G_k$ and $T_2 = G'_1 \xrightarrow{\Delta t} G'_2 \xrightarrow{\Delta t} \cdots \xrightarrow{\Delta t} G'_l$, T_1 and T_2 are overlapping if (1) $k = l$; and (2) $\forall 1 \leq i \leq k, G_i \cap G'_i \neq \phi$.*

DEFINITION 7 (FINE-GRAINED PATTERN). *Given a frequent G-sequence $T = G_1 \xrightarrow{\Delta t} G_2 \xrightarrow{\Delta t} \cdots \xrightarrow{\Delta t} G_k$, T is fine-grained if: (1) the places in each G_i have the same semantic category; and (2) $\frac{1}{k} \sum_{i=1}^{k} Var(G_i) \leq \rho$, where $Var(G_i)$ is the spatial variance of the places in G_i and ρ is a variance threshold.*

Note the above three types of constraints for a fine-grained pattern: (1) the maximum transition time Δt ensures the temporal continuity; (2) the semantic constraint ensures the semantic consistency; and (3) the variance threshold ρ ensures the spatial compactness. Our goal is to find a set $\mathcal{R}$ of fine-grained patterns that are non-overlapping. Meanwhile, we want $\mathcal{R}$ to be as complete as possible. Since any two patterns in $\mathcal{R}$ must be non-overlapping, we define the coverage of $\mathcal{R}$ as follows.

DEFINITION 8 (COVERAGE). *Given $\mathcal{R}$, a set of fine-grained sequential patterns that are non-overlapping, the coverage of $\mathcal{R}$ is $Coverage(\mathcal{R}) = \sum_{T \in \mathcal{R}} Sup(T)$.*

We are now ready to formulate our problem. Given a support threshold σ, a temporal constraint Δt, and a spatial variance threshold ρ, find in database $\mathcal{D}$ a set $\mathcal{R}$ of non-overlapping fine-grained patterns such that the coverage of $\mathcal{R}$ is as high as possible.

2.2 Overview of Splitter

Even for discrete data, the sequential pattern mining problem has been shown to be NP-hard [14]. In our problem, the combinatorial nature of G-sequence makes this task even more challenging. Assume the places in $\mathcal{P}$ distribute in 2 categories A and B, and each category has 100 distinct places. To mine length-2 fine-grained patterns, there are $4 \times (2^{100} - 1) \times (2^{100} - 1)$ candidate G-sequences, where the term 4 is derived for the four cases $A \rightarrow A$, $A \rightarrow B$, $B \rightarrow A$ and $B \rightarrow B$, and the term $2^{100} - 1$ is derived as each group can contain any number of places from the same category. It is prohibitively expensive to enumerate all the possible grouping strategies and search for the best $\mathcal{R}$ by running classic sequential pattern mining algorithms [1, 10, 16] repeatedly. To avoid the costly enumerate-and-test process, we examine several characteristics of fine-grained patterns in the sequel, which underpin the design of SPLITTER.

With a time constraint Δt, let $T = G_1 \rightarrow G_2 \cdots \rightarrow G_k$ and $T' = G'_1 \rightarrow G'_2 \cdots \rightarrow G'_k$ be two G-sequences that satisfy $G_i \subseteq G'_i$ for $1 \leq i \leq k$. Obviously, it is ensured $Sup(T) \leq Sup(T')$, thus T' is frequent so long as T is frequent. We call T' a *parent* pattern of T. Now, suppose we want to find fine-grained patterns for *office* $\rightarrow$ *gym*, that is, visiting gyms after work. We can construct a group G_1 to include all the places in category *office*, and G_2 to include all the places in *gym*. Then $G_1 \rightarrow G_2$ will be the parent of any fine-grained patterns that we want to find. Meanwhile, if $G_1 \rightarrow G_2$ is infrequent in the database, it is ensured no fine-grained patterns can exist for *office* $\rightarrow$ *gym*.

The above inspires a two-step design for SPLITTER. In the first step, we group the places in $\mathcal{P}$ by category. The places in each group, while being spatially scattered, have the same category. By viewing each group as an item, we extract all the frequent sequential patterns from the database. These patterns, which we call coarse patterns, satisfy semantic and temporal constraints but may not be spatially compact. In the second step, we consider each coarse pattern independently and explore fine-grained patterns from it.

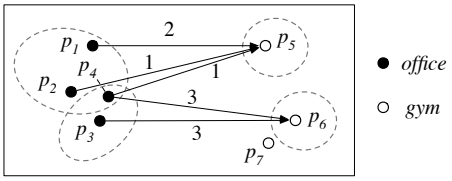


Figure 2: A coarse pattern *office* $\rightarrow$ *gym* ($\sigma = 3$ and $\Delta t = 60$). The number beside each line is the number of objects having the movement (assume all the 10 objects are distinct).

The discovery of coarse patterns significantly reduces the search space, because it not only filters all the infrequent combinations of semantic categories, but also allows us to focus on one coarse pattern at each time. But the challenge is, how to explore fine-grained patterns from a coarse one? Figure 2 shows an example. Assume $\sigma = 3$ and $\Delta t = 60$. With $G_1 = \{p_1, p_2, p_3, p_4\}$ and $G_2 = \{p_5, p_6, p_7\}$, we obtain a coarse pattern $G_1 \rightarrow G_2$. To explore

fine-grained patterns from it, an intuitive idea is to split both G_1 and G_2 into compact subgroups using spatial clustering. By splitting each group into 2 subgroups using K-Means, we will obtain the following subgroups: $\{p_1\}$, $\{p_2, p_3, p_4\}$, $\{p_5\}$ and $\{p_6, p_7\}$, then the pattern $\{p_2, p_3, p_4\} \rightarrow \{p_6, p_7\}$ appears (support = 6).

Although it seems effective at first glance, splitting each group independently is actually problematic under careful scrutiny. First, each subgroup may include useless places, like p_2 and p_7 in the pattern $\{p_2, p_3, p_4\} \rightarrow \{p_6, p_7\}$. Worse still, interesting patterns, like $\{p_1, p_2, p_4\} \rightarrow \{p_5\}$, may be lost. Second, it assigns each place to only one subgroup. In practice, however, people may move from one place, like p_4 in Figure 2, to different places. Hence, it is desirable to allow one place to belong to multiple subgroups, thereby contributing to different patterns.

To address the above problems, SPLITTER employs a more effective strategy to explore fine-grained patterns from a coarse one: it directly clusters the movements that match the coarse pattern, which we call *trajectory snippets*.

DEFINITION 9 (TRAJECTORY SNIPPET). *Given a length-k G-sequence $T = G_1 \xrightarrow{\Delta t} G_2 \cdots \xrightarrow{\Delta t} G_k$, and a trajectory $o = \langle (p_1, t_1), (p_2, t_2), \ldots, (p_l, t_l) \rangle$ satisfying $T \sqsubseteq o$, a place sequence $p_{j_1} p_{j_2} \cdots p_{j_k}$ in o is called a snippet for T if it satisfies: (1) $1 \leq j_1 < j_2 < \cdots < j_k \leq l$; (2) $\forall 1 \leq i \leq k$, $p_{j_i} \in G_i$; and (3) $\forall 1 \leq i \leq k-1$, $t_{j_{i+1}} - t_{j_i} \leq \Delta t$.*

Informally, in Figure 2, the snippets for the pattern *office* $\rightarrow$ *gym* are the lines with arrows. SPLITTER directly merges spatially close lines to form fine-grained patterns. The benefits are two-fold: (1) the snippets precisely record a coarse pattern's place-level occurrences in the trajectory database, and filters all irrelevant places (*e.g.*, p_7 in Figure 2); and (2) each snippet preserves the movements among places, thus snippet merging considers not only spatial proximity but also sequential information.

EXAMPLE 2. *In Figure 2, the snippets $p_1 \rightarrow p_5$, $p_2 \rightarrow p_5$, $p_4 \rightarrow p_5$ are similar as both their starting and ending places are spatially close. By merging these snippets, we obtain a fine-grained pattern $\{p_1, p_2, p_4\} \rightarrow \{p_5\}$. Similarly, by merging the snippets $p_3 \rightarrow p_6$ and $p_4 \rightarrow p_6$, we obtain the pattern $\{p_3, p_4\} \rightarrow \{p_6\}$.*

Two questions remain to be answered: (1) how to mine the coarse patterns and their snippets? and (2) how to effectively cluster the snippets given the fact that we do not know the correct number of clusters? In the next two sections, we elaborate the two steps of SPLITTER and answer these two questions.

3. MINING COARSE PATTERN SNIPPETS

Given the place set $\mathcal{P}$, we first group the places in $\mathcal{P}$ by category. Let $\{G_1, G_2, \ldots, G_d\}$ be the results such that the places in each G_i have the same category. By viewing each G_i as an *item*, we can transform a semantic trajectory to a timestamped item sequence. Consider the database in Figure 1. With G_1 = *office*, G_2 = *gym*, and G_3 = *restaurant*, Table 1 shows the transformed database.

Table 1: The transformed semantic trajectory database.

Object	Timestamped item sequence
o_1	$\langle (G_2, 0), (G_1, 10), (G_2, 30), (G_3, 40) \rangle$
o_2	$\langle (G_1, 0), (G_2, 30), (G_1, 360), (G_2, 400), (G_3, 420) \rangle$
o_3	$\langle (G_2, 0), (G_3, 30) \rangle$
o_4	$\langle (G_1, 0), (G_1, 120), (G_3, 140), (G_2, 150), (G_3, 180) \rangle$
o_5	$\langle (G_2, 50), (G_2, 80), (G_3, 120), (G_1, 210) \rangle$

After the transformation, it is natural to use some classic sequential pattern mining algorithms (*e.g.*, PrefixSpan) to extract all the coarse patterns. However, recall the temporal constraint Δt. A sequential pattern mining algorithm needs to be tailored to ensure the transition time between two consecutive groups is no more than Δt. We tailor PrefixSpan as it has proved to be one of the most efficient sequential pattern mining algorithms. The basic idea of PrefixSpan is to use short patterns as prefixes to project the database and progressively grow the short patterns by searching for local frequent items. For a short pattern β, the β-projected database $\mathcal{D}_\beta$ includes the postfixes from the sequences that contain β. Local frequent items in $\mathcal{D}_\beta$ are then identified and appended to β to form longer patterns. Such a process is repeated recursively until no more local frequent items exist. One can refer to [10] for more details. For the purpose of mining time-constrained sequential patterns, we revise the notions of *postfix* and *local frequent item* as follows.

DEFINITION 10 (POSTFIX). *Given a timestamped sequence $\alpha = \langle (G_1, t_1), (G_2, t_2), \ldots, (G_n, t_n) \rangle$, and an element G_m ($1 \leq m < n$) in α, the postfix of α* w.r.t. *G_m is $\langle (G_{m+1}, t_{m+1} - t_m), (G_{m+2}, t_{m+2} - t_m), \ldots, (G_n, t_n - t_m) \rangle$.*

DEFINITION 11 (LOCAL FREQUENT ITEM). *Given a timestamped sequence $\alpha = \langle (G_1, t_1), (G_2, t_2), \ldots, (G_n, t_n) \rangle$, an item G, and a time constraint Δt, the sequence α contains G if there exists an integer i ($1 \leq i \leq n$) such that $G_i = G$ and $t_i \leq \Delta t$. In a projected database, an item G is frequent if there are at least σ postfixes that contain G.*

Given a sequence α and a frequent item G, when creating G-projected database, the standard PrefixSpan procedure generates one postfix based on the first occurrence of G in α. This strategy, unfortunately, can miss time-constrained patterns in our problem.

EXAMPLE 3. *Let $\Delta t = 60$ and $\sigma = 3$. In the database shown in Table 1, item G_1 is frequent. The G_1-projected database generated by PrefixSpan is:*

(1) $o_1/G_1 = \langle (G_2, 20), (G_3, 30) \rangle$
(2) $o_2/G_1 = \langle (G_2, 30), (G_1, 360), (G_2, 400), (G_3, 420) \rangle$
(3) $o_4/G_1 = \langle (G_1, 120), (G_3, 140), (G_2, 150), (G_3, 180) \rangle$

The elements satisfying $t \leq 60$ are $(G_2, 20)$, $(G_3, 30)$ and $(G_2, 30)$. No local item is frequent, hence G_1 cannot be grown any more.

To overcome the above problem, we introduce a simple principle called *full projection*. Specifically, for a sequence α and a frequent item G, we generate a postfix for every occurrence of G in α.

EXAMPLE 4. *With full projection, G_1-projected database is:*

(1) $o_1/G_1 = \langle (G_2, 20), (G_3, 30) \rangle$
(2) $o_2/G_1 = \langle (G_2, 30), (G_1, 360), (G_2, 400), (G_3, 420) \rangle$
(3) $o_2/G_1 = \langle (G_2, 40), (G_3, 60) \rangle$
(4) $o_4/G_1 = \langle (G_1, 120), (G_3, 140), (G_2, 150), (G_3, 180) \rangle$
(5) $o_4/G_1 = \langle (G_3, 20), (G_2, 30), (G_3, 60) \rangle$

Items G_2 and G_3 are frequent and meanwhile satisfy the temporal constraint, thus longer patterns $G_1 \xrightarrow{60} G_2$ and $G_1 \xrightarrow{60} G_3$ are found in the projected database.

With full projection, the projected database includes all postfixes to avoid missing patterns under the time constraint. That said, multiple postfixes from the same trajectory can appear simultaneously. Hence, we should attach each postfix with its trajectory id to prevent one trajectory from being counted repeatedly.

Another intention of full projection is to collect all the distinct snippets for a coarse pattern. Continuing the above example, o_2 contains two different snippets (p_5p_7 and p_2p_7) for the coarse pattern $G_1 \rightarrow G_2$, these two snippets correspond to the two projections of o_2/G_1. In order to extract snippets along with a coarse pattern, we maintain an additional snippet field in each projection and grow the snippet along with the pattern, as exemplified below.

EXAMPLE 5. *In Table 1, G_1 is frequent and $G_1 \sqsubseteq o_1$. In the projection o_1/G_1, we additionally store the snippet p_1, which records G_1's occurrence in o_1. Note that, by preserving the original place id information in the transformed database, p_1 is attached with the element $(G_1, 10)$ and thus readily available. As G_1 grows to $G_1 \rightarrow G_2$, the snippet field p_1 is also extended to p_1p_7. When reporting the pattern $G_1 \rightarrow G_2$, the snippets in its projected database are aggregated and reported. Using the pseudo projection technique [10], maintaining the snippet field incurs little overhead.*

Algorithm 1 sketches our algorithm for mining coarse patterns and their snippets. As shown, given the transformed database $\mathcal{D}$ and threshold σ, we first extract all the single frequent items in $\mathcal{D}$ (note that we do not need to check the constraint Δt when searching for single frequent items in $\mathcal{D}$). Then for each frequent item i, we build the i-projected database using full projection. Once the projected database is built, we call PrefixSpan to recursively output frequent patterns. The PrefixSpan procedure is similar to the standard version in [10], except that the time constraint Δt is checked when searching for local frequent items. Moreover, full projection is adopted, and each projected postfix maintains a snippet field that is grown along with the pattern.

The output of Algorithm 1 is a set of coarse patterns. Each coarse pattern T is attached with a snippet set $\mathcal{S} = \{(s, V, w) | T \sqsubseteq s\}$. In the triple (s, V_s, w_s), s is a snippet for T, V_s is the set of objects containing s (we call them *visitors* of s), and $w = |V|$. For example, in Figure 1, $p_1 \rightarrow p_7$ is a snippet for the coarse pattern *office* $\rightarrow$ *gym*, and o_1 is the only object containing this snippet, so the respective triple is $(p_1 \rightarrow p_7, \{o_1\}, 1)$.

Algorithm 1: Mining coarse patterns and snippets.

Input: support threshold σ, temporal constraint Δt, transformed semantic trajectory database $\mathcal{D}$

Procedure *InitialProjection($\mathcal{D}$, σ, Δt)*
- $\mathcal{L} \leftarrow$ frequent items in $\mathcal{D}$;
- **foreach** *item i in $\mathcal{L}$* **do**
  - $S \leftarrow \phi$;
  - **foreach** *trajectory o in $\mathcal{D}$* **do**
    - $R \leftarrow$ postfixes for all occurrences of i in o;
    - $S \leftarrow S \cup R$;
  - Output i and its snippets;
  - PrefixSpan(i, 1, S, Δt);

Function *PrefixSpan(α, l, $S|_\alpha$, Δt)*
- $\mathcal{L} \leftarrow$ frequent items in $S|_\alpha$ meeting time constraint Δt;
- **foreach** *item i in $\mathcal{L}$* **do**
  - $\alpha' \leftarrow$ append i to α;
  - Build $S|_{\alpha'}$ using full projection and grow the snippet field in each projection;
  - Output α' and its snippets;
  - PrefixSpan(α', $l+1$, $S|_{\alpha'}$, Δt);

4. FINDING FINE-GRAINED PATTERNS

For each coarse pattern T, we now have a set $\mathcal{S}$ of its snippets that are spatially scattered. Next task is to explore fine-grained pattens for T by merging close snippets in $\mathcal{S}$. To this end, we transform each snippet into a weighted point in a higher-dimensional Euclidean space. Given a length-k snippet $s = p_1p_2 \cdots p_k$, we transform it into a $2k$-dimensional point $\mathbf{x}$ by assembling the coordinates of p_i ($1 \leq i \leq k$). Meanwhile, we attach the weight of s, namely the number of visitors, to $\mathbf{x}$. The key observation is that, if a fine-grained pattern exists, its snippets will form a *dense* and *compact* cluster in the transformed space. The cluster needs to be *dense* in order to meet support threshold σ, and be *compact* to meet the variance threshold ρ. Hence, the problem is reduced to detecting such dense and compact clusters in the transformed space.

One may suggest some classic clustering algorithms such as K-means, GMM and DBSCAN to detect clusters. However, the effectiveness of these algorithms relies on the prior knowledge about the underlying data distribution and/or correctly guessing the number of clusters. For a coarse pattern T, snippet distribution in the transformed space can be really complex, rendering these algorithms intractable. To overcome this challenge, we propose an adaption of the mean shift algorithm [3], called *weighted snippet shift*.

Mean shift is a non-parametric method widely used in the computer vision community. It has several nice properties for our task. First, it does not assume any prior knowledge about the number of clusters or data distribution. Thus it can effectively discover arbitrarily shaped clusters in a complex data space. Second, it has only one parameter, namely the bandwidth, which has a physical meaning as the scale of observation. The tuning of bandwidth h can effectively control the granularity of observation. This property makes mean shift well suited for finding fine-grained patterns in a top-down manner: Starting with a large h, if some snippet clusters are dense and compact, we grab them out as fine-grained patterns. For the remaining snippets, we reduce h to observe at a finer granularity, such a process continues until no more clusters can be dense enough to exceed σ.

In the following, we first describe the weighted snippet shift algorithm. Then we introduce the details of the top-down pattern discovery process. Finally, we discuss the algorithm efficiency.

4.1 Pattern Splitting via Weighted Snippet Shift

We first introduce standard mean shift, then describe how we adapt it to cluster snippets and split a coarse pattern.

Standard Mean Shift. Mean shift is essentially a kernel-based *mode* (*i.e.*, local maxima of density) seeking method. While various kernel functions can be used for mean shift, we choose the Epanechnikov kernel due to its simplicity and optimality in terms of bias-variance tradeoff. The Epanechnikov kernel is defined as

$$K(\mathbf{x}) = \begin{cases} c(1 - \|\mathbf{x}\|^2) & if \|\mathbf{x}\| < 1 \\ 0 & otherwise, \end{cases}$$

here c is a constant to ensure $\int K(\mathbf{x})\, \mathrm{d}\mathbf{x} = 1$.

Informally, for a d-dimensional point, mean shift finds its mode by iteratively shifting a radius-h window towards a local density maxima. The window is called the kernel window and the radius is called the bandwidth. In each iteration, let $\mathbf{y}^{(k)}$ be the center of current window, and $\mathcal{N} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ be the m data points inside the window, then the kernel window is shifted towards the maximum increase of density for $\mathbf{y}^{(k)}$. Using the Epanechnikov kernel, the mean shift vector for $\mathbf{y}^{(k)}$ is

$$\mathbf{m}(\mathbf{y}^{(k)}) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}_i - \mathbf{y}^{(k)}. \quad (1)$$

Then $\mathbf{y}^{(k)}$ is shifted by $\mathbf{m}(\mathbf{y}^{(k)})$, resulting in a new kernel window located at the mean of $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$, namely

$$\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)} + \mathbf{m}(\mathbf{y}^{(k)}) = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i. \quad (2)$$

Figure 3 illustrates the mean shift operation. Given a point $\mathbf{x}$, mean shift starts with an initial window center $\mathbf{y}^{(0)} = \mathbf{x}$, and iteratively shifts the window according to Equation 2. The sequence $\{\mathbf{y}^{(k)}\}$ will converge to the mode $\mathbf{x}$ belongs to. Then the data points that converge to the same mode are grouped as one cluster.

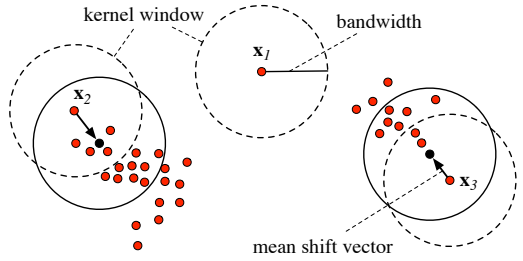


Figure 3: Mean shift with the Epanechnikov kernel.

Weighted Snippet Shift. We have transformed the coarse pattern snippets into weighted points and the weight denotes the number of visitors. Intuitively, a large-weight snippet is a popular place sequence, and should have a higher chance to attract the mode towards it. Below, we adapt the standard mean shift procedure to incorporate snippet weight and prove its convergence. Let $\mathcal{X} = \{(\mathbf{x}_1, w_1), (\mathbf{x}_2, w_2), \ldots, (\mathbf{x}_n, w_n)\}$ be those weighted points in the d-dimensional space. Using the Epanechnikov kernel, the density estimator at any location $\mathbf{y}$ is

$$\hat{f}(\mathbf{y}) = \frac{1}{h^d w}\sum_{i=1}^{n} w_i K(\frac{\mathbf{y}-\mathbf{x}_i}{h}) = \frac{c}{h^{d+2}w}\sum_{\mathbf{x}_i\in\mathcal{N}} w_i(h^2 - \|\mathbf{y}-\mathbf{x}_i\|^2).$$

where $w = \sum_{i=1}^{n} w_i$, and $\mathcal{N} \subseteq \mathcal{X}$ are points inside the radius-h window centered at $\mathbf{y}$ (*i.e.*, the distance to $\mathbf{y}$ is smaller than h). The density gradient at $\mathbf{y}$ is then given by

$$\begin{aligned}\nabla\hat{f}(\mathbf{y}) =& \frac{2c}{h^{d+2}w}\sum_{\mathbf{x}_i\in\mathcal{N}} w_i(\mathbf{x}_i - \mathbf{y}) \\ =& \frac{2c}{h^{d+2}w}\left(\sum_{\mathbf{x}_i\in\mathcal{N}} w_i\right)\left(\frac{\sum_{\mathbf{x}_i\in\mathcal{N}} w_i\mathbf{x}_i}{\sum_{\mathbf{x}_i\in\mathcal{N}} w_i} - \mathbf{y}\right).\end{aligned}$$

To perform gradient ascent, the weighted shift vector becomes

$$\mathbf{m}(\mathbf{y}) = \frac{\sum_{\mathbf{x}_i\in\mathcal{N}} w_i\mathbf{x}_i}{\sum_{\mathbf{x}_i\in\mathcal{N}} w_i} - \mathbf{y}, \quad (3)$$

which is simply the difference between the weighted mean of points in $\mathcal{N}$, and $\mathbf{y}$, the current window center. One can observe that the shifting vector is proportional to the density gradient $\nabla\hat{f}(\mathbf{y})$, it thus moves $\mathbf{y}$ towards the densest location in the current window. The snippet weight is playing an important role in shifting: if a snippet has a large number of visitors, then it is more likely to attract the new center towards itself.

We sketch the weighted snippet shift procedure in Algorithm 2. As shown, starting from the initial point $\mathbf{x}$, the procedure first uses $\mathbf{y}^{(0)} = \mathbf{x}$ as the window center and retrieves points $\mathcal{N}_k$ inside the radius-h window. It then moves the window center to the weighted mean of $\mathcal{N}_k$. The procedure is repeated until the window center becomes (approximately) stationary.

Algorithm 2: Weighted snippet shift.

Input: a set of $\mathcal{S}$ of weighted snippets, bandwidth h
foreach $\mathbf{x} \in \mathcal{S}$ **do**
  $k \leftarrow 0, \mathbf{y}^{(0)} \leftarrow \mathbf{x}$;
  while True do
    $\mathcal{N}_k \leftarrow \{\mathbf{x}\}_{i=1}^{m}$ *s.t.* $\forall i, \|\mathbf{y}^{(k)} - \mathbf{x}_i\| \le h$;
    $\mathbf{y}^{(k+1)} \leftarrow (\sum_{i=1}^{m} w_i\mathbf{x}_i)/(\sum_{i=1}^{m} w_i)$;
    if $\|\mathbf{y}^{(k+1)} - \mathbf{y}^{(k)}\| \le \epsilon$ **then**
      return $(\mathbf{x}, \mathbf{y}^{(k+1)})$;
    $k \leftarrow k + 1$;

THEOREM 1. *Starting from any snippet $\mathbf{x} \in \mathcal{S}$, the weighted snippet shift procedure will converge.*

PROOF. See Appendix. □

Pattern Splitting. For a coarse pattern T and its snippet set $\mathcal{S}$, Algorithm 2 shifts each snippet in $\mathcal{S}$ to a stationary point (mode). The snippets shifted to the same mode are then grouped together. In this way, $\mathcal{S}$ is split into a number of clusters $\Gamma_{\mathcal{S}} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$. A cluster $\mathcal{S}_i \in \Gamma_{\mathcal{S}}$ forms a fine-grained pattern if it satisfies the following two conditions: (1) $\mathcal{S}_i$ is frequent, namely $|\cup_{s\in\mathcal{S}_i}| \ge \sigma$; and (2) the G-sequence derived from $\mathcal{S}_i$ has a spatial variance smaller than ρ (Definition 7).

4.2 Top-Down Pattern Discovery

To split a coarse pattern, it is hard to pre-specify an optimal bandwidth h: if h is small, we may obtain many small snippet clusters that cannot exceed the support threshold σ. On the other hand, if h is too large, we may obtain some large clusters that cannot satisfy the variance constraint ρ.

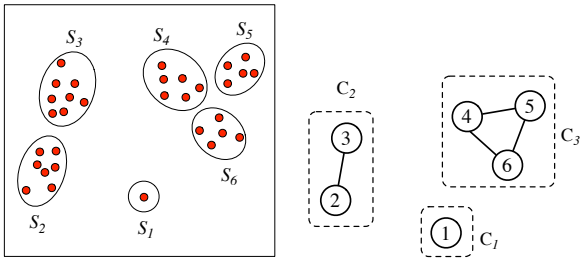
To avoid guessing a fixed bandwidth beforehand, we develop a top-down pattern discovery process: We start snippet clustering with an initial bandwidth h that is large. From the result snippet clusters, we grab out the ones forming fine-grained patterns. Then, we dampen h and zoom into the remaining snippets to find additional patterns. This process continues until no more patterns exist, and fine-grained patterns are reported on-the-fly.

However, after each round of clustering, the set of remaining snippets could still be large, it is costly to repeat Algorithm 2 on a large set of snippets. To speed up the top-down discovery process, we design a divide-and-conquer strategy. The key idea is to organize the remaining clusters into several communities that are mutually faraway. We prove that the further clustering with a smaller h can be carried out in each community independently. Better still, small communities that cannot exceed the support threshold are pruned early on. Let $\Gamma_{\mathcal{S}} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n\}$ be a set of snippet clusters. Below, we introduce the notion of γ-*community*.

DEFINITION 12 (γ-ADJACENCY). *Given two clusters $\mathcal{S}_i$ and $\mathcal{S}_j$ ($i \ne j$), they are γ-adjacent if $\exists\mathbf{x} \in \mathcal{S}_i, \exists\mathbf{y} \in \mathcal{S}_j, \|\mathbf{x}-\mathbf{y}\| \le \gamma$.*

DEFINITION 13 (γ-ADJACENT GRAPH). *Given a distance γ, the γ-adjacent graph G_γ constructed from $\Gamma_{\mathcal{S}}$ is as follows: each node of G_γ is a cluster $\mathcal{S}_i \in \Gamma_{\mathcal{S}}$, and there exists an edge between two nodes if the corresponding clusters are γ-adjacent.*

Based on Definition 13, we define a γ-*community* as a connected component in the γ-adjacent graph. Figure 4 shows a concrete example. Given 6 snippet clusters $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_6$, their γ-adjacent graph is shown in Figure 4(b). There are 3 connected components in the graph, corresponding to the γ-communities C_1, C_2 and C_3.



(a) Snippet clusters. (b) γ-adjacent graph.

Figure 4: Illustration of γ-community.

Let $\Omega_C = \{C_1, C_2, \ldots, C_t\}$ be the γ-communities obtained from $\Gamma_\mathcal{S}$, the distance between any two γ-communities is at least γ. Recall Algorithm 2, which iteratively shifts a radius-h window until convergence. Below, we prove that if $h \leq \gamma/\sqrt{2}$, points from different γ-communities will never converge to the same mode.

LEMMA 1. *When running Algorithm 2 from any point $\mathbf{x}$, let $\mathbf{y}^{(k)}$ be the center of the window after k iterations. With $h \leq \gamma/\sqrt{2}$, if the window centered at $\mathbf{y}^{(k)}$ includes only points from C_i, then the window at $\mathbf{y}^{(k+1)}$ also includes only points from C_i.*

PROOF. See Appendix. □

LEMMA 2. *Given a point $\mathbf{x} \in C_i$, suppose $\mathbf{x}$ converges to $\mathbf{y}_\mathbf{x}$ by running Algorithm 2 with bandwidth h. If $h \leq \gamma/\sqrt{2}$, the window centered at $\mathbf{y}_\mathbf{x}$ includes only points from C_i.*

PROOF. By the definition of γ-community, the radius-h window centered $\mathbf{y}^{(0)} = \mathbf{x}$ includes only points from C_i because $h \leq \gamma/\sqrt{2} < \gamma$. By Lemma 1, if the window at $\mathbf{y}^{(k)}$ includes only points from C_i, so does the window at $\mathbf{y}^{(k+1)}$. As $\mathbf{y}_\mathbf{x}$ is derived from the sequence $\{\mathbf{y}^{(k)}\}$, the lemma holds immediately. □

Given $\Omega_C = \{C_1, C_2, \ldots, C_t\}$, Lemma 2 amounts to saying that an arbitrary point $\mathbf{x}$ in any C_i converges to a kernel window that only contains points in C_i. We proceed to show when $\gamma \geq \sqrt{2}h$, this condition suffices to guarantee points from two different γ-communities will not converge to the same mode.

THEOREM 2. *Given two γ-communities C_i and C_j ($i \neq j$), and two points $\mathbf{x} \in C_i$ and $\mathbf{x}' \in C_j$. With $h \leq \gamma/\sqrt{2}$, suppose $\mathbf{x}$ converges to $\mathbf{y}$ and $\mathbf{x}'$ converges to $\mathbf{y}'$, it is ensured $\mathbf{y} \neq \mathbf{y}'$.*

PROOF. See Appendix. □

Theorem 2 implies a nice property for γ-community. Suppose we run Algorithm 2 on $\Omega_C = \{C_1, C_2, \ldots, C_t\}$ with a bandwidth $h \leq \gamma/\sqrt{2}$, each community C_i is totally independent of others. Hence, the task of clustering Ω_C can be broken into t smaller tasks. Moreover, small communities that cannot exceed σ can be safely pruned, as suggested in Theorem 3.

THEOREM 3. *For $C_i \in \Omega_C$, let V_s be the set of visitors for a snippet $s \in C_i$. Given the support threshold σ, if $|\cup_{s \in C_i} V_s| < \sigma$, then C_i cannot generate any fine-grained patterns.*

PROOF. By Theorem 2, any snippet cluster $\mathcal{S}$ generated from C_i satisfies $|\cup_{s \in \mathcal{S}} V_s| \leq |\cup_{s \in C_i} V_s| < \sigma$. The correctness of Theorem 3 then becomes immediate. □

Algorithm 3 presents the top-down pattern discovery process. Let $\mathcal{S}$ be the snippet set of a coarse pattern, to discover fine-grained patterns from $\mathcal{S}$, we need to specify two parameters: an initial bandwidth h_0 and a dampening factor τ ($0 < \tau < 1$). The top-down discovery process is recursive. Given a snippet set $\mathcal{S}$ and a bandwidth h, it first clusters $\mathcal{S}$ into $\Gamma_\mathcal{S}$ using weighted snippet shift. If fine-grained patterns exist among $\Gamma_\mathcal{S}$ (by aggregating the snippets in the cluster and checking support and variance), we report them and remove them from $\Gamma_\mathcal{S}$. For the remaining clusters, we organize them into several γ-communities with $\gamma = \sqrt{2}\tau h$. γ is set to $\sqrt{2}\tau h$ because it ensures each community is independent when performing snippet clustering with bandwidth τh. By Theorem 3, those small communities that cannot generate frequent patterns are pruned, then the remaining communities are further clustered with bandwidth τh to find additional fine-grained patterns.

Algorithm 3: Top-down pattern discovery.

Input: snippet set $\mathcal{S}$, support threshold σ, variance threshold ρ, bandwidth h, dampening factor τ

```
Procedure SplitPattern(S, σ, ρ, h, τ)
    Γ_S ← cluster S via weighted snippet shift with h;
    foreach S_i ∈ Γ_S do
        if Sup(S_i) ≥ σ and Var(S_i) ≤ ρ then
            Report S_i as a fine-grained pattern;
            Remove S_i from Γ_S;
    γ ← √2τh ;
    Ω_C ← γ-communities constructed from Γ_S;
    foreach C_i ∈ C do
        if Sup(C_i) ≥ σ then
            SplitPattern(C_i, σ, ρ, τh, τ);
```

The γ-communities can be efficiently constructed as a byproduct of weighted snippet shift. The key to constructing γ-communities is to find the γ-neighbors for each snippet $s \in \mathcal{S}$. Recall the initial step of Algorithm 2, which launches a range query to find h-neighbors for each snippet. We can simply modify the initial step by launching a range query with radius $= \max\{h, \sqrt{2}\tau h\}$. The query results can generate two lists: one is the h-neighbors to allow Algorithm 2 to proceed, and the other is the γ-neighbors as a byproduct to enable γ-community construction.

4.3 Discussion

In the top-down pattern discovery process, weighted snippet shift (Algorithm 2) is called multiple times. Given n d-dimensional points, the complexity of Algorithm 2 is $O(kdn^2)$, where k is the average number of iterations before convergence. The running time is mainly affected by n, but n will not be large in our problem. The reasons are two-fold: (1) The snippet set $\mathcal{S}$ includes *distinct* snippets extracted for a coarse pattern. The cardinality of $\mathcal{S}$ is expected to be much smaller than the database size. (2) As the top-down process proceeds, a snippet set is recursively broken into smaller and smaller communities.

5. EXPERIMENTS

In this section, we evaluate the empirical performance of SPLITTER. All algorithms were implemented in JAVA and the experiments were conducted on a computer with Intel Core i7 2.4Ghz CPU and 8GB memory.

5.1 Experimental Setup

Data Sets. Our experiments are based on both real and synthetic semantic trajectory data sets. The real data set, referred to as 4SQ,

is collected from Foursquare. As aforementioned, the check-in sequence of each Foursquare user is essentially a low-sampling semantic trajectory. Our 4SQ data set consists of the semantic trajectories of 14,909 users living in New York. There are totally 48,564 distinct places in the data set, distributed in a $0.5^\circ \times 0.5^\circ$ space and 15 categories. The average length of each trajectory, *i.e.*, number of check-ins, is 20. For most trajectories in 4SQ, some parts in a trajectory are dense while the rest are sparse. Note that this fact does not affect the applicability of the proposed problem. With the time constraint Δt, an effective method should automatically detect fine-grained patterns from the dense parts.

We generate two synthetic data sets using Brinkhoff's network-based generator of moving objects[1], with San Francisco's map as the underlying network. The first synthetic data set, called S1K, consists of 10^3 trajectories and 3.0×10^4 distinct places. We first generate 10^3 trajectories for 100 timestamps and randomly choose one category for each place from 20 pre-defined categories. Then we define 50 length-2 fine-grained patterns and 10 length-3 ones. For each pattern, we insert a supporting snippet to every trajectory with probability 0.05. The second synthetic data set S10K consists of 10^4 trajectories, recorded for 100 timestamps. There are about 3.2×10^5 distinct places in S10K. Similarly, 10^4 random trajectories are first generated and then mixed with pre-defined patterns.

Compared Methods. To the best of our knowledge, no existing methods can be directly used to mine fine-grained sequential patterns in semantic trajectories. However, some existing techniques can be extended for our problem, we describe two compared methods as follows.

The first method, referred to as GRID, is adapted from the algorithm for mining sequential patterns in GPS trajectories [5]. The key is to identify a set of disjoint Region-of-Interest (RoI) in the space. Each RoI is a dense rectangle-shaped region. To find such RoIs, the space is first partitioned into numerous small grids, then each dense grid is gradually merged with its neighboring grids until the density of the merged region is below a threshold δ. Once the RoIs are identified, the places inside the same RoI are grouped together. To ensure semantic consistency, GRID identifies the RoIs for each category independently. Based on such RoIs, GRID transforms the database (*i.e.*, mapping place ids to RoI ids) and runs Algorithm 1 to select out fine-grained patterns. Since δ greatly affects mining effectiveness, GRID repeats the above process t times with different δ, and reports the best performance.

The second method, referred to as HC, integrates Algorithm 1 with hierarchical spatial clustering. First, HC groups the places in $\mathcal{P}$ by category, and uses K-Means to cluster the places in each category into K groups based on spatial proximity. With these groups, HC transforms the database by mapping place ids to group ids, and runs Algorithm 1. Among the output patterns, HC focuses on the coarse ones that do not satisfy the spatial constraint ρ. Specifically, HC breaks each group in a coarse pattern into K smaller subgroups. With the new grouping scheme, HC runs Algorithm 1 again. Such a process continues until all the output patterns become fine-grained. To be efficient, after each call of Algorithm 1, HC prunes the groups that do not appear in any output patterns, and thus the trajectory database keeps shrinking.

5.2 Illustrating Cases

We first demonstrate how SPLITTER works by mining length-1 patterns on 4SQ. Although length-1 patterns actually do not contain sequential information, they are easier to visualize as each length-1 snippet is simply a place. Figure 5 shows the process of mining length-1 patterns in category *transportation*, with $\sigma = 150$ and $\rho = 2 \cdot 10^{-4}$. SPLITTER first groups all the transportation places and finds a coarse pattern (Figure 5(a)). By splitting the pattern with an initial bandwidth $h_0 = 0.02$, C_1, C_2 and C_3 are reported as fine-grained patterns (Figure 5(b)). Interestingly, C_1, C_2 and C_3 correspond to three airport areas in New York. R_1, which corresponds to transportation places in Manhattan, is a large cluster meeting the support threshold σ but not the variance threshold ρ. After grabbing out C_1, C_2, C_3 and pruning small communities, SPLITTER proceeds with bandwidth $h = 0.016$ ($\tau = 0.8$), and finds two additional patterns C_4 and C_5, which are the transportation places around the Wall Street and Midtown Manhattan, respectively. Finally, SPLITTER finds the 8 fine-grained patterns $C_1, C_2, \ldots, C_8$ shown in Figure 5(d).

With $\sigma = 150$ and $\Delta t = 120$ minutes, we find 73 length-2 coarse patterns and 6 length-3 ones on 4SQ. Table 2 shows the coarse patterns with the largest support. We can see these patterns are quite sensible. For example, "Shop $\rightarrow$ Food $\rightarrow$ Shop" implies many people first went shopping, then after having lunch/dinner, they returned to continue. "Professional $\rightarrow$ Food $\rightarrow$ Nightlife Spot" implies a common behavior that people had dinner after work, and then went to nightlife spots to relax. We now refine the most frequent length-2 pattern "Shop $\rightarrow$ Food" with $h_0 = 0.02$, $\tau = 0.8$ and $\rho = 2 \cdot 10^{-4}$. SPLITTER discovers 6 fine-grained sequential patterns from this coarse pattern. Figure 6 depicts 3 representative ones. The patterns $P_1 = S_1 \rightarrow R_1$ and $P_2 = S_2 \rightarrow R_2$ imply many people just ate at nearby restaurants after shopping, but $P_3 = S_3 \rightarrow R_3$ shows there are also people willing to eat at faraway restaurants after shopping (S_3 is a group of shops in Manhattan, and R_3 is a group of restaurants near the Prospect Park).

Table 2: Top five length-2 and length-3 coarse patterns.

	Pattern	Sup
length=2	Shop $\rightarrow$ Food	1819
	Food $\rightarrow$ Shop	1464
	Professional $\rightarrow$ Nightlife Spot	1121
	Outdoor $\rightarrow$ Food	947
	Residence $\rightarrow$ College & University	647
length=3	Shop $\rightarrow$ Food $\rightarrow$ Shop	262
	Professional $\rightarrow$ Food $\rightarrow$ Nightlife Spot	240
	Entertainment $\rightarrow$ Food $\rightarrow$ Shop	178
	Transportation $\rightarrow$ Shop $\rightarrow$ Shop	174
	Residence $\rightarrow$ Outdoor $\rightarrow$ Food	163

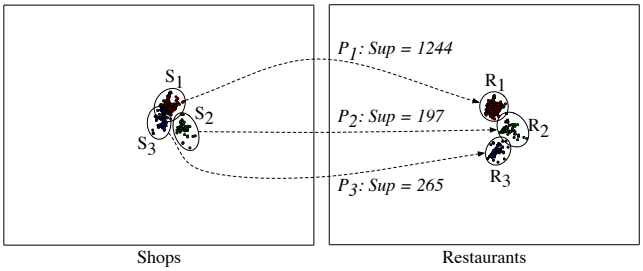
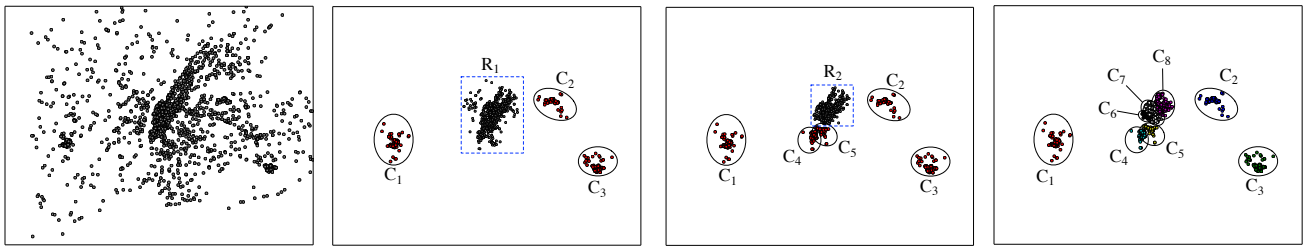


Figure 6: Example length-2 fine-grained patterns.

5.3 Effectiveness Study

In this subsection, we compare the effectiveness of SPLITTER, GRID and HC in terms of **coverage** and **number of patterns**. We only mine patterns with length no less than 2. On 4SQ, we set the default parameters of fine-grained pattern as $\sigma = 150$, $\rho = 2 \cdot 10^{-4}$ and $\Delta t = 120$ minutes. For SPLITTER, we set its default parameters as $h_0 = 0.02$ and $\tau = 0.8$ because such a setting achieves

[1] http://iapg.jade-hs.de/personen/brinkhoff/generator/



(a) Initial coarse pattern. (b) Split with $h = 0.02$. (c) Split with $h = 0.016$. (d) Fine-grained patterns.

Figure 5: Illustration of mining fine-grained patterns in category *transportation* on 4SQ. Each dot represents a *transportation* place.

a good tradeoff between effectiveness and efficiency. We set the repeating number $t = 10$ for GRID, and $K = 2$ for HC.

Varying σ. In the first set of experiments, we examine the effect of σ on the performance of the three methods. As shown in Figure 7(a) and 7(b), the coverage and number of patterns of all the three methods decrease with σ. This is because when σ is large, more places need to be included in a group, which in turn can violate the spatial and temporal constraints. Comparing the performance of the three methods, SPLITTER consistently outperforms the other two in terms of both coverage and number of patterns. We have also examined the lengths of result patterns, and found that GRID and HC can only find length-2 patterns in all settings. In contrast, SPLITTER can discover length-3 patterns if σ is not too large. When $\sigma = 50, 100, 150$, the number of length-3 patterns discovered by SPLITTER is 8, 4, and 1, respectively.

Varying Δt. Figure 7(c) and 7(d) show the performance of the three methods as Δt varies. The coverage and number of patterns of these methods increase roughly linearly with Δt. This is intuitive as a larger Δt imposes a weaker constraint on the transition time between consecutive groups, thus G-sequences tend to gain more support in the database. We also found that when Δt is small, the groups in each pattern are spatially close, but as Δt becomes larger, patterns containing faraway groups gradually appear. The reason behind is that, when Δt is small, long-range movements in trajectories are eliminated. Under different Δt, SPLITTER still significantly outperforms the compared methods.

Varying ρ. Figure 7(e) and 7(f) show the performance of the three methods as ρ varies from $2 \cdot 10^{-4}$ to 10^{-3} (when $\rho > 10^{-3}$, the result patterns become not compact already). Again, SPLITTER is much more effective than GRID and HC under different ρ. Another interesting finding is that, there are fewer fine-grained patterns than coarse ones for all methods, especially when ρ is small. For example, when $\rho = 2 \cdot 10^{-4}$, there are 79 coarse patterns, but even SPLITTER only reports 34 fine-grained ones. We checked the coarse patterns that have been eliminated, and found that the movements in most of them are quite scattered. They do not contain any sub-patterns that are frequent and compact, thus eliminated by the spatial constraint ρ.

Effects of h_0 and τ. Finally, we examine the effects of h_0 and τ on the performance of SPLITTER. Figure 7(g) and 7(h) shows the coverage of SPLITTER as h_0 and τ increase (we omit their effects on the number of patterns because the trend is similar). As h_0 increases, the performance of SPLITTER first increases dramatically and then becomes steady. This phenomenon is expected. During the top-down splitting process, if h_0 is too small, we will generate many small snippet clusters even in the first round of splitting. Such clusters do not meet the support threshold σ, nor will clusters further generated with dampened h. However, so long as h_0 is large enough (0.002), this problem does not exist any more, and the performance of SPLITTER becomes insensitive to h_0. Figure 7(h) shows that the performance of SPLITTER increases with τ. Intuitively, if τ is too small, many appropriate bandwidths may be skipped during the top-down discovery of fine-grained patterns. Hence, the parameter τ should not be set too small in practice.

Summary of effectiveness study. The above observations demonstrate the effectiveness of SPLITTER. It always outperforms GRID and HC significantly under different settings. The experimental results and findings are similar on the two synthetic data sets, we omit them to save space.

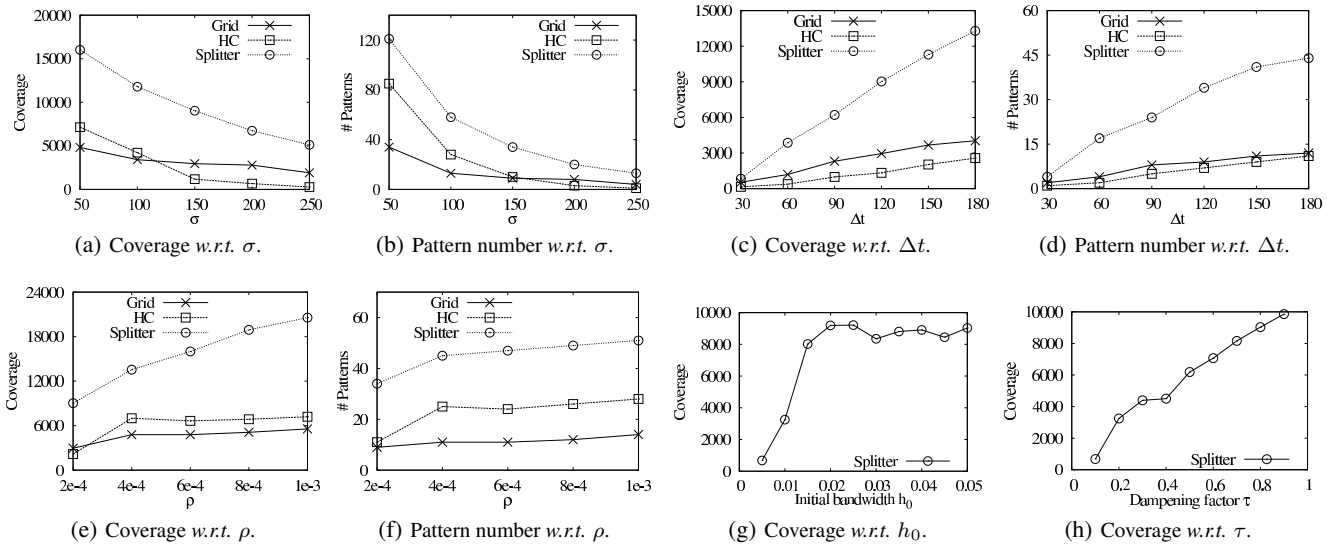
5.4 Efficiency Study

In this subsection, we compare the efficiency of SPLITTER, GRID and HC. Unless otherwise stated, all parameters are set to their default values as in Section 5.3.

Efficiency comparison on 4SQ. Figure 8 reports the running time of the three methods on 4SQ. Since SPLITTER is a two-step method, we break its total running time into two parts: S-Coarse is the running time for mining coarse patterns, and S-Refine is the running time for discovering fine-grained patterns from the coarse ones. We compare the running time of SPLITTER, GRID and HC when σ and Δt vary. The effect of ρ is omitted because the running time of all methods changes slightly when ρ varies from $2 \cdot 10^{-4}$ to 10^{-3}.

As shown in Figure 8(a), the running time of all methods decreases with σ. GRID is much slower than HC and SPLITTER because it needs to run Algorithm 1 with 10 different density threshold δ. If we run GRID with only one fixed δ, the effectiveness of GRID drops dramatically. HC also needs to run Algorithm 1 multiple times, but it is quite efficient as it can prune the clusters that do not appear in any frequent patterns. Comparing the performance of SPLITTER and HC, SPLITTER consistently outperforms HC, but the difference becomes less obvious when σ is large. This is because when σ is large, only few clusters derived by HC can form frequent patterns while most clusters are pruned, making HC very ineffective (see Figure 7(b)). Figure 8(b) shows that the running time of all methods increases with Δt. SPLITTER is faster than HC when Δt is small. However, its time cost increases more quickly with Δt, and finally becomes larger than HC when $\Delta t \geq 150$. The reason is that when Δt is large, much more coarse patterns are discovered, which makes the pattern splitting step more costly. Note that the efficiency of HC comes with the price of being ineffective when Δt is large (see Figure 7(d)).

Effects of h_0 and τ. Figure 9 shows the effects of parameters h_0 and τ on the performance of SPLITTER. As shown, when h_0 and τ increase, the running time of SPLITTER's first step does not change, while the running time of second step keeps increasing. The increase with h_0 is quite rapid, as a large h makes SPLITTER execute snippet clustering more times, and makes it harder to orga-



(a) Coverage *w.r.t.* σ. (b) Pattern number *w.r.t.* σ. (c) Coverage *w.r.t.* Δt. (d) Pattern number *w.r.t.* Δt.

(e) Coverage *w.r.t.* ρ. (f) Pattern number *w.r.t.* ρ. (g) Coverage *w.r.t.* h_0. (h) Coverage *w.r.t.* τ.

Figure 7: Effectiveness comparison of Splitter, Grid, and HC on 4SQ.

nize snippets into γ-communities. However, as mentioned earlier, a very large h_0 brings no extra benefit to the effectiveness of SPLITTER. Hence, h_0 does not need to be too large in practice.

Effect of the speedup strategy. In this set of experiments, we study the effectiveness of the divide-and-conquer strategy for SPLITTER. For comparison, we implemented a naïve version of top-down splitting without the speedup strategy, referred to as Naïve-Refine. As shown in Figure 10, under various settings of σ and Δt, S-Refine always outperforms Naïve-Refine significantly, which validates the effectiveness of the divide-and-conquer strategy.

Efficiency on synthetic data sets. Finally, we report the efficiency study on our synthetic data sets. Figure 11 shows that SPLITTER outperforms the compared methods on both data sets. The performance gap is more obvious on S1K, mainly because HC and GRID are ineffective on S10K and terminate at an early stage. For example, when $\sigma = 300$, SPLITTER finds 6 length-3 fine-grained patterns on S10K, while HC and GRID find none.

Summary of efficiency study. The above results demonstrate the efficiency of SPLITTER. Under a few parameter settings (when σ and Δt are large), HC may take less time than SPLITTER, but that comes with the price of being much less effective.

6. RELATED WORK

Sequential pattern mining in transactional data has been extensively studied. Agrawal and Srikant [1] first introduce this problem and employ Apriori to discover patterns. Other efficient solutions include projection-based method PrefixSpan [10] and vertical formatting method SPADE [16]. However, none of these algorithms can handle trajectory data due to spatial continuity.

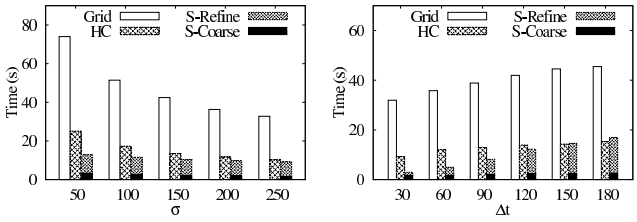
Several pioneering studies [11, 12] have investigated mining sequential patterns in spatio-temporal databases. To handle spatial continuity, they adopt the space partitioning strategy, which discretizes the whole space into many small grids based on a prespecified granularity. Though simple and efficient, rigid space partitioning is not suitable for mining fine-grained sequential patterns. It suffers from the sharp boundary problem, namely the locations close to grid boundaries may be assigned to different grids and thus potential fine-grained patterns may be lost.

Giannotti *et al.* [5] define the *T-pattern* in a collection of GPS trajectories. A T-pattern is a Region-of-Interest (RoI) sequence with temporal annotations, where each RoI as a rectangle whose density is larger than a threshold δ. However, their method still relies on rigid space partitioning. In addition, the threshold δ is hard to prespecify for our problem: a small δ will lead to very coarse regions while a large one may eliminate fine-grained patterns.

Zheng *et al.* [18] study the problem mining interesting travel sequences from GPS trajectories. They extract top-m most interesting place sequences in a given region. Such sequences, however, are not necessarily frequent among the input trajectories. Moreover, in order to extract top-m length-n sequences, they need to enumerate all possible place sequences and compute their scores. Luo *et al.* [9] proposed a scalable method for finding the most frequent path in trajectory data. Given two nodes (a source and a destination) in a road network and a time period, their method efficiently finds the most frequent path between the two nodes during the given time period. In contrast to their problem, we seek to find a complete set of fine-grained patterns in the Euclidean space.

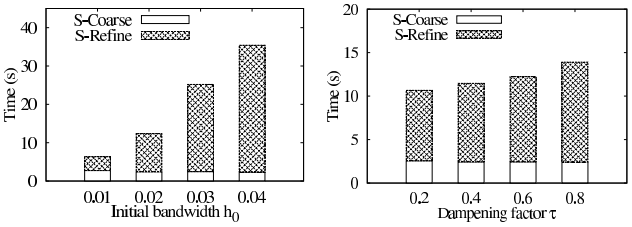
Another important line in trajectory data mining is to mine a set of objects that are frequently co-located. Efforts along this line include mining *flock* [7], *convoy* [6], *swarm* [8], and *gathering* [17] patterns. All these patterns differ from our work in two aspects: (1) they only model the spatio-temporal information without considering place semantics; and (2) they require the trajectories are aligned by the absolute timestamps to discover co-located objects, while we focus on the relative time interval in a trajectory.

There are a few studies on mining sequential patterns in semantic trajectories. Alvares *et al.* [2] first identify the *stops* in GPS trajectories, then match these stops to semantic places using a background map. By viewing each place as an item, they extract the frequent place sequences as sequential patterns. Unfortunately, due to spatial continuity, such place-level sequential patterns can appear only when the support threshold is very low. Ying *et al.* [15] mine sequential patterns in semantic trajectories for location prediction. They define a sequential pattern as a sequence of semantic labels (*e.g.*, school $\rightarrow$ park). Such a definition ignores spatial and temporal information. In contrast, our fine-grained patterns consider the spatial, temporal and semantic dimensions simultaneously.



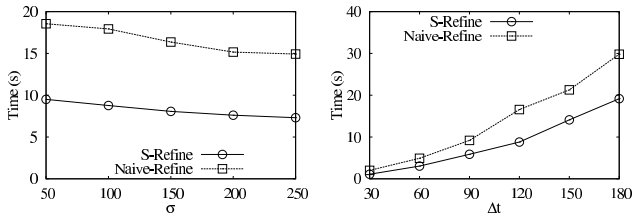
(a) Running time *w.r.t.* σ. (b) Running time *w.r.t.* Δt.

Figure 8: Efficiency comparison on the 4SQ data set.



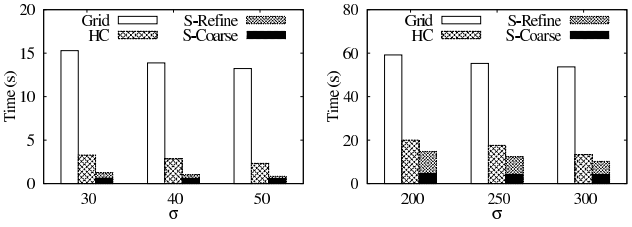
(a) Running time *w.r.t.* h_0. (b) Running time *w.r.t.* τ.

Figure 9: Varying parameters of SPLITTER.



(a) Varying σ. (b) Varying Δt.

Figure 10: Effect of the speedup strategy for splitting.



(a) Running time on S1K. (b) Running time on S10K.

Figure 11: Efficiency study on synthetic data sets.

7. CONCLUSIONS

We introduced and studied the problem of mining fine-grained sequential patterns in semantic trajectories. We proposed SPLITTER to discover fine-grained patterns in a two-step manner. By mining coarse patterns and then progressively refining them via weighted snippet shifting, SPLITTER can mine fine-grained patterns effectively. We also proposed a divide-and-conquer strategy to speed up the top-down pattern splitting process. Our extensive experiments demonstrated the effectiveness and efficiency of SPLITTER.

In our experiments, no coarse pattern has a length larger than 3 under meaningful settings of σ and ρ_t. In case there exist very long frequent coarse patterns, the effectiveness of the weighted snippet shift procedure could be affected by the high dimensionality of the transformed space. However, this limitation can be remedied using dimension reduction techniques or methods developed for mean shift in high-dimensional space [4].

While designed for semantic trajectories, SPLITTER can be easily adapted to mine fine-grained patterns in GPS trajectories. By clustering places into coarse regions, we can first mine coarse patterns with Algorithm 1, and then feed these coarse patterns along with their snippets to Algorithm 3 for refinement. Other interesting future directions include evaluating the usability of the result patterns and extending SPLITTER to find patterns in one long semantic trajectory.

8. ACKNOWLEDGEMENTS

We thank the reviewers for their insightful comments. The work was supported in part by the U.S. Army Research Laboratory under Cooperative Agreement No. W911NF-09-2-0053 (NS-CTA) and W911NF-11-2-0086 (Cyber-Security), the U.S. Army Research Office under Cooperative Agreement No. W911NF-13-1-0193, U.S. National Science Foundation grants CNS-0931975, IIS-1017362, IIS-1320617, IIS-1354329, DTRA, NASA NRA-NNH10ZDA001N, National Science Foundation of China grant No. 61170034, and MIAS, a DHS-IDS Center for Multimodal Information Access and Synthesis at UIUC.

9. REFERENCES

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

[2] L. O. Alvares, V. Bogorny, B. Kuijpers, B. Moelans, J. A. Fern, E. D. Macedo, and A. T. Palma. Towards semantic trajectory knowledge discovery. *Data Mining and Knowledge Discovery*, 2007.

[3] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, 2002.

[4] B. Georgescu, I. Shimshoni, and P. Meer. Mean shift based clustering in high dimensions: A texture classification example. In *ICCV*, pages 456–463, 2003.

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

[6] 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.

[7] P. Laube and S. Imfeld. Analyzing relative motion within groups of trackable moving point objects. In *GIScience*, pages 132–144, 2002.

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

[9] 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.

[10] 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.

[11] I. Tsoukatos and D. Gunopulos. Efficient mining of spatiotemporal patterns. In *SSTD*, pages 425–442, 2001.

[12] J. Wang, W. Hsu, M.-L. Lee, and J. T.-L. Wang. Flowminer: Finding flow patterns in spatio-temporal databases. In *ICTAI*, 2004.

[13] Z. Yan, D. Chakraborty, C. Parent, S. Spaccapietra, and K. Aberer. Semitri: a framework for semantic annotation of heterogeneous trajectories. In *EDBT*, pages 259–270, 2011.

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

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

[16] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.

[17] K. Zheng, Y. Zheng, N. J. Yuan, and S. Shang. On discovery of gathering patterns from trajectories. In *ICDE*, pages 242–253, 2013.

[18] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *WWW*, pages 791–800, 2009.

APPENDIX

Proof of Theorem 1: As $\hat{f}(\mathbf{y})$ is bounded, it suffices to prove the density at the kernel window center increases after each shifting, namely $\forall k,\ \hat{f}(\mathbf{y}^{(k+1)}) > \hat{f}(\mathbf{y}^{(k)})$ if $\mathbf{y}^{(k+1)} \neq \mathbf{y}^{(k)}$. Without loss of generality, assume $\mathbf{y}^{(k)}$ is the origin of the space, namely $\mathbf{y}^{(k)} = \mathbf{0}$. We denote by $\mathcal{N}_k$ the set of points inside the window of $\mathbf{y}^{(k)}$. The kernel density at $\mathbf{y}^{(k)}$ is thus

$$\hat{f}(\mathbf{y}^{(k)}) = \frac{c}{h^{d+2}w} \sum_{\mathbf{x}_i \in \mathcal{N}_k} w_i (h^2 - \|\mathbf{x}_i\|^2). \tag{4}$$

Denote by $\mathcal{N}_{k+1}$ the points inside the window of $\mathbf{y}^{(k+1)}$, and let $\mathcal{N}_\cap = \mathcal{N}_k \cap \mathcal{N}_{k+1}$. Since $\mathcal{N}_\cap \subseteq \mathcal{N}_{k+1}$, we have

$$\hat{f}(\mathbf{y}^{(k+1)}) \geq \frac{c}{h^{d+2}w} \sum_{\mathbf{x}_i \in \mathcal{N}_\cap} w_i (h^2 - \|\mathbf{y}^{(k+1)} - \mathbf{x}_i\|^2). \tag{5}$$

To prove $\hat{f}(\mathbf{y}^{(k+1)}) > \hat{f}(\mathbf{y}^{(k)})$, we introduce

$$\begin{aligned}\Delta_f &= \sum_{\mathbf{x}_i \in \mathcal{N}_\cap} w_i (h^2 - \|\mathbf{y}^{(k+1)} - \mathbf{x}_i\|^2) - \sum_{\mathbf{x}_i \in \mathcal{N}_k} w_i (h^2 - \|\mathbf{x}_i\|^2) \\ &= \sum_{\mathbf{x}_i \in \mathcal{N}_k} w_i \|\mathbf{x}_i\|^2 - \sum_{\mathbf{x}_i \in \mathcal{N}_\cap} w_i \|\mathbf{y}^{(k+1)} - \mathbf{x}_i\|^2 - \sum_{\mathbf{x}_i \in \mathcal{N}_k - \mathcal{N}_\cap} w_i h^2.\end{aligned}$$

Note that $\forall \mathbf{x}_i \in \mathcal{N}_k - \mathcal{N}_\cap, \|\mathbf{y}^{(k+1)} - \mathbf{x}_i\|^2 > h^2$, hence

$$\sum_{\mathbf{x}_i \in \mathcal{N}_k - \mathcal{N}_\cap} w_i h^2 < \sum_{\mathbf{x}_i \in \mathcal{N}_k - \mathcal{N}_\cap} w_i \|\mathbf{y}^{(k+1)} - \mathbf{x}_i\|^2. \tag{6}$$

With Equation 6, Δ_f satisfies:

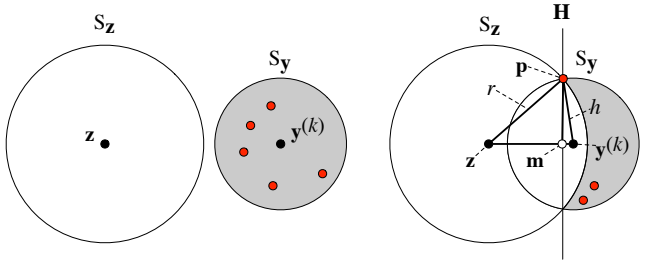
$$\begin{aligned}\Delta_f &> \sum_{\mathbf{x}_i \in \mathcal{N}_k} w_i \|\mathbf{x}_i\|^2 - \sum_{\mathbf{x}_i \in \mathcal{N}_k} w_i \|\mathbf{y}^{(k+1)} - \mathbf{x}_i\|^2 \\ &= 2\mathbf{y}^{(k+1)^{\mathrm{T}}} \sum_{\mathbf{x}_i \in \mathcal{N}_k} w_i \mathbf{x}_i - \|\mathbf{y}^{(k+1)}\|^2 \sum_{\mathbf{x}_i \in \mathcal{N}_k} w_i \\ &= \|\mathbf{y}^{(k+1)}\|^2 \sum_{\mathbf{x}_i \in \mathcal{N}_k} w_i > 0.\end{aligned}$$

Recall Equation 4 and 5, since $\Delta_f > 0$, it is ensured

$$\hat{f}(\mathbf{y}^{(k+1)}) - \hat{f}(\mathbf{y}^{(k)}) \geq \frac{c}{h^{d+2}w} \Delta_f > 0,$$

thus completing the proof.

Proof of Lemma 1: Let $\mathbf{z}$ be a point in any other γ-communities C_j, namely $1 \leq j \leq t$ and $i \neq j$. Consider two hyperspheres: (1) $S_\mathbf{z}$ is a radius-γ hypersphere centered at $\mathbf{z}$, and (2) $S_\mathbf{y}$ is a radius-h hypersphere centered at $\mathbf{y}_k$. Since $S_\mathbf{y}$ cannot be inside $S_\mathbf{z}$, the relationships of $S_\mathbf{z}$ and $S_\mathbf{y}$ can be categorized into two cases.



Case 1: Seperation Case 2: Intersection

Figure 12: Separation and intersection between $S_\mathbf{z}$ and $S_\mathbf{y}$.

Case 1: Separation. When the two hyperspheres are separate, since the new center must fall inside $S_\mathbf{y}$, it is ensured $\|\mathbf{z} - \mathbf{y}^{(k+1)}\| > \gamma \geq \sqrt{2}h$, thus the new hypersphere at $\mathbf{y}^{(k+1)}$ cannot include $\mathbf{z}$.

Case 2: Intersection. When $S_\mathbf{y}$ and $S_\mathbf{z}$ intersect, we denote by $\mathbf{p}$ an intersecting point, $\mathbf{H}$ the hyperplane of intersection, and $\mathbf{m}$ the pedal point of $\mathbf{z}$ on $\mathbf{H}$. Let $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ be the points from C_i that reside in $S_\mathbf{y}$. Points in $\mathcal{X}$ must fall in $S_\mathbf{y} - S_\mathbf{z}$ and on the opposite side of $\mathbf{H}$, *i.e.*, $\forall \mathbf{x}_i \in \mathcal{X}, (\mathbf{z} - \mathbf{m})^{\mathrm{T}}(\mathbf{x}_i - \mathbf{m}) < 0$. With $\mathbf{y}^{(k+1)} = (\sum_{i=1}^n w_i \mathbf{x}_i)/(\sum_{i=1}^n w_i)$, we have

$$(\mathbf{z} - \mathbf{m})^{\mathrm{T}}(\mathbf{y}^{(k+1)} - \mathbf{m}) = \frac{1}{\sum_{i=1}^n w_i} \sum_{i=1}^n w_i (\mathbf{z} - \mathbf{m})^{\mathrm{T}}(\mathbf{x}_i - \mathbf{m}) < 0.$$

Hence, the distance between $\mathbf{z}$ and $\mathbf{y}^{(k+1)}$ satisfies:

$$\begin{aligned}&\|\mathbf{z} - \mathbf{y}^{(k+1)}\|^2 = \|(\mathbf{z} - \mathbf{m}) + (\mathbf{m} - \mathbf{y}^{(k+1)})\|^2 \\ =&\|\mathbf{z} - \mathbf{m}\|^2 + \|\mathbf{m} - \mathbf{y}^{(k+1)}\|^2 + 2(\mathbf{z} - \mathbf{m})^{\mathrm{T}}(\mathbf{m} - \mathbf{y}^{(k+1)}) \\ >&\|\mathbf{z} - \mathbf{m}\|^2 = \|\mathbf{z} - \mathbf{p}\|^2 - \|\mathbf{m} - \mathbf{p}\|^2 \geq \gamma^2 - h^2.\end{aligned}$$

Given $h \leq \gamma/\sqrt{2}$, it is ensured $\|\mathbf{z} - \mathbf{y}^{(k+1)}\|^2 > \gamma^2 - h^2 \geq h^2$, thus the radius-$h$ sphere centered at $\mathbf{y}^{(k+1)}$ cannot include $\mathbf{z}$. As $\mathbf{z}$ is an arbitrary data point not in C_i, the window centered at $\mathbf{y}^{(k+1)}$ can include only points from C_i.

Proof of Theorem 2: Assume the hypersphere at $\mathbf{y}$ encompasses m points $\mathcal{N} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$; the hypersphere at $\mathbf{y}'$ encompasses n points $\mathcal{N}' = \{\mathbf{x}'_1, \mathbf{x}'_2, \ldots, \mathbf{x}'_n\}$. By Lemma 2, we know that $\mathcal{N} \subseteq C_i$ and $\mathcal{N}' \subseteq C_j$. Below, we prove $\mathbf{y} \neq \mathbf{y}'$ by contradiction.

Suppose $\mathbf{y} = \mathbf{y}'$, then the two hypersphere completely overlap, denoted as $S_\mathbf{y}$. Then all points in $\mathcal{N}$ and $\mathcal{N}'$ must fall inside $S_\mathbf{y}$. Now consider the following two cases.

Case 1: $m = 1$ or $n = 1$. Without loss of generality, assume $m = 1$, then we have $\mathbf{x} = \mathbf{y} = \mathbf{y}'$. Since all the points in $\mathcal{N}'$ reside in $S_{\mathbf{y}'}$, there must exist $\mathbf{x}' \in \mathcal{N}'$ such that $\|\mathbf{x} - \mathbf{x}'\| \leq h < \gamma$. This contradicts the definition of γ-community because the minimum distance between C_i and C_j must be at least γ.

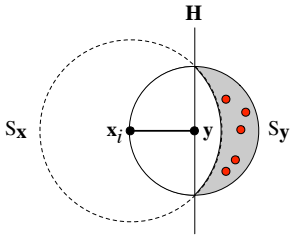


Figure 13: Hypershpere $S_\mathbf{x}$ and $S_\mathbf{y}$.

Case 2: $m > 1$ and $n > 1$. Consider an arbitrary point $\mathbf{x}_i \in \mathcal{N}$ $(1 \leq i \leq m)$. As shown in Figure 13, let $\mathbf{H}$ be the hyperplane that is perpendicular to $\mathbf{y} - \mathbf{x}_i$ and passes $\mathbf{y}$. For any point $\mathbf{x}'_j \in \mathcal{N}'$ $(1 \leq j \leq n)$, we have $\|\mathbf{x}_i - \mathbf{x}'_j\|^2 > \gamma^2$, namely

$$\|(\mathbf{x}_i - \mathbf{y})\| + \|(\mathbf{y} - \mathbf{x}'_j)\| + 2(\mathbf{x}_i - \mathbf{y})^{\mathrm{T}}(\mathbf{y} - \mathbf{x}'_j) > \gamma^2.$$

It follows immediately that

$$2(\mathbf{x}_i - \mathbf{y})^{\mathrm{T}}(\mathbf{y} - \mathbf{x}'_j) > \gamma^2 - \|(\mathbf{x}_i - \mathbf{y})\| - \|(\mathbf{y} - \mathbf{x}'_j)\| \geq \gamma^2 - 2h^2 > 0.$$

Namely $(\mathbf{y} - \mathbf{x}_i)^{\mathrm{T}}(\mathbf{y} - \mathbf{x}'_j) < 0$ for all $\mathbf{x}'_j \in \mathcal{N}'$. In other words, given $\mathbf{x}_i$, all points in $\mathcal{N}'$ must fall in the opposite side of $\mathbf{H}$. This contradicts the fact $\mathbf{y} = \sum_{\mathbf{x}'_j \in \mathcal{N}'} w_i \mathbf{x}'_j / \sum_{\mathbf{x}'_j \in \mathcal{N}'} w_j$, which does not allow the points in $\mathcal{N}'$ to reside in the same side of $\mathbf{H}$. Therefore, the assumption $\mathbf{y} = \mathbf{y}'$ does not hold.