Efficient Processing of Spatiotemporal Pattern Queries on Historical Frequent Co-Movement Pattern Datasets

Shahab Helmi and Farnoush Banaei-Kashani

Department of Computer Science and Engineering, University of Colorado Denver
`[shahab.helmi, farnoush.banaei-kashani]@ucdenver.edu`

Abstract. Thanks to recent prevalence of location sensors, collecting massive spatiotemporal datasets containing moving object trajectories has become possible, providing an exceptional opportunity to derive interesting insights about behavior of the moving objects such as people, animals, and vehicles. In particular, mining patterns from co-movements of objects (such as movements by players of a sports team, joints of a person while walking, and cars in a transportation network) can lead to the discovery of interesting patterns (e.g., offense tactics of a sports team, gait signature of a person, and driving behaviors causing heavy traffic). Given a dataset of frequent co-movement patterns, various spatial and spatiotemporal queries can be posed to retrieve relevant patterns among all generated patterns from the pattern dataset. We term such queries, *pattern queries.* Co-movement patterns are often numerous due to combinatorial complexity of such patterns, and therefore, co-movement pattern datasets often grow very large in size, rendering naive execution of the pattern queries ineffective. In this paper, we propose the *FCPIR framework*, which offers a variety of index structures for efficient answering of various range pattern queries on massive co-movement pattern datasets, namely, spatial range pattern queries, temporal range (time-slice) pattern queries, and spatiotemporal range pattern queries.

Keywords— spatiotemporal indexing; spatiotemporal query processing; pattern query;

1 Introduction

Recent advances in location sensors, for instance wearable devices and cell phones with built-in GPS sensors, have enabled collection of massive moving object datasets. We term these datasets multi-variate spatiotemporal event sequence datasets (or MVS datasets, for short) where each event captures the location of an object (among other possible information) at a specific time, and accordingly, each event sequence (or variate) represents the trajectory of an object. MVSs can capture movements/events about various objects in different applications; e.g., joints in human body as a person walks, vehicles navigating a transportation network, ants in a colony as they follow their daily affairs, players in sports teams (e.g., soccer teams) competing on a field, etc. In this paper, we are interested in frequent co-movement patterns, where a frequent co-movement pattern is a pattern of movement for a set of objects that are temporally (not necessarily spatially) close and repeatedly appears in various time windows. Here, we describe three use-cases of frequent co-movement pattern mining[1]: (i) in traffic behavior analysis, finding frequent co-movement patterns of vehicles on a transportation network enables study of the dominating driving behaviors that determine and affect traffic condition in the network; (ii) in human gait analysis, skeletal gait of human, where movement of each join is captured by a variate in the corresponding MVS, can be analyzed to identify joint co-movement patterns that uniquely identify a gait, e.g., for gait-based identification; (iii) in sports analytics, tactics of sports teams can be uncovered by discovering frequent co-movement patterns from MVS datasets collected from players during games.

Co-movement patterns are often numerous due to combinatorial complexity of such patterns, and therefore, the mined co-movement pattern datasets often grow very large in size. Accordingly, naive exploration of such pattern datasets to retrieve relevant patterns becomes very time-consuming, if not infeasible. We term such search queries to retrieve relevant patterns from co-movement pattern datasets, co-movement pattern queries (or *pattern queries* for short). Various conventional (spatiotemporal) queries can be posed as pattern queries. In this paper, we propose the *Frequent Co-movement Pattern Query Indexing and Retrieval (FCPIR)* framework, which introduces novel index structures and query processing algorithms, in order to efficiently process pattern queries on *frequent co-movement pattern datasets*. In particular, we present how *FCPIR* enables efficient processing of three spatiotemporal query families on such frequent pattern datasets: spatial range pattern queries, temporal (or time-slice) range pattern queries, and spatiotemporal range pattern queries.

Such queries are among main types of queries used for exploration of pattern datasets. For example in traffic behavior analysis, one can find and compare frequent driving patterns in the downtown versus suburbs using spatial range pattern queries, while the frequent driving patterns in the whole city can be studied during the rush hours versus regular hours using temporal range pattern queries. Finally, spatiotemporal range pattern query can be used to study frequent driving patterns that have caused an accident chain on a major highway.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents necessary notations and formal definition of the problem. In Section 4, we present the the *SFCMP-tree* and *LM* query processing algorithm for spatial range pattern queries. In Section 5, the *TFCMP-tree* and the *TSPQ* query processing algorithms are proposed for temporal range pattern queries.The *LM-MIF* query processing algorithm is proposed for spatiotemporal range pattern queries in Section 6. Experimental results are provided in Section 7, and Section 8 concludes the paper and discusses the future work.

[1] See [2] for more details about existing methods for MVS frequent co-movement pattern mining.

2 Related Work

In this section, we review the most relevant bodies of literature to our work. Section 2.1 presents trajectory indexing approaches, and Section 2.2 reviews indexing of frequent patterns. Finally, mobility pattern queries are reviewed in Section 2.3

2.1 Trajectory Indexing and Retrieval

Spatiotemporal range queries are used to retrieve the trajectories that are contained in (or intersect with) a given spatiotemporal range. Index structures proposed for these types of queries can be roughly categorized into three groups. The first group treats the temporal dimension similar to the spatial dimensions by adding it to the traditional 2D R-trees (e.g., STR-tree [9].) However, the performance of 3D R-trees significantly degrades if the given dataset is dynamic or trajectories in the dataset are long. This is due to exponential increase in the number of conflicts between MBRs in such 3D R-trees. In order to reduce the number of confilcts, the second group of work proposes constructing a separate R-tree at each time-stamp (time-interval) for the part of dataset introduced in that time-stamp (e.g., HR$^+$-tree [11]). With the third group, the region of interest is divided into a number of cells. Each cell stores the information of those trajectory segments that intersect it (e.g., SETI [1]).
In this paper, we study pattern range queries to retrieve co-movement patterns from pattern datasets, rather than range queries on trajectory datasets. Moreover, we are interested in the retrieval of frequent patterns; hence, checking for intersecting patterns in a given range is not sufficient and the number (frequency) of range intersection must be considered as well.

2.2 Mobility Pattern Queries on Trajectory Datasets

In [6], authors propose a query language for mobility patterns. Similarly, in [10] generic operators are described to support retrieval of known mobility queries (such as finding all trajectories that have moved from point a to point b and finally stayed in point c for 10 minutes). However, since we do not assume any predefined constraints for co-movement patterns, to find all frequent co-movement patterns using the aforementioned methods we need to generate all possible co-movement patterns to be processed using methods such as those discussed in [10], which is most inefficient if at all feasible. To address this problem, we introduce a two-step pattern mining approach instead. With this approach, first we mine all frequent co-movement patterns from a given trajectory dataset using a spatiotemporal extension of the well-known apriori algorithm that we proposed in our prior work [2]. In the second step, which is the focus of this work, we introduce pattern index structures and corresponding query processing algorithms to efficiently retrieve such patterns from frequent co-movement pattern datasets.

2.3 Frequent Pattern Indexing and Management

Indexing structures for frequent patterns can be categorized into two major groups. The first group focuses on similarity search, e.g., finding the k-nearest patterns to a given pattern (e.g., see [7]), while the second group aims at compressing frequent pattern datasets, e.g., by storing only maximal [12] or closed [8] patterns. However, they are neither developed for trajectory (spatial) data, nor allow time-slice (temporal) pattern queries which should account for cases where some frequent patterns become infrequent during a given time-slice. The closest work to our work is [4], where authors propose an index structure for spatiotemporal range queries by repurposing the traditional R-trees in order to reduce the number of disk accesses. However, this approach is only applicable to periodic patterns. Moreover, it only considers spatiotemporal intersection of patterns with the given range not the frequency of intersection. Finally, it only returns the id of qualified objects not the actual trajectories, while in this paper we are interested in retrieving the actual patterns not just their ids.

3 Problem Definition

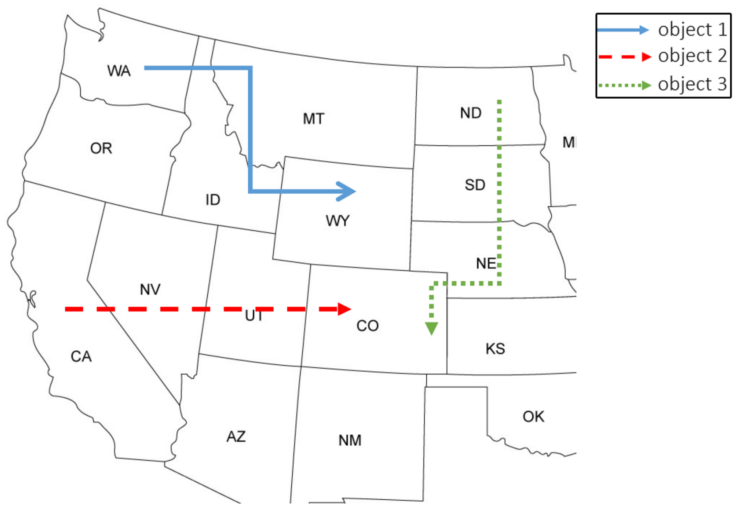

Fig. 1. A co-movement pattern including three objects

Let $O = \{O_1, O_2, \ldots, O_n\}$ be a set of n moving objects. For example, in Figure 1 $O = \{object1, object2, object3\}$. Let S_i show the trajectory of object O_i denoting the spatial path that it follows over time. S_i can be captured as a sequence of locations in 2D space ordered by time as follows: $S_i = (l_{t_1}, l_{t_2}, \ldots, l_{t_m})$, where l_{t_j} shows the location of O_i at the j_{th} timestamp. In this paper, frequent co-movement patterns are of interest, where a frequent co-movement pattern is a pattern of movement for a subset of objects in O that repeatedly appears in various time windows. In this paper, we assume a discrete spatial model (i.e., a grid) by dividing the entire region of interest into smaller regions and replacing absolute locations of objects by their corresponding regions. For example, in Figure 1, the trajectory of object 1 can be modeled as $S_1 = (WA_{t_1}, ID_{t_2}, MT_{t_3}, ID_{t_4}, WY_{t_5})$. Finally, we define multivariate spatial event sequence $MVS = \{S_1, S_2, \ldots, S_n\}$ as a set of n moving object trajectories. In the rest of this section, we provide necessary definitions to define pattern queries on a given dataset of frequent co-movement patterns, which is assumed to be (previously) mined from an MVS dataset.

Definition 1 (Co-Movement Pattern). *We define a co-movement pattern as $\alpha = \{s_1, s_2, \ldots\}$, where each s_i is a continuous sub-sequence of a trajectory S_i. $t_s(\alpha)$ and $t_e(\alpha)$ are also defined to represent the start time and end time of the co-movement pattern α, respectively. We also define the time-span of the co-movement pattern α as $T(\alpha) = (t_e(\alpha) - t_s(\alpha)) + 1$.*

Definition 2 (Valid Co-Movement Pattern). *We consider a co-movement pattern α as a valid co-movement pattern if it satisfies the following conditions: i) α cannot be empty ii) α must not contain more than one sub-sequence s_i from the same event sequence S_i, iii) all sub-sequences in α must have the same length, i.e., the same number of regions in their sequence, and iv) $T(\alpha) \leq T_{max}$, where T_{max} is user-defined maximum allowed time-span of a co-movement pattern.*

To elaborate, Condition *i* is self-explanatory; Condition *ii* ensures we only consider patterns across different objects; Condition *iii* limits the patterns to those that include same number of events from each object invoked in the pattern; and finally, Condition *iv* ensures we only consider temporally local patterns i.e., patterns that appear in the same time window rather than far apart in time. For the sake of simplicity, in the rest of this paper we use co-movement pattern and valid co-movement pattern interchangeably.
In Figure 1, $\alpha = \{(ID, MT, ID)_{S_1}, (CA, NV, UT)_{S_2}\}$ is a valid patterns with $t_s(\alpha) = 1$, $t_e(\alpha) = 4$, and $T(\alpha) = (4 - 1) + 1 = 4$.

Definition 3 (Sub-Pattern and Super-Pattern). *We say α is a sub-pattern of the co-movement pattern α', i.e., $\alpha \subset \alpha'$ (or α' is a pattern of the α) if for each $s_i \in \alpha$ there is a $s'_i \in \alpha'$, such that s_i is a continuous sub-sequence of s'_i.*

In Figure 1, if $\alpha = \{(ID, MT)_{S_1}\}$, $\alpha' = \{(ID, MT, ID)_{S_1}\}$, and $\alpha'' = \{(ID, MT, ID)_{S_1}, (CA, NV, UT)_{S_2}\}$, then $\alpha \subset \alpha' \subset \alpha''$.

Definition 4 (Occurrence and Support). *We say an occurrence of co-movement pattern $\alpha = \{s_1, s_2, \ldots, s_k\}$ exists in a multivariate spatial event sequence dataset MVS, if MVS contains α. We denote all occurrences of α in an MVS as $occ(\alpha) = (\lambda_1, \lambda_2, ..., \lambda_f)$, where λ_j captures the start and end time of each occurrence, and $sup(\alpha) = |occ(\alpha)|$ denotes the support of a co-movement pattern α in MVS dataset.*

Note that our proposed framework allows for any of the numerous existing frequency measures to count occurrences of a pattern in sequence data (e.g., [3]) assuming they preserve the monotonicity property, (i.e., the frequency of a co-movement pattern α cannot be greater than those of its sub-patterns).
In Figure 1, for $\alpha = \{(ID, WY)_{S_1}\}$, $occ(\alpha) = ([4, 5])$ and $|occ(\alpha)| = 1$.

Definition 5 (Frequent Co-movement Pattern). *A co-movement pattern α is a frequent co-movement pattern, if $sup(\alpha) \geq \mu$, where μ is the user-defined minimum support threshold. Given a maximum time-span T_{max}, a minimum-support μ, and a multivariate spatial event sequence dataset MVS of n moving objects, we assume F denotes the set of all frequent co-movement patterns mined from the given MVS. Accordingly, we define $FCPD$ (short for frequent co-movement pattern dataset) a pattern dataset containing F, where for each pattern α in F we store $id(\alpha)$, which is unique for each pattern, as well as $occ(\alpha)$ along with the pattern itself.*

Given the above definitions, we define the patterns queries studied in this paper as follows.

Definition 6 (Spatial Range Pattern Query (SRPQ)). *Given an $FCPD$ and a spatial range query Q_r containing the set of grid regions $Q_r = \{r_1, r_2, ...\}$ from the grid superimposed on the $FCPD$'s region of interest, a spatial range pattern query Q_S returns a subset $P \subseteq FCPD$ of frequent co-movement patterns in the given $FCPD$ such that each pattern $\alpha \in P$ is fully contained in Q_r, i.e., all events in α are located inside some $r_i \in Q_r$.*

Definition 7 (Temporal Range (or Time-Slice) Pattern Query (TRPQ)). *Given an $FCPD$ and a temporal range query (time-slice) Q_t which is a continuous time-interval $Q_t = [t_s, t_e]$, a temporal range pattern query Q_T returns a subset $P \subseteq FCPD$ of frequent co-movement patterns in the given $FCPD$ such that each pattern $\alpha \in P$ frequently occurs during Q_t, i.e., for each pattern $\alpha \in P$, $sup(\alpha) \geq \mu$ during Q_t.*

Definition 8 (Spatiotemporal Range Pattern Query (STRPQ)). *Given an $FCPD$ and a spatial range query Q_r, and a temporal range query Q_t, a spatiotemporal range pattern query $Q_{ST} =< Q_r, Q_t >$ returns returns a subset $P \subseteq FCPD$ of frequent co-movement patterns in the given $FCPD$ such that each pattern $\alpha \in P$ is fully contained in Q_r and frequently occurs during Q_t.*

In Sections 4 through 6, we propose efficient index structures and query processing algorithms for each of the aforementioned queries, respectively.

4 Spatial Range Pattern Queries (SRPQ)

In this section, first we present the *SFCMP-tree* (short for Spatial Frequent Co-Movement Pattern), a clustered index structure for efficient *SRPQ* processing. Then, we describe our *Longest-Match (LM)* query processing algorithm which exploits the *SFCMP-tree* in order to efficiently retrieve qualified frequent co-movement patterns for a given query Q_S.

4.1 SFCMP Tree

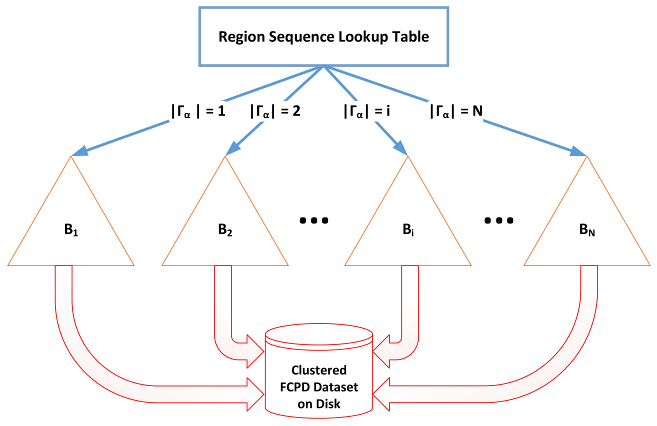

Fig. 2. High-level structure of the *SFCMP-tree* for SRPQ processing

Definition 9 (Region Sequence). *Suppose C is the set of regions covered by α, i.e., the set of regions where each region is the location of at least one of the events in α. We define the region sequence of α, $\Gamma(\alpha)$, as the ordered set of regions in C, where the order is arbitrary, e.g., based on lexicographic order of IDs associated with the regions. We also define $|\Gamma(\alpha)|$ as the level of α.*

For example, if $\alpha = \{(ID, MT, ID)_{S_1}\}$, and $\alpha' = \{(ID, MT, ID)_{S_1}, (CA, NV, UT)_{S_2}\}$, $\Gamma(\alpha) = (ID, MT)$ and $\Gamma(\alpha)' = (CA, ID, MT, NV, UT)$.
Figure 2 shows the high-level structure of the *SFCMP-tree*, τ_s. The *SFCMP-tree* consists of a collection of B^+-tree index structures. Each B^+-tree B_i indexes patterns $\alpha \in FCPD$ that are at level i (i.e., $|\Gamma(\alpha)| = i$), for $i = 1 \cdots N$, where N is the highest level. The region sequence of the pattern α is used as the index key to construct the B^+-tree indexes. Moreover, the B^+-tree indexes are clustered index structures. This is ensured by ordering patterns in the given $FCPD$ first based on their level, and within each level, based on the corresponding region sequence of the patterns at that level. The ordered dataset is then stored in the disk as a clustered file.

4.2 LM Query Processing Algorithm

Lemma 1 (Apriori Property). *If all regions in a region sequence Γ_i for a frequent co-movement pattern α are contained in the given query Q_r, then each pattern α' with $\Gamma_i' \subset \Gamma_i$ is also contained in Q_r.*

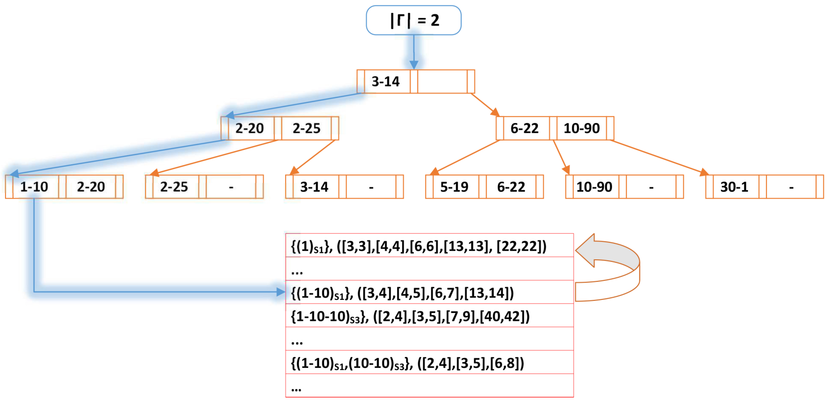
|Γ| = 2
3-14
2-20
2-25
6-22
10-90
1-10
2-20
2-25
-
3-14
-
5-19
6-22
10-90
-
30-1
-
{(1)S1}, ([3,3],[4,4],[6,6],[13,13], [22,22])
...
{(1-10)S1}, ([3,4],[4,5],[6,7],[13,14])
{1-10-10)S3}, ([2,4],[3,5],[7,9],[40,42])
...
{(1-10)S1,(10-10)S3}, ([2,4],[3,5],[6,8])
...

Fig. 3. A detailed example of SFCMP-tree

With the *Largest-Match(LM) SRPQ* processing algorithm, once query Q_r is received, first the region sequence and level of the pattern query is generated as defined in Section 4.1. Next, the region sequence lookup table of *SFCMP-tree* is used to identify the B^+-tree index B_i which indexes patterns at the same level. Thereafter, B_i is traversed using the region sequences of the Q_r as the search key to identify the leaf node that contains the same region sequence (if it exists in $FCPD$). This leaf node has a pointer to a page/block on the disk where all patterns with the same region sequence are clustered; these patterns will be efficiently retrieved to be added to the result set P. However, note that given Lemma 1, we know that all patterns whose regions sequences are sub-sequences of the regions sequence Q_r also belong to P. To retrieve such patterns, the naive approach is to recursively generate all sub-sequences of the Q_r region sequence, and run them as separate queries (as explained above), adding their result to P. As mentioned in Section 4.1, with *SFCMP-tree* we include pointers in the file itself that directly point to the pattern groups with regions sequences Γ_i's, where Γ_i's are sub-sequences of the region sequence Γ_i. Therefore, with *SFCMP-tree* we can directly retrieve all relevant patterns without having to traverse the tree for each sub-query.
As an example, consider $Q_r = \{1, 10\}$. the region sequence of this level 2 Q_r is $1 - 10$. Figure 3 (partially) shows the path which will be traversed to retrieve patterns to process Q_r
Algorithm 1 illustrates the pseudocode for LM algorithm. The input to the algorithm is a set of regions Q_{r_0}, and the output of the algorithm is P, which contains all frequent co-movement patterns that are completely inside Q_{r_0}. First result set P and query queue q are initialized to $\emptyset$ and Q_{r_0}, respectively (Line 3). Then if q is not empty, the head element of q is popped into a temporarily variable tmp. Thereafter, the *Search* method is called to check if τ_s contains tmp, and if so, the corresponding node is returned and added to p (Lines 5-6). Then, using the pointer of p all patterns that their region sequences are equal to

that of Q_{r_i}, are fetched from the stored $FCPD$ by the $Fetch$ method and added to P (Line 6). If tmp does not exist in τ_s, then the $Search$ method returns $\emptyset$. In this case, all direct subsets of tmp are generated by the $Subset$ method and pushed into q (Lines 9-10). Finally, P is returned (Line 11).

Algorithm 1 LM Query Processing Algorithm

```
1: Input: $Q_{r_0}$: a given SRPQ
2: Output: $P$: all frequent co-movement patterns contained in $Q_{r_0}$
3: $P \leftarrow \emptyset, q \leftarrow Q_{r_0}$
4: while ($q \neq \emptyset$) do
5:     $tmp \leftarrow q.pop()$
6:     $p \leftarrow \tau_s$.Search($tmp$)
7:     if $p \neq \emptyset$ then
8:         $P \leftarrow P \cup$Fetch($F, p.pointer$)
9:     else
10:        $q \leftarrow q \cup tmp.Subset()$
11: return $P$
```

5 Temporal Range Pattern Queries (TRPQ)

In this section, we first present the *TFCMP-tree (short for Temporal Frequent Co-Movement Pattern)*. Then, the *Minimum Interval Frequency (MIF)* query processing algorithm is presented to efficiently process *TRPQ* queries using *TFCMP-tree*.

5.1 TFCMP Index Structure

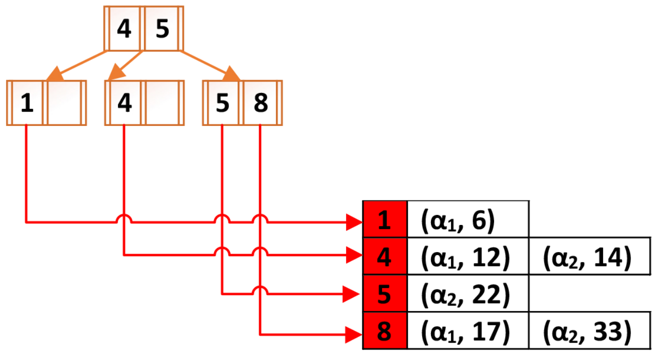

Fig. 4. An instance of the TFCMP-tree

Definition 10 (Minimum Frequency Interval). *Given a timestamp t_i, we define the minimum frequency interval $I_{\alpha,t_i} = [t_i, t_j]$ of a frequent co-movement pattern α such that α is a frequent pattern in any interval $[t_i, t_k]$ if $t_j \geq t_k$, and α is not frequent in any interval $[t'_i, t'_k]$ if $t_j < t_k$.*

For example, assume $occ(\alpha_1) = ([1,3],[4,6],[8,12],[13,17])$, $occ(\alpha_2) = ([4,11],[5,14],[8,22],[12,33])$, and $\mu = 2$. Then, $I_{\alpha_1,1} = [1,6]$. The *TFCMP-tree* corresponding to this example is illustrated in Figure 4.
The *TFCMP-tree* τ_t is basically an inverted index on time, where each node $I_{\alpha,T}$ includes all patterns α in $FCPD$ that have an occurrence which starts at time T. For each such pattern α, the pair $(id(\alpha), I_{\alpha,T})$ is stored in $\tau_t[T]$. Moreover, for efficient lookup of the index nodes the nodes are indexed using a B^+-tree with T as the index key. Figure 4 shows an instance of a *TFCMP-tree.*

5.2 MIF Query Processing Algorithm

A naive way to answer time-slice pattern queries is to sequentially read patterns from F and check which ones are frequent during the given time-interval $Q_T = [t_s, t_e]$. However, when F is large, using this approach causes many I/O requests which is time consuming. Also, for each pattern α, $occ(\alpha)$ must be enumerated to see if it is frequent during Q_t or not. To address these issues, we propose the *Minimum Interval Frequency (MIF)* query processing algorithm using the *TFCMP-tree* in order to efficiently retrieve co-movement patterns that are frequent during Q_t.
First, the *MIF* algorithm needs to find $t_1 = t_s$ (or the earliest timestamp after t_s if t_s does not exist in τ_t). In order to speed this up, τ_t contains a B^+-tree that is constructed on timestamps (see Figure 4). After finding t_1, for each $t_j \in [t_1, t_e]$, the *MIF* algorithm checks the elements of $\tau_t[t_j]$. For each element $(id(\alpha), I_{\alpha,t_j})$, if $I_{id(\alpha),t_j} \leq t_e$ then α is frequent in Q_t and $id(\alpha)$ must be added to the result set p. After finding the id of all frequent co-movement patterns in Q_t, actual patterns must be fetched from F. To reduce the fetching time MIF uses F', which contains the same set of patterns as F but patterns are sorted based on their ids.
For example, for the given *TFCMP* index τ_t in Figure 4 if $Q_t = [2,13]$, t_1 will be 4, since $t = 2$ does not exist in τ_t. There are two elements in $\tau_t[4]$. $id(\alpha_1)$ will be added to p since $I_{\alpha_1,4} = 12 \leq 13$ but $id(\alpha_2)$ will not since $I_{\alpha_2,4} = 14 > 13$, which means that α_2 is not frequent during Q_t.
The details of the *MIF* algorithm is illustrated in Algorithm 2. The input to the algorithm is a time-interval Q_t and the output is P which is the set of all co-movement patterns that are frequent during Q_t (Lines 1-2). First P and p are initialized to $\emptyset$ (Line 3). Then the *Find* method is called finding t_1 which will be set to t_s if t_s exist in τ_t. Otherwise, it will be set to the nearest timestamp after t_s that exists in τ_t (Line 4). Then for each t_i, such that $t_1 \leq t_i \leq t_e$, all $(id(\alpha), I_{\alpha,t_i}) \in \tau_t[t_i]$ are checked and if $I_{id(\alpha),t_i} \leq t_e$, then $id(\alpha)$ is added to set of frequent co-movement pattern ids p (Lines 6-8). Then, all frequent patterns whose ids are in p, will be fetched from F' using the *Fetch* method and added to P (Line 9). Finally, P is returned (Line 10).

Algorithm 2 MIF Query Processing Algorithm

```
1: Input: $Q_t = [t_s, t_e]$: a given time-interval
2: Output: $P$: all frequent co-movement patterns during $Q_t$
3: $P \leftarrow \emptyset, p \leftarrow \emptyset$
4: $t_1 = \tau_t.Find(t_s)$
5: for $t_i = t_1 : t_e$ do
6:     for each $(id(\alpha), I_{\alpha,t_i}) \in \tau_t[t_i]$ do
7:         if $(I_{\alpha,t_i} \leq t_e)$ then
8:             $p \leftarrow p \cup id(\alpha)$
9: $P \leftarrow Fetch(F', p)$
10: return $P$
```

6 Spatiotemporal Pattern Queries (STRPQ)

One approach to answer a spatiotemporal pattern query $Q_{ST} =< Q_r, Q_t >$ is to perform the spacial range pattern query first (as explained in Section 4) following by a frequency count for each pattern that meets the spatial criteria to check whether they are frequent during Q_t or not. We propose the *LM-MIF* query processing algorithm which uses both of the two index structures proposed in Sections 4 and 5 to process spatiotemporal pattern queries.

6.1 LM-MIF Query Processing Algorithm

For a given spatiotemporal query Q_{ST}, first the *LM-MIF* finds all patterns that are completely contained in Q_r using the *LM* algorithm described in Section 4. The results of this step is stored in P_1. Next, it finds the id of patterns that are frequent during Q_t using the MIF algorithm and stores them in P_2. Then, it returns patterns in P_1 that their ids exist in P_2 and there is no need to perform the frequency count for patterns in P_1. Note that the actual patterns are no longer fetched by MIF since they are fetched by LM.

7 Experiments

In this section, first we discuss our experimental setup in Section 7.1. The results of our empirical performance evaluations of the index construction and query processing with the proposed *FCPIR* framework are provided in Sections 7.2 and 7.3, respectively.

7.1 Experimental Setup

The *FCPIR* framework is implemented in C# and experiments are carried out on a workstation with Intel Core-i7 3.6GHz CPU and 16GB of memory, running Windows 10.
We tested the *FCPIR* framework using the Porto Taxi Dataset [5], which is a

real dataset and contains trajectories of 442 taxis traveling in the city of Porto, in Portugal. The dataset was captured during a period of 11 months and includes many features such as latitude and longitude, call type, date, etc.
We used the methods proposed in our prior work [2] to mine all frequent co-movement patterns with $\mu = 15$, $T_{MAX} = 8$, and $n = 30$ from the aforementioned dataset. Then subsets of this $FCPD$ dataset with different number of patterns were randomly selected to be used as input for our experiments.

7.2 Index Construction

In this section, we study the performance of the *TFCMP* and *SFCMP* trees by measuring their construction times and their compression ratio to the size of corresponding $FCPD$, as the number of patterns in the dataset grows. The results of this experiment are provided in Figures 5-a and 5-b. In both charts the x-axis shows the number of patterns in the given $FCPD$, while the primary and secondary y-axis in Figure 5-a show the construction time (in milliseconds) for TCMP and TSPQ respectively.; finally, the y-axis in Figure 5-b shows the compression ratios of the *TFCMP* and *TSPQ* trees to the size of given $FCFD$ in percentage.

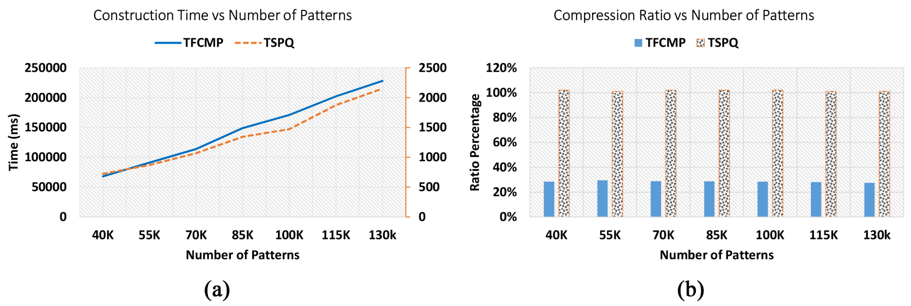

Fig. 5. (a) construction time in milliseconds and (b) compression ratio percentage vs. number of patterns in the dataset

As it can be observed in Figure 5-a, the construction times of both index structures grow linearly with the number of patterns in dataset, as expected. The construction time of the *SFCMP-tree* is smaller than that of the *TFCMP-tree* up to 10 orders of magnitude, since it reads patterns sequentially from the dataset and computes their minimum frequency intervals while the *TFCMP-tree* needs to scan the dataset once for each region sequence in order to find all relevant patterns; hence more I/O time.
As illustrated in Figure 5-b, the compression ratio remains almost steady for both trees as the size of the input dataset grows. However, *TFCMP* consumes

less storage as compared to *SFCMP*, since it only stores unique region sequences, while *SFCMP* stores all occurrences alongside with repetitive pattern ids.

7.3 Query Processing

In this section we discuss the performance of the *LM*, *MIF*, and *LM-MIF* query processing algorithms in terms of number of required I/O operations to process queries; note that this metric is proportional to the average query response time. For all experiments, we used an $FCPD$ with 130K patterns with an average support of 17 for a pattern.

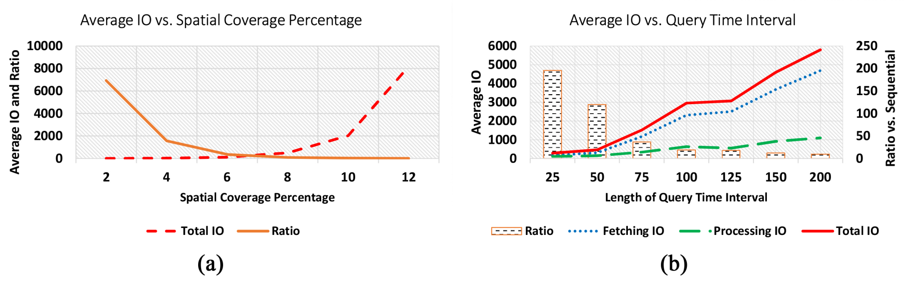

Fig. 6. (a) average I/O access vs. the length of query time interval, and (b) average I/O access vs. the size of spatial range query

7.3.1 SRPQ In this section we compare the performance of the *LM* query processing algorithm with that of the basic approach, where all patterns are scanned and fetched from the disk in a sequential manner and, those patterns that are infrequent in Q_r are filtered out. The results for this experiment are illustrated in Figure 6-a, where the x-axis shows the size of the query Q_r (in percentage of the total size of the region of interest for the given $FCPD$), while the y-axis shows the average number of I/O requests made by LM (the dashed red line) and its ratio to the number of I/O requests made by the basic approach (the solid orange line).
As, it can be observed in the figure, the number of I/O requests exponentially increases with the size of Q_r. This increase is mainly due to the *Subset* method since each generated sub-sequence of the Q_r region sequence must be looked up in the *TFCMP-tree* while processing the query.

7.3.2 TRPQ For this experiment, we varied the length of query time-interval $|Q_t|$ from 25 to 200 time units (0.5 to 20 percent of the total temporal length of the trajectories in dataset) and compared the number of I/O requests made

by the *MIF* algorithm with the basic approach, where all patterns are scanned and fetched from the disk, and those that are infrequent in Q_t are filtered out. The results of this experiment are illustrated in Figure 6-b, where the x-axis shows $|Q_t|$, while the primary y-axis (on the left) shows the average number of I/O requests made by *MIF*, and the secondary y-axis (on the right) shows ratio of the number I/O request between *MIF* and the basic approach. The solid red line shows the total number of I/O requests made by *MIF*; the breakdown of this cost to the number of I/O requests to find the frequent pattern ids using the *TFCMP-tree* (the dotted blue line), and the number of I/O requests caused by fetching those patterns from the disk (the dashed green line).
As it can be observed, the performance of the *MIF* algorithm is mainly determined by the number of fetch requests, which exponentially increases with $|Q_t|$. The reason is that as $|Q_t|$ increases, the number of patterns that become frequent increases as well; hence, more pages must be fetched from the disk. Moreover, as shown in Figure 6-b, the number of I/O requests grows gradually between $|Q_t| = 100$ and $|Q_t| = 125$. The reason is that in the used $FCPD$ the number of frequent patterns for $|Q_t| = 100$ and $|Q_t| = 125$ are not significantly different.

7.4 STRPQ

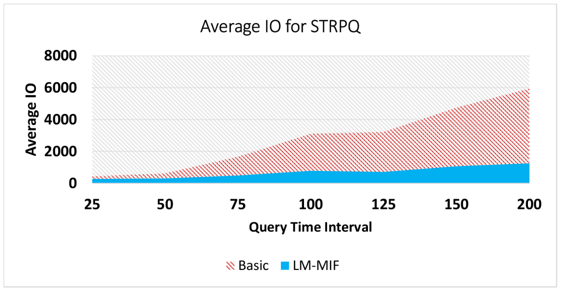

Fig. 7. Average I/O requests for STRPQ vs query time interval

In this section we compare the performance of the *LM-MIF* algorithm with the basic approach, where spatial and temporal range patterns queries are processed separately and then their results are merged together. The empirical results for this experiment are illustrated in Figure 7, where the x-axis shows $|Q_t|$, while the y-axis shows the average number of I/O requests made by the basic approach (the shaded red area) and the *LM-MIF* algorithm (the solid blue area). As shown in the figure, the number of I/O request made by the *LM-MIF* algorithm (the basic approach) grows gradually (exponentially). The reason is that the *LM-MIF* algorithm does not need to fetch actual patterns from the disk (as opposed to the basic approach) and the pattern ids can be used to filter out infrequent patterns from the result of spatial range pattern query; hence less I/O requests.

8 Conclusion and Future Work

In this paper, for the first time we defined range pattern queries on *frequent co-movement pattern datasets*, and proposed the *FCPIR* framework, including novel index structures and query processing algorithms, for efficient processing of various range pattern queries, namely, spatial range pattern queries, time-slice range pattern queries, and spatiotemporal range pattern queries.
We intend to extend this work in many directions. However, as the very next steps, we will introduce a hybrid tree structure by integrating *SFCMP* and *TFCMP* trees to reduce the storage requirement as well as the number of I/O requests to process spatiotemporal range pattern queries.

References

1. V. P. Chakka, A. C. Everspaugh, and J. M. Patel. Indexing large trajectory data sets with seti. *Ann Arbor*, 1001(48109-2122):12, 2003.
2. S. Helmi and F. Banaei-Kashani. Mining frequent episodes from multivariate spatiotemporal event sequences. In *Proceedings of the 7th ACM SIGSPATIAL International Workshop on GeoStreaming*, page 9. ACM, 2016.
3. S. Laxman, P. Sastry, and K. Unnikrishnan. A fast algorithm for finding frequent episodes in event streams. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 410–419. ACM, 2007.
4. N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 236–245. ACM, 2004.
5. L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas. Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1393–1402, 2013.
6. C. Mouza and P. Rigaux. Mobility patterns. *GeoInformatica*, 9(4):297–319, 2005.
7. A. Nanopoulos and Y. Manolopoulos. Efficient similarity search for market basket data. *The VLDB JournalThe International Journal on Very Large Data Bases*, 11(2):138–152, 2002.
8. F. Nori, M. Deypir, and M. H. Sadreddini. A sliding window based algorithm for frequent closed itemset mining over data streams. *Journal of Systems and Software*, 86(3):615–623, 2013.
9. D. Pfoser, C. S. Jensen, Y. Theodoridis, et al. Novel approaches to the indexing of moving object trajectories. In *Proceedings of VLDB*, pages 395–406, 2000.
10. M. A. Sakr and R. H. Güting. Group spatiotemporal pattern queries. *GeoInformatica*, 18(4):699–746, 2014.
11. Y. Tao and D. Papadias. Efficient historical r-trees. In *Scientific and Statistical Database Management, 2001. SSDBM 2001. Proceedings. Thirteenth International Conference on*, pages 223–232. IEEE, 2001.
12. U. Yun and G. Lee. Incremental mining of weighted maximal frequent itemsets from dynamic databases. *Expert Systems with Applications*, 54:304–327, 2016.