4-2021

Time period-based top-k semantic trajectory pattern query

Munkh-Erdene YADAMJAV

Farhana Murtaza CHOUDHURY

Zhifeng BAO

Baihua ZHENG
Singapore Management University, bhzheng@smu.edu.sg

Time Period-based Top-k Semantic Trajectory Pattern Query

Munkh-Erdene Yadamjav[1], Farhana M. Choudhury[2], Zhifeng Bao[1], and Baihua Zheng[3]

[1] RMIT University, Melbourne, Australia
{munkh-erdene.yadamjav,zhifeng.bao}@rmit.edu.au
[2] The University of Melbourne, Australia fchoudhury@unimelb.edu.au
[3] Singapore Management University, Singapore bhzheng@smu.edu.sg

Abstract. The sequences of user check-ins form semantic trajectories that represent the movement of users through time, along with the types of POIs visited. Extracting patterns in semantic trajectories can be widely used in applications such as route planning and trip recommendation. Existing studies focus on the entire time duration of the data, which may miss some temporally significant patterns. In addition, they require thresholds to define the interestingness of the patterns. Motivated by the above, we study a new problem of finding top-k semantic trajectory patterns w.r.t. a given time period and categories by considering the spatial closeness of POIs. Specifically, we propose a novel algorithm, *EC2M* that converts the problem from POI-based to cluster-based pattern search and progressively consider pattern sequences with efficient pruning strategies at different steps. Two hashmap structures are proposed to validate the spatial closeness of the trajectories that constitute temporally relevant patterns. Experimental results on real-life trajectory data verify both the efficiency and effectiveness of our method.

Keywords: Pattern search · Trajectory queries · Semantic-temporal.

1 Introduction

Recommendation systems utilize data analysis techniques to identify items that match the user's preferences and interests. According to Verified Market Research, global recommendation system market is projected to reach $15.46B by 2026 from $1.12B in 2018 [1]. In this paper, we focus on finding *top-k semantic trajectory patterns* which is related to a type of recommendation particularly useful for entertainment and travel. A typical use case is that Alice, who is going to visit New York City for the first time during Easter holiday, wants to spend quality time at museum, park, and shops. She does not have time to study NYC before her trip. Instead, she relies on the wisdom of the crowd and wants to follow the popular routes people took to visit museum/park/shops last Easter.

Popular location-based services such as Foursquare, Gowalla, and Yelp allow users to upload and update the description of *Points-of-interests (POIs)* and hence lots of POIs are associated with semantic information such as categories. Accordingly, the sequence of check-ins of a specific user over time forms

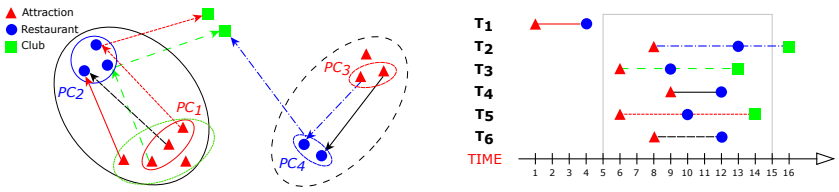


Fig. 1: A running example of five trajectories with their corresponding check-in sequences and timestamps

a semantic trajectory, which represents the movement of that user at different timestamps. To support Alice's query, we focus only on the part of the trajectories during last Easter where the visited POIs belong to museum, and/or park, and/or shops. Ideally, such trajectories of many other users should constitute the most popular routes taken during Easter by users who share similar interests as Alice. Since there can be many spatially close-by POIs of the same category (e.g., shops), the popular routes are expected to include the trajectories that go through close-by POIs of the same category. We study the problem of finding such top-k popular routes, in other words, top-k semantic trajectory patterns.

Although there are existing studies on semantic trajectory pattern mining [18, 5] and top-k frequent pattern mining [9], they suffer from at least one of the following drawbacks. (i) *Finding a threshold that defines the interestingness of a pattern is difficult.* Existing studies [18, 5] measure the interestingness of a pattern based on the number of trajectories that exhibit that pattern, namely *support*, and rely on users to specify a minimum support. Hence, the search results highly depend on users' knowledge on the support number of certain patterns, and an improper value may run the risk of missing interesting patterns. (ii) *Not considering time may miss interesting patterns.* Existing studies [18, 5] mine the patterns from the entire trajectory time duration. However, it is well-known that the activities people perform and the places people visit vary at different times. Although these mining algorithms can be applied over the sub-trajectories w.r.t. the given time period, their efficiency suffer greatly when we consider a set of different category sequences. (more details in Section 6). Moreover, the other drawbacks still apply even their work is extended for time dimension. (iii) *Lack of spatial closeness consideration.* The study [9] can only be applied over semantic trajectory sequences by ignoring the spatial closeness between trajectories.

Motivated by the above and to complement existing studies, we propose a new problem of finding top-k semantic trajectory patterns for a given set of categories ψ and time period P. Informally, the query finds k semantic trajectory patterns, where each pattern is represented by a sequence of POI clusters of given categories. A POI cluster in the pattern consists of the POIs of the same category and are spatially close (e.g., shops that are close-by) to overcome the third drawback. The consideration of P addresses the first drawback. The rank of the pattern considers both the number of query categories appearing in the pattern and the number of trajectories that cover the pattern. Thus, we avoid the necessity of specifying any threshold and overcome the second drawback.

To illustrate the patterns of interest, we plot an example in Figure 1 with six trajectories. Existing work [9] will return the most frequent pattern containing two categories $cs_1 = \langle attraction \rightarrow restaurant \rangle$ with a support of all six trajectories. If the time period is set to $P = \langle 5 : 15 \rangle$, the support of cs_1 changes to T_2, T_3, T_4, T_5, T_6. However, since these trajectories are spatially far-away, such a sequence is not able to suggest any practical route that the user could follow. If we consider the spatial closeness of matching categories, pattern $s_1 = \langle PC_1^{attraction} \rightarrow PC_2^{restaurant} \rangle$ shown with black line ellipse containing trajectories (T_3, T_5, T_6) is a frequent route to take w.r.t. P.

To find such top-k semantic trajectory patterns, we have to address two challenges. *First*, how to assign close-by trajectories into the same pattern efficiently and represent the pattern in an informative way? *Second*, how to accelerate the finding of top-k patterns without enumerating all the subsets of given categories, where we can check as few candidates as possible and prune unpromising patterns at an early stage. In order to address the above challenges, we make the following contributions:

- We address the drawbacks of existing studies that interesting semantic patterns may get missed by proposing the novel *Time Period-based Top-k Semantic Pattern* query, which considers a given time period of interest and a set of categories as input and returns top-k patterns w.r.t. the constraints.
- We propose the algorithm *EC2M* that converts the problem from POIs to a cluster-based pattern search problem to limit the search space significantly. We apply a progressive search strategy from shorter to longer patterns to guarantee the return of top-k patterns. We present a hashmap-based data structure, namely *Enclosing Cluster Cooccurrence Map* for efficient pruning at different steps and another structure namely *Neighbors Bitmap* for validating the spatial closeness of trajectories within given time period.
- We conduct extensive experiments to evaluate the efficiency of our query processing algorithm over two real-world datasets and case studies to demonstrate the effectiveness of our top-k semantic trajectory patterns.

2 Problem Formulation

Let $\mathcal{S}$ be a set of semantic categories (e.g., restaurant, park), and $\mathcal{O}$ be a set of POIs, where each $o \in \mathcal{O}$ is a pair $(o.loc, o.cat)$ of a location $o.loc$ and a category $o.cat \in \mathcal{S}$. Let $\mathcal{D}$ be a database of trajectories, where a trajectory $T \in \mathcal{D}$ is represented as a finite sequence of pairs of a POI and a timestamp (o_i, t_i). Here, timestamp t_i corresponds to the time when the POI o_i was visited by T.

A **POI cluster** (PC) is formed by a set of POIs that belong to the same category and meanwhile are located close to each other. A **semantic trajectory pattern (pattern** in short) is a sequence of such POI clusters where each trajectory in the pattern visits at least one POI in every PC of the sequence.

Example 1. Three close-by POIs with the category 'restaurant' (blue dots) in Figure 1 form a POI cluster PC_1 bounded by a blue ellipse. Note that POIs of the same category inside the black dotted ellipse are not a part of PC_1 because

they are located far away. Three close-by 'attraction' POIs (red triangles) also form a POI cluster PC_2 bounded by a red ellipse. Since trajectories T_3, T_5, T_6 visit at least a POI in PC_1 followed by another POI in PC_2, these trajectories form a pattern $s = \langle PC_1^{attraction} \rightarrow PC_2^{restaurant} \rangle$.

The closeness relationship for clustering depends on the intended application. We use DBSCAN [8] to guarantee the spatial proximity among POIs in a PC, while our problem and approaches are orthogonal to the choice of clustering technique. To find patterns during a specific time period P, we only need to consider the sub-trajectories where the corresponding timestamps are within P. Now, we are ready to formally define the trajectory coverage of a pattern.

Definition 1. ***Trajectory coverage:*** *Given a time period P and a threshold Δt, a trajectory T covers a pattern $s = \langle PC_1^{cat_1}, \ldots, PC_i^{cat_i} \rangle$ with the category sequence $\langle cat_1, \cdots, cat_i \rangle$ if the following conditions are satisfied: (i) there is a subsequence of POIs in T, denoted as $T' = (o_1, o_2, \ldots, o_i)$, such that $\forall o_i \in T'$, $o_i.cat = cat_i$; (ii) the timestamps of all POIs in T' are within P; and (iii) for any POI $o_j \in T'$ with $j < i$, the time gap between o_j and its subsequent POI o_{j+1} is always bounded by Δt, i.e., $(t_{j+1} - t_j) \leq \Delta t$ always holds.*

Here, the parameter Δt is used to find the patterns where the consecutive POI visits happened within a reasonable time gap (e.g., by setting $\Delta t = 24$ hours). Note that, if a trajectory contains multiple sub-trajectories with the same category sequence, that trajectory covers the pattern only once, but all those unique sub-trajectories are used to form the clusters of the pattern.

Example 2. Given a query with $P = \langle 5, 15 \rangle$, a pattern $s = \langle PC_1^{attraction} \rightarrow PC_2^{restaurant} \rangle$ located in the solid black ellipse, and $\Delta t = 10$, trajectories T_3, T_5, and T_6 cover s (Figure 1). Although T_1 contains POIs belonging to the given categories, its corresponding timestamps are not within P. Hence, it is worth noting that the POIs that belong only to T_1 are not included in the pattern's POI clusters (shown as the black line ellipses).

When a pattern w.r.t. a time period is covered by many trajectories, it actually implies a "popular" route taken by users when visiting those categories within that time period. Now we introduce a measure of the popularity of a pattern that considers both the number of trajectories covering the pattern, and its semantic importance to a user query.

Definition 2. ***Popularity measure:*** Given a set of categories ψ, a pattern s where $\forall cat_i \in s$, $cat_i \in \psi$, and the trajectories covering s w.r.t. P and Δt (by Definition 1), the popularity of s, denoted as $Sr(s, \psi)$, is computed by Equation (1).

$$Sr(s, \psi) = \alpha \frac{|D_P^s|}{|D_P|} + (1 - \alpha) \times \frac{|s|}{|\psi|} \tag{1}$$

Here, D_P^s is the set of trajectories covering s, D_P is the set of trajectories containing at least one POI of any category in ψ, $|s|$ is the number of categories

with at least one trajectory covering s, and $\alpha \in [0,1]$ is used to set the preference over one component to the other. The second component $\frac{|s|}{|\psi|}$ (denoted as 'category sub-score') quantifies the matching between the categories of s and ψ. Although there are many ways to combine two components in a scoring function, weighted summation is the most common in many spatial-keyword studies [7, 10]. Now we introduce our *Time Period-based Top-k Semantic Pattern* query.

Definition 3. ***Time period-based Top-k Semantic Pattern (TkSP):*** Given a trajectory database $\mathcal{D}$, a time period P, a time threshold Δt, and categories of interest ψ, the *TkSP* query is to find k highest scoring patterns w.r.t. P and ψ (by Definition 2).

Example 3. A *TkSP* query is given with $k = 1$, $P = \langle 5, 15\rangle$, $\Delta t = 10$, and $\psi = \langle$restaurant, attraction$\rangle$. We find six candidate patterns $s_1 = \langle PC_1^{attraction}\rangle : \{T_3, T_5, T_6\}$, $s_2 = \langle PC_2^{restaurant}\rangle : \{T_3, T_5, T_6\}$, $s_3 = \langle PC_3^{attraction}\rangle : \{T_2, T_4\}$, $s_4 = \langle PC_4^{restaurant}\rangle : \{T_2, T_4\}$, $s_5 = \langle PC_1^{attraction} \rightarrow PC_2^{restaurant}\rangle : \{T_3, T_5, T_6\}$, and $s_6 = \langle PC_3^{attraction} \rightarrow PC_4^{restaurant}\rangle : \{T_2, T_4\}$. Their scores for $\alpha = 0.5$ are calculated as, $Sr(s_1, \psi) = Sr(s_2, \psi) = 0.5 \cdot \frac{3}{5} + 0.5 \cdot \frac{1}{2} = 0.55$, $Sr(s_3, \psi) = Sr(s_4, \psi) = 0.5 \cdot \frac{2}{5} + 0.5 \cdot \frac{1}{2} = 0.45$, $Sr(s_5, \psi) = 0.5 \cdot \frac{3}{5} + 0.5 \cdot \frac{2}{2} = 0.8$, $Sr(s_6, \psi) = 0.5 \cdot \frac{2}{5} + 0.5 \cdot \frac{2}{2} = 0.7$. Pattern s_5 that contains both query keywords and 3 trajectories is returned as the result with the highest score.

3 Baseline Method

Since existing approaches do not directly answer *TkSP* query, we tailor the state-of-the-art Top-k sequential pattern mining (*TkS*) [9], denoted as *TkS**, for our baseline. It follows a *retrieval-and-refinement* framework. It first retrieves the POIs of trajectories satisfying ψ and P, with trajectories indexed by their timestamps using a B^+-tree [6]. It then applies *TkS** to find the top-k patterns from these sub-trajectories. We use 'trajectory' instead of 'sub-trajectory' in the following for simplicity. One may think of enumerating all permutations of query categories and finding the trajectories covering a permutation as a pattern. However, trajectories covering spatially distant patterns with the same category sequence need to be distinguished (Section 2). Different from *TkS*, *TkSP* expects multiple patterns corresponding to a category sequence and hence *TkS* is tailored to compute their popularity scores independently.

Generation of top-k patterns. *TkS** first generates the POI clusters from the retrieved POIs. Similar to *TkS*, we use a vertical bitmap representation to find trajectories covering a specific POI and create a cooccurrence map to find subsequent POIs visited within Δt timestamps for a specfic POI. If a trajectory contains a POI in the POI cluster, the position that POI of that trajectory in the bitmap is set to 1, otherwise 0. These POI clusters are the 1-length patterns, which are the current candidate patterns. The candidates are maintained in descending order of their popularity scores. In each iteration, if the popularity score of the head pattern *hp* (i.e., the candidate with the highest score) is larger than that of a current result, the result set is updated accordingly with *hp*.

Since a pattern can be extended by adding a POI cluster of a new category, the popularity score of any longer pattern can increase. Hence, we check what could be the 'potential' score by extending hp. As an extended pattern from hp can have at most the same trajectories as hp and at most $|\psi|$ categories, the potential score is calculated by putting these maximum values in Equation (1). If this potential score is larger than that of the k-th pattern in the current result set, we extend hp using the POI cooccurrence map w.r.t. Δt constraint and the cluster validity (Section 4.1). If the extended patterns of hp have potential popularity scores that are greater than k-th pattern in the current result set, these patterns are added as candidates. Once all candidates are checked, the result set of k patterns is returned.

*TkS** is simple but suffers from several major drawbacks: (i) Generating the POI clusters is expensive. (ii) The candidates cannot be pruned if a longer pattern with potential score higher than the current results can be generated from them. As a result, a huge number of candidates need to be checked and the actual trajectories covering those patterns need to be verified in each iteration.

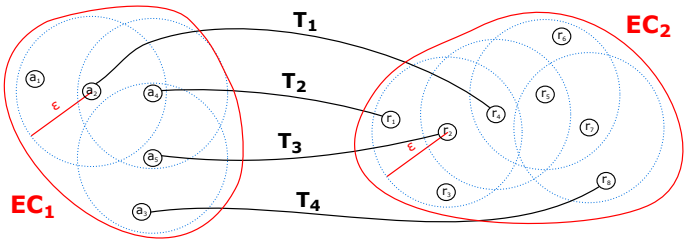
4 Our approach

To overcome the drawbacks of the baseline, we propose a novel algorithm based on an *Enclosing Cluster Cooccurrence Map*, namely *EC2M*, that takes advantage of the closeness relations of POI clusters and a progressive search technique. We start by presenting some high level key concepts of the algorithm.

1) **POI to cluster conversion.** It is well-known that POIs (e.g., restaurants) do not change their locations often. Hence, we include this pre-processing step where all POIs in the dataset are clustered based on the category and spatial closeness, but without considering the exact trajectories passing through them. We denote these clusters as the '*enclosing clusters*'. In Figure 1, the green ellipse shows an enclosing cluster corresponding to category 'attraction'.

2) **Pruning search space by pattern length.** *For simplicity, we refer to the number of POI-clusters in a pattern as its 'length'.* The following key concept can significantly prune the search space – **The maximum length of a pattern for any given query is $|\psi|$.** The reason is: the popularity score of pattern s is determined by two sub-scores, where one depends on the number of covering trajectories and the other on its number of categories. Adding a new cluster to s whose category already exists in s does not increase its number of categories. Moreover, for any pattern s' extended from s, the trajectories that cover s' will always cover s. Therefore, extending a pattern from a given pattern s by adding a new cluster whose category already exists in s, the popularity score does not increase. Therefore, we only need to consider the patterns up to length $l = |\psi|$.

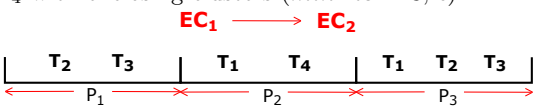
3) **Upper bound score.** To enable the filtering of unpromising candidates, we introduce the *enclosing cluster cooccurrence map*. This map has each enclosing cluster as the keys and the list of its covering trajectories w.r.t. Δt and P as values. It is created only when we need to search for patterns greater than 1-length (explained later). The maximum number of the covering trajectories for a pattern is the number of trajectories that are common in all 'enclosing clusters' of that pattern, which can be readily obtained from the map. Such upper bound



(a) Check-in sequence $\langle attraction \rightarrow restaurant \rangle$ for trajectories T_1, T_2, T_3, T_4 with enclosing clusters ($minPts = 3$, ϵ)

POI	BITMAP
r_1	01000000
r_2	10110000
r_3	01000000
r_4	01001000
r_5	00010110
r_6	00001000
r_7	00001001
r_8	00000010

(b) Neighbours bitmap for EC_2



(c) Covering trajectories of enclosing cluster sequence $EC_1 \rightarrow EC_2$ for different time bins

Fig. 2: Enclosing cluster information

can be loose if we consider a specific time period. Therefore, we split the time dimension into multiple bins of fixed duration and organize the cooccurrence map accordingly as shown in Figure 2c. If a trajectory spans two or more consecutive time bins, we store it in all the corresponding bins.

4) **Pattern extension and progressive search.** If a pattern s is extended to s' by adding a category that is not in s, s' may have a higher popularity score than s as its category sub-score increases. Thus, we cannot prune any candidate s without checking all its potential extensions that might outscore it.

Therefore, we progressively search patterns from length $l = 1$ to length $l = |\psi|$. For each length, we find top-k patterns up to that length. Hence, for pattern s with length i currently under consideration, it will be guaranteed that all the shorter patterns that s might outscore, have been already considered. We also compute the maximum possible score that can be obtained by extending current s using the enclosing cluster co-occurance map. Only when that score is greater than the current results, we consider the possible patterns of length $i + 1$ extended from s (also obtained from the map). This step guarantees that any longer pattern that might outscore a shorter pattern will not be missed.

If the pattern under consideration cannot be pruned based on enclosing cluster based upper bound, we need to find the covering trajectories on the fly to refine its actual POI clusters. All clusters in a pattern need to be checked against the **'valid cluster connectivity'** using the covering sub-trajectories. The cluster validation is presented in more details next.

4.1 Cluster validity check

All patterns that are passed to the cluster validity check contain POIs grouped by the enclosing cluster information. However, as the enclosing clusters are formed without considering the exact trajectories actually passing through their POIs, it is possible that some POIs in an enclosing cluster are not part of the actual pattern. The following example illustrate one such scenario.

Example 4. Figure 2a shows a pattern s generated based on the enclosing clusters EC_1 and EC_2. Enclosing clusters are generated based on all POIs w.r.t. spatial closeness. We use DBSCAN clustering w.r.t. the parameters: $minPts$ and ϵ, where at least $minPts$ POIs have to be within ϵ distance from any POI of the same cluster to form a cluster. EC_1 contains five '*attraction*' and EC_2 contains eight '*restaurant*' POIs. Assume the pattern is covered by trajectories T_1, T_2, T_3, and T_4. The attraction POIs of the four trajectories form a cluster, but T_4's restaurant POI cannot form a cluster with the restaurant POIs of other trajectories. This is because none of the restaurant POIs in T_1, T_2, T_3 is within ϵ from the restaurant POI in T_4. Hence, there are actually two patterns (split from EC_1 and EC_2), where one pattern s_1 is covered by T_1, T_2, and T_3, and the other pattern s_2 is covered only by T_4.

Although the number of POIs in an enclosing cluster is much smaller than the total number of POIs, we still need to compute the distances of every pairs of POIs if we re-cluster them. To overcome this limitation, we store a bitmap representation of neighbors for each POI w.r.t. its enclosing cluster. Such structure eliminates the need for spatial distance computation for every pair. Furthermore, the POIs are compared with only the near-by POIs that potentially could form a cluster. Figure 2b shows the neighbor bitmaps for each POI in EC_2. The length of each POI's bitmap equals to the size of the enclosing cluster, e.g., eight in this example. A bit position is assigned to each POI in the cluster. If two POIs are within ϵ distance, then the corresponding bit is 1. For example, the bit sequence 01000000 associated with POI r_1 indicates only r_2 is located within ϵ to it.

4.2 Algorithm

The pseudo code of the *TkSP* query processing algorithm is presented in Algorithm 1. In this approach, the trajectories are indexed by a B^+-tree. A max-priority queue PQ is maintained to keep track of the candidate patterns, where the key is their upper bound popularity score. Result set R keeps k patterns with the highest popularity scores found so far (Line 1.1). At first, the set of enclosing clusters is obtained by retrieving trajectories that pass through at least one query category within P using *tree*. Since each enclosing cluster is a pattern of length 1 (Line 1.2), they are enqueued to PQ. An enclosing cluster co-occurrence map is created w.r.t. P and the query categories ψ by considering time bins that intersect with P. This map is created only when we need to search for longer patterns, i.e., when $|\psi| > 1$ (Lines 1.3-1.4). The map contains the enclosing clusters as keys, and the list of trajectory IDs as values.

The candidate patterns are progressively considered from length $j = 1$ to length $j = |\psi|$ (Line 1.5). For a length j under consideration, another priority queue NQ stores the patterns that may need to be extended to length $(j + 1)$ in next iteration. For a candidate pattern *cand* dequeued from PQ, we perform two actions. First, we compare its upper bound score with the current k-th best score. Note, when R has less than k results, $getMinScore(R)$ returns 0. If *cand* has a higher score, we extract all the trajectories that cover *cand*, and further validate the pattern via the function $validateClusters$ (Lines 1.9-1.11).

Algorithm 1: *Time Period-based Top-k Semantic Pattern* query

```
    Input: B+-tree tree, time period P, time constraint Δt, query categories ψ, k
    Output: set of k patterns R
1.1 R ← ∅, PQ ← an empty priority queue
1.2 Enqueue(PQ, enclosing clusters satisfying constraints(tree, ψ, P))
1.3 if |ψ| > 1 then
1.4 |   ccMap ← generate co-occuring cluster map(ψ, P, Δt)
1.5 for j ← 1 to |ψ| do
1.6 |   NQ ← ∅
1.7 |   while PQ is not empty do
1.8 |   |   cand ← dequeue(PQ)
1.9 |   |   if getScore(cand) > getMinScore(R) then
1.10|   |   |   List ← getTrajectoryList(cand)
1.11|   |   |   validateClusters(List, j, R)
1.12|   |   if getMaxScore(cand) > getMinScore(R) then
1.13|   |   |   Enqueue(NQ, extendP(cand, ccMap))
1.14|   PQ ← NQ
1.15 return R
```

The pseudo-code for cluster validation step is presented in Algorithm 2 and will be explained later. Second, we estimate the maximum possible popularity score of any longer pattern that can be extended from *cand*. If this estimated score is greater than the current k-th best score, we generate the extensions of *cand* of length $j+1$ using the co-occurrence map, and enqueue them to NQ (Lines 1.12-1.13). At the end of the iteration corresponding to a length j, we replace PQ with NQ for the next length $j+1$ (Line 1.14). Finally, we return result set R with k most popular patterns (Line 1.15).

Cluster validation algorithm. The function *validateClusters* validates a candidate pattern, with its pseudo-code presented in Algorithm 2. As mentioned in Section 3, the candidate patterns are generated from enclosing clusters that do not consider the exact trajectories passing through them, hence the validation procedure is necessary to find the actual patterns with valid POI clusters. Here, two hashmaps MAP_P and MAP_{ind} are created from the input list of trajectories LT (Line 2.1). MAP_P has each POI as a key and the list of trajectories passing through that POI as the value. A trajectory ID in LT and the order of a POI visit is a key (as a tuple) in MAP_{ind} and the list of POIs visited by that trajectory at the corresponding order of visit are the value. A first-come-first-serve queue Q_T is maintained for the set of trajectories that needs to be considered. Q_T is initialized with LT (Line 2.2). For the trajectory set dequeued from Q_T, we find the first POI visited by each trajectory using MAP_{ind}, and store these POIs in LP. We use the neightbors bitmap information (Section 4.1) to obtain the actual POI clusters C formed by the POIs in LP (Line 2.4).

For each cluster $c \in C$, we obtain the trajectories covering c using MAP_P (Line 2.6), and store them in list LT'. Note, $LT' \subseteq LT$. To facilitate pruning of an unpromising candidate at this stage, we compute an upper bound popularity score using c, where the number of covering trajectories is set to $|LT'|$ (as any extended pattern will not have more covering trajectories) and the number of categories is set to l (the maximum categories in a pattern of length l). If this score is lower than the k-th best score, we can safely terminate the examination

Algorithm 2: validateClusters()

```
   Input: list of trajectories LT, length l, current result set R
2.1  MAP_P ← createPOIMap(LT); MAP_ind ← createIndexMap(LT)
2.2  Q_T ← getIDs(LT)
2.3  while Q_T is not empty do
2.4      LP ← getPOIs(Q_T.poll(), MAP_ind, 1); C ← clusterPOIs(LP)
2.5      foreach cluster c ∈ C do
2.6          LT' ← getTrajectory(c, MAP_P)
2.7          if getScore(l, |LT'|) > getMinScore(R) then
2.8              p ← Initialize with cluster c; Update R if the length of p equals l
2.9              for j ← 2 to l do
2.10                 NLP ← getPOIs(LT', MAP_ind, j)
2.11                 CN ← clusterPOIs(NLP)
2.12                 if |CN| = 1 then
2.13                     p.add(c' ∈ CN); Update R if the length of p equals l
2.14                 else
2.15                     foreach cluster c' ∈ CN do
2.16                         NLT' ← LT' ∩ getTrajectory(c', MAP_P)
2.17                         if |LT'| = |NLT'| then
2.18                             p.add(c'); Update R if the length of p equals l
2.19                         else
2.20                             Q_T.add(NLT')
2.21                     if ∀c' ∈ CN not extends p then
2.22                         break
```

of c. Otherwise, we initialize a pattern p with c. If $l = 1$, we update R accordingly (Line 2.8). For $l > 1$, we extend p by checking the POIs visited subsequently until its length reaches l. We use parameter j to indicate the visiting order of POIs to be evaluated next (Line 2.9).

We scan each trajectory in LT' to retrieve the j-th visited POIs using MAP_{ind} and store these POIs in NLP (Line 2.10). The neightbors bitmap is used to obtain the actual POI clusters formed by the POIs in NLP, and the resulting clusters are stored in CN (Line 2.11). If CN has only one cluster c' (i.e., the POIs visited next are all contained in one POI cluster), we can extend the current pattern p by appending c'. If the length of p becomes l and it's score is higher than the k-th best score, R is updated (Lines 2.12-2.13). If there are multiple clusters in CN, it indicates that there are multiple options in terms of the next POI cluster to visit from the current p, and we have to explore each $c' \in CN$. For $c' \in CN$, we obtain the list of trajectories in LT' that also visit a POI in c' in its j-th place, and store them in NLT' (Line 2.16). If $|NLT'| = |LT'|$, c' is added to p and we check if R needs to be updated (Lines 2.17-2.18).

Otherwise, we add NLT' to Q_T as a candidate pattern (Lines 2.19-2.20). If no cluster in CN could extend p, we stop the validation of the current c (Lines 2.21-2.22).

5 Experimental evaluation

In this section, we compare our proposed algorithm with the baseline through an experimental evaluation using real datasets. All algorithms are implemented in Java. Experiments were ran on a 24 core Intel Xeon E5-2630 2.3 GHz using 256GB RAM, and 1TB 6G SAS 7.2Krpm SFF (2.5-inch) SC Midline disk

Description	*Foursquare*	*Yelp*
# of POIs	61,856	209,393
# of check-ins	573,012	8,016,526
# of users	2,293	1,968,703
# of categories	247	21

Table 1: Database statistics

Parameter	*Dataset*	*Values*
Time period P	Foursquare	2, 4, **6**, 8, 11 (month)
	Yelp	**3**, 6, 9, 12, 15 (year)
Preference α	Both	0.1, 0.3, **0.5**, 0.7, 0.9
# of categories $\|S\|$	Both	1, 2, **3**, 4

Table 2: Experimental parameters

drives running Red Hat Enterprise Linux Server release 7.5. We test the following methods to answer *TkSP* queries on real-life datasets: (1) *TkS**, a tailored Top-k Sequential Pattern Mining [9] on top of a B^+-tree index as baseline (introduced in Section 3); (2) *EC2M*, a *Time Period-based Top-k Semantic Pattern* query processing algorithm on top of B^+-tree index using the enclosing clusters cooccurrence map (introduced in Section 4).

Datasets. Foursquare [17] dataset includes check-in data collected from 04 April 2012 to 16 February 2013 in Tokyo, Japan. Yelp dataset includes check-in data between 12 October 2004 and 13 Dec 2019. Each trajectory in the dataset is a sequence of POIs with the corresponding timestamps and semantic categories. Table 1 shows statistics on Foursquare and Yelp datasets.

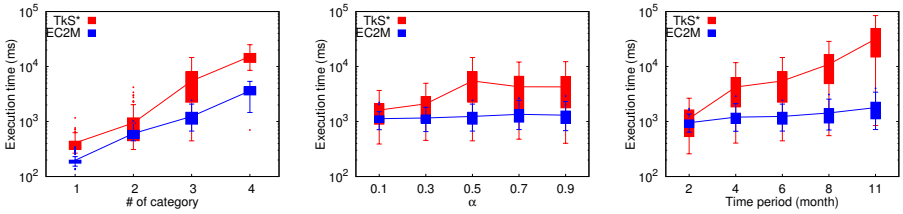
Evaluation and Parameterization. We compared the runtime of all methods by varying the query input parameters as shown in Table 2, where the values in bold represent the default values. For all experiments, a single parameter is varied while other parameters are set to their default values.

Clustering. We evaluated varying parameter combinations for $minPts$ and ϵ to cluster POIs w.r.t. the category using DBSCAN [8] algorithm and chose the following values for our experiment by considering the number of obtained clusters: $minPts = 3$ and $\epsilon = 100$ meters.

5.1 Efficiency Study

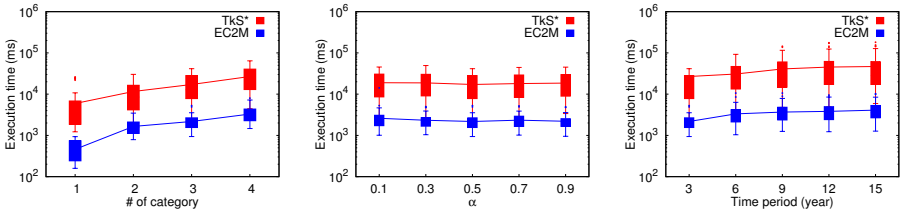
We conduct experiments to evaluate the efficiency of our proposed algorithm against the baseline. We study the impact of each parameter by running 100 queries and report the average query execution time when varying parameters. The results over Foursquare and Yelp datasets are shown in Figures 3 and 4 respectively. The performance for multiple runs is shown in boxplots, where the bounding box shows the first and third quartiles; the whiskers show the range, up to 1.5 times of the interquartile range; and the outliers are shown as separate points. The average values are shown as connecting lines.

Effect of the number of categories. Figures 3a and 4a show the efficiency studies for varying $|\psi|$ over Foursquare and Yelp datasets, respectively. The execution time gap between the algorithms is small for $|\psi| = 1$ since there is no need to generate longer patterns. *EC2M* outperforms *TkS** in all cases by using the bitmap-based POI neighbors information to cluster POIs of the same category. As we add more categories to the query, the benefit of enclosing cluster cooccurrence map becomes more significant. The performance gap between *TkS** and *EC2M* is larger for Yelp than Foursquare dataset. The reason is, Yelp contains more POIs and less categories than Foursquare, which results in more POIs in the POI clusters of Yelp than Foursquare dataset. Hence, the clustering step in the query processing over Yelp dataset greatly benefits from using the bitmap-based POI neighbors information.



(a) Varying ψ (b) Varying α (c) Varying P

Fig. 3: Efficiency studies on Foursquare dataset



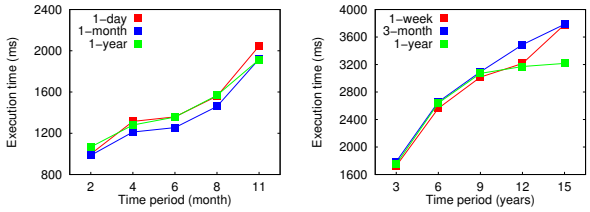
(a) Varying ψ (b) Varying α (c) Varying P

Fig. 4: Efficiency studies on Yelp dataset

Effect of α. Figures 3b and 4b show the performance for varying α over Foursquare and Yelp. The performance gap between two algorithms is not big for smaller values of α in Foursquare. The reason is that Foursquare contains only 2,293 trajectories. As we check the query result for those values of α, the result mostly contains patterns of 1-length. The scores of extended patterns are still lower than the k-th best result. The execution time of the baseline declines for values of α that are higher than 0.5. The reason is that the increase of the k-th best score allows the skip of more shorter patterns, leading to less candidate check. In contrast, Yelp has 1,968,703 trajectories and a small value of α can still contribute significantly to the popularity score.

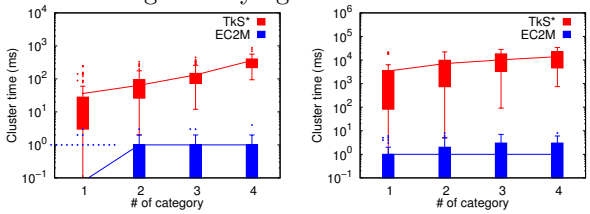
Effect of time period. As we increase time period P, the runtime increases substantially for the baseline in Foursquare dataset, as shown in Figure 3c. In contrast, the query execution time gradually increases w.r.t. varying time periods for Yelp dataset. The reason is that time periods for Foursquare dataset are in the unit of months while time periods for Yelp dataset are in the unit of years (Figure 4c). The number of trajectories covering a certain pattern significantly differs w.r.t. the short time periods. As we expand the time period of interest to a longer time span, eventually we find almost all the trajectories covering a given pattern and the number of new candidate trajectories starts decreasing.

Enclosing clusters cooccurrence map. We obtain 777,018 and 1,513,218 enclosing clusters w.r.t. our clustering settings for Foursquare and Yelp datasets, respectively. Furthermore, we split trajectories covering each cooccurrence into different time bins to see the correlation between time bin size and the co-ocurrrence map size. The sizes of enclosing cluster cooccurrence maps for Foursquare dataset are 25MB, 22MB, 20MB for time bins of 1-day, 1-month, and 1-year, respectively. The sizes of enclosing cluster cooccurrence maps for Yelp dataset are 54MB,



(a) Foursquare (b) Yelp

Fig. 5: Varying sizes of time bins



(a) Foursquare (b) Yelp

Fig. 6: Clustering time

79MB, 48MB for time bins of 7-day, 3-month, and 1-year, respectively. The size increases for 3-month time bin, likely due to the larger number of covering trajectories that span two consecutive time bins.

Figures 5a and 5b depict the query execution time of *EC2M* for varying time periods over Foursquare and Yelp, respectively, by using three different time bins of enclosing cluster cooccurrence maps. Overall, queries over Foursquare dataset run slightly faster on time bins of 1-month for all different time periods. Bigger time bins perform better for large Ps, while queries for short time periods perform slightly faster using small time bins. Queries over Yelp dataset consider different time periods of interest, starting from 3 years to 15 years. For larger Ps, the enclosing cluster cooccurrence map on 1-year time bins shows better query performance. However, queries using 1-week time bins perform slightly better for time periods up to 9 years. Here, non-uniform check-in distribution and sparsity in a user's trajectory result in better performance even we use smaller time bins.

Cluster computation. Our algorithm uses a bitmap representation for clustering close-by POIs. Figures 6a and 6b show the clustering time by running queries for varying number of categories over Foursquare and Yelp datasets, respectively. Since Yelp dataset contains larger clusters, the query performance over Yelp dataset saves significantly more time in the POI clustering than Foursquare dataset. The cluster connectivity check in the pattern greatly benefits from the bitmap-based clustering algorithm as the length of the pattern increases.

5.2 Case study

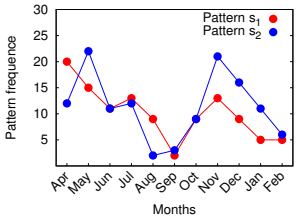
Last, we conduct a case study by presenting the difference between traditional pattern mining over category sequences and semantic trajectory pattern mining proposed in this paper. Table 3 shows top-5 frequent semantic sequences in Foursquare dataset of length 2. The support values depict the number of people (out of 2,293) whose trajectories cover the corresponding pattern. Next, we run

Semantic sequence
Subway→Train Station: 1479
Train Station→Subway: 1452
Train Station→Japanese Restaurant: 1381
Train Station→Ramen/Noodle House: 1366
Japanese Restaurant→Train Station: 1294

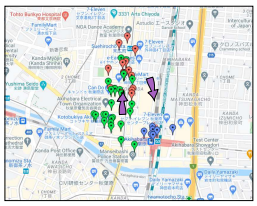
Table 3: Top-5 frequent category sequence over Foursquare

Semantic trajectory pattern
Train station→Electronics Store: 465
Electronics Store→Train Station: 377
Train Station: 1517
Train Station→Hobby Shop: 296
Train station→Electronics Store: 295

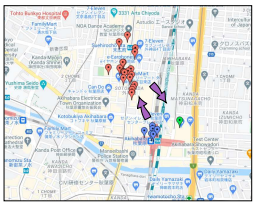
Table 4: Top-5 popular semantic trajectory patterns over Foursquare



(a) Monthly trajectory coverage of s_1 and s_2



(b) Semantic trajectory pattern s_1



(c) Semantic trajectory pattern s_2

Fig. 7: Case Study on the Foursquare dataset

our query to find the top-5 semantic trajectory patterns of up to 2-length, which considers the spatial closeness among covering trajectories. The result shown in Table 4 contains 1-length pattern at the third result. The rank is computed by both the number of trajectories and the number of categories.

Typical patterns (e.g. Subway → Train Station) created by people's daily movement can outweigh the potential patterns of interest. Thus, to guide a user to make a better planning based on the interest of places to visit, we accept a set of categories as a user input. Here, we choose the following keywords: Train Station, Hobby Shop, Electronics Store to further explore the region where that pattern is mostly observed. We find two patterns s_1 = { Electronics Store → Hobby Shop → Train Station } and s_2 = { Train Station → Hobby Shop → Electronics Store } shown in Figures 7b and 7c, respectively. Pattern s_1 is covered by 36 trajectories for the whole database timespan while pattern s_1 is covered by 37 trajectories.

Next, we show the frequency of those two patterns w.r.t. the given time intervals. Since Foursquare contains 11-month check-in data, we split data into one-month intervals and show the changes in the frequency of each pattern w.r.t the given month. From the results presented in Figure 7a, we find the pattern frequencies do change, depending on the time interval of interest which can also contribute to the overall ranking of specific pattern in the result set.

6 Related work

In this section, we review the studies closely related to our work, including (i) *semantic trajectory pattern mining*, and (ii) *top-k sequential pattern mining*.

Semantic trajectory pattern mining. A semantic pattern defined in [18] is the closest to the pattern we consider in this paper. Specifically, a pattern is defined as a sequence of areas in [18], with each area containing places that

are spatially close-by and belong to the same category. A top-down pattern discovery technique called Splitter was proposed. It first generates spatially coarse patterns via a tailored PrefixSpan [12], and then clusters trajectories for each coarse pattern by a variant of the mean shift algorithm. Thus, Splitter works on each category sequence independently, while our work considers different category sequences that can be formed by the given set of categories. Choi et. al. [5] find all regional areas where a semantic pattern is expected to be locally frequent in each area. Trajectories that contain each semantic pattern are clustered by a tailored DBSCAN. The sub-trajectories covering the pattern form a dense cluster of routes. However, the corresponding categories of the sub-trajectories that cover a pattern are not necessarily spatially clustered w.r.t. the category. In contrast, the pattern in our work consists of POI clusters where POIs in a cluster reside spatially close-by and belong to the same category. In addition, the above techniques consider the whole time period in a database and require an input of the minimum support threshold to mine semantic patterns, which is a challenging task for most users. Even if these techniques can be extended by considering sub-trajectories w.r.t. a specific time period, the semantic trajectory patterns found for a given category sequence are not necessarily to be same patterns in our problem setting.

Top-k sequential pattern mining. Many studies have been proposed to mine sequential patterns in transactional databases. Majority of them require specifying a threshold for the minimum number of transactions that need to be contained in a frequent pattern. The performance of the mining algorithms can degrade substantially if the support threshold is set to a smaller value, while the patterns of interest can be overlooked by a larger threshold. Top-k sequential pattern mining algorithms [14, 9, 13] have been proposed to find the k most frequent patterns without requiring to specify the threshold. However, these techniques do not consider the spatial property of the trajectories contained in the result pattern. Thus, the trajectories that cover a specific pattern might be scattered over the search space. In contrast, we aim to find semantic patterns where the trajectories that contain each result pattern are spatially close-by. Moreover, none of the existing work supports a query input for categories of interest.

Other related work. Our query is also loosely related to top-k spatial keyword query and collective spatial keyword query. A traditional top-k spatial-keyword query has been studied extensively in the literature [7, 15, 19]. It returns k most similar objects w.r.t. a query location and keywords by considering both spatial and textual similarities. One variant is a *collective spatial keyword query* [3, 11, 2, 4, 16] which aims to fulfil a user request by considering multiple objects collectively instead of a single object. However, none of the methods is applicable in our case as we do not require the result to be close to a query location. Instead, we aim at supporting users who prefer the past travel experience of other users *over* the proximity between the places to be visited and the query location.

Acknowledgement. Zhifeng Bao is supported by ARC DP200102611. Baihua Zheng is supported by the Ministry of Education, Singapore, under its AcRF Tier 2 Funding (Grant No: MOE2019-T2-2-116).

7 Conclusion

In this paper, we studied the problem of finding top-k semantic trajectory pattern w.r.t. a set of query categories and a time period of interest. We formally defined the problem and proposed algorithms and data structures that improve the efficiency of the query processing. Experimental study on real-life datasets shows the efficiency of our approach.

References

1. Global recommendation engine market by type, by application, by geographic scope and forecast to 2026. https://www.verifiedmarketresearch.com/product/recommendation-engine-market/
2. Cao, X., Cong, G., Guo, T., Jensen, C.S., Ooi, B.C.: Efficient processing of spatial group keyword queries. TODS **40**(2), 1–48 (2015)
3. Cao, X., Cong, G., Jensen, C.S., Ooi, B.C.: Collective spatial keyword querying. In: SIGMOD. pp. 373–384 (2011)
4. Chan, H.K.H., Long, C., Wong, R.C.W.: On generalizing collective spatial keyword queries. TKDE **30**(9), 1712–1726 (2018)
5. Choi, D.W., Pei, J., Heinis, T.: Efficient mining of regional movement patterns in semantic trajectories. VLDB **10**(13), 2073–2084 (2017)
6. Comer, D.: Ubiquitous b-tree. ACM Computing Surveys **11**(2), 121–137 (1979)
7. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. VLDB **2**(1), 337–348 (2009)
8. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: SIGKDD. pp. 226–231 (1996)
9. Fournier-Viger, P., Gomariz, A., Gueniche, T., Mwamikazi, E., Thomas, R.: Tks: efficient mining of top-k sequential patterns. In: ADMA. pp. 109–120 (2013)
10. Li, Z., Lee, K.C., Zheng, B., Lee, W.C., Lee, D., Wang, X.: Ir-tree: An efficient index for geographic document search. TKDE **23**(4), 585–599 (2010)
11. Long, C., Wong, R.C.W., Wang, K., Fu, A.W.C.: Collective spatial keyword queries: a distance owner-driven approach. In: SIGMOD. pp. 689–700 (2013)
12. Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Prefixspan: Mining sequential patterns by prefix-projected growth. In: ICDE. pp. 215–224 (2001)
13. Petitjean, F., Li, T., Tatti, N., Webb, G.I.: Skopus: Mining top-k sequential patterns under leverage. Data Mining & Knowledge Disc. **30**(5), 1086–1111 (2016)
14. Tzvetkov, P., Yan, X., Han, J.: Tsp: Mining top-k closed sequential patterns. KAIS **7**(4), 438–457 (2005)
15. Wu, D., Cong, G., Jensen, C.S.: A framework for efficient spatial web object retrieval. The VLDB Journal **21**(6), 797–822 (2012)
16. Xu, H., Gu, Y., Sun, Y., Qi, J., Yu, G., Zhang, R.: Efficient processing of moving collective spatial keyword queries. The VLDB Journal pp. 1–25 (2019)
17. Yang, D., Zhang, D., Zheng, V.W., Yu, Z.: Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. Transactions on Systems, Man, and Cybernetics: Systems **45**(1), 129–142 (2014)
18. Zhang, C., Han, J., Shou, L., Lu, J., La Porta, T.: Splitter: Mining fine-grained sequential patterns in semantic trajectories. VLDB **7**(9), 769–780 (2014)
19. Zhang, C., Zhang, Y., Zhang, W., Lin, X.: Inverted linear quadtree: Efficient top k spatial keyword search. TKDE **28**(7), 1706–1721 (2016)