Spatiotemporal Range Pattern Queries on Large-scale Co-movement Pattern Datasets

Shahab Helmi, Farnoush Banaei-Kashani
Department of Engineering and Computer Science
University of Colorado Denver, Colorado, USA
Email: [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 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 the sports team, gait signature of the person, and driving behaviors causing heavy traffic). With our prior work, we proposed efficient algorithms to mine frequent co-movement patterns from trajectory datasets. In this paper, we focus on the problem of efficient query processing on massive co-movement pattern datasets generated by such pattern mining algorithms. Given a dataset of frequent co-movement patterns, various 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 grow very large, rendering naive execution of the pattern queries ineffective. In this paper, we propose novel index structures and query processing algorithms for efficient answering of two families of range pattern queries on massive co-movement pattern datasets, namely, spatial range pattern queries and temporal range pattern queries. Our extensive empirical studies with three real datasets have demonstrated the efficiency of the proposed methods.**

***Keywords*-spatiotemporal indexing; spatiotemporal query processing; pattern query;**

I. 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 consider these datasets as *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. With this terminology, we emphasize that moving object datasets can capture joint (but not necessarily coordinated) movements of multiple objects, such as players in sports teams competing on a field, joints in human body as a person walks, vehicles navigating a transportation network, ants in a colony as they follow their daily affairs, etc. Accordingly, in our prior work [6] and [5], we introduced efficient algorithms to mine *frequent co-movement patterns* from MVS datasets, where a frequent co-movement pattern is a pattern of movement for a set of objects repeatedly appears in various time windows. Such patterns are important to find and study in nowadays applications; for example, in sports analytics, tactics of sports teams can be uncovered by discovering frequent co-movement patterns from MVS datasets collected from players during games; 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 the traffic condition in the network; in human gait analysis, skeletal gait of an individual, where movement of each joint is captured by a variate in the corresponding MVS, can be analyzed to identify co-movement patterns of human joints that uniquely identify the gait of the individual, e.g., for gait-based identification.

Frequent co-movement patterns are often numerous due to combinatorial complexity of such patterns, and therefore, the mined frequent co-movement pattern datasets 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 frequent co-movement pattern datasets, co-movement pattern queries (or *pattern queries* for short). Various conventional (spatiotemporal) queries can be posed as pattern queries.

The previous work on querying frequent pattern datasets (e.g., [9] and [13]) are not applicable to the aforementioned pattern queries since they neither are developed for (spatial) trajectory data, nor allow temporal range pattern queries that should account for cases where some frequent patterns become infrequent during a given time-interval. Moreover, the existing solutions for querying mobility patterns (e.g., flock detection [19], co-location analysis [1], and spatiotemporal mobility queries [16]) also do not serve our purpose because they are essentially pattern mining solutions that are reduced to queries by assuming apriori knowledge about the spatiotemporal constraints of such patterns. However, we do not assume any apriori knowledge about co-movement

978-1-5386-2715-0/17/$31.00 ©2017 IEEE

patterns, and therefore, pattern queries must be applied to the previously mined co-movement patterns which are not constrained by any specific "form". In Section II, we review all relevant bodies of the literature in detail.
In this paper, for the first time:

- We formally define (co-movement) pattern queries.
- We introduce index structures and corresponding query processing algorithms for two specific variations of pattern queries, namely *spatial range pattern queries* and *temporal range pattern queries*, which are the fundamental types of queries used for exploration of pattern datasets. For example, in traffic behavior analysis, one can find and compare frequent driving patterns in downtown versus rural areas using spatial range pattern queries, while the frequent driving patterns in the whole city can be studied during the rush hours versus the regular hours using temporal range pattern queries. As another example, in sports analytics, the offensive and defensive tactics of a team throughout a soccer match and inside the penalty area can be retrieved using spatial range pattern queries, while the overall strategy of the team can be studied before an event (e.g., scoring a goal) using temporal range pattern queries.
- Finally, we conduct an extensive experimental evaluation using three real datasets and demonstrated that our proposed range pattern query processing solutions improve baseline solutions up to four orders of magnitude.

The remainder of this paper is organized as follows. Section II reviews the related work. Section III presents the necessary notations and formal definition of the problem. In Sections IV and V, we present index structures and query processing algorithms for spatial and temporal range pattern queries, receptively. Experimental results are provided in Section VI, and Section VII concludes the paper and discusses the future work.

II. Related Work

In this section, we review various types of related queries studied in the literature to our work with increasing order of relevance to pattern queries studied in this work.

A. Spatiotemporal Range Queries

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 and augments the traditional 2D R-trees (e.g., STR-tree [15]) with temporal information. 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 conflicts, 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 [17]). 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 with the cell (e.g., SETI [3]). In this paper, we study range queries to retrieve frequent co-movement patterns, from pattern datasets rather than range queries on trajectory datasets.

B. Similarity Search Queries on Point-Sets

In Section IV, we formally reduce the problem of spatial range pattern queries to the problem of *exact-match* in point-sets (or equivalently, region-sets). The work presented in [4] is the closest work in this category to our work, which introduces the problem of similarity search among sets of points in spatio-textual datasets. However, their proposed methods are not applicable to our problem since they aim at finding similar set of points while we are interested in exact-match. Moreover, their methods fail to address temporal range queries since they consider neither the temporal dimension of the data, nor the notion of temporal frequency, which is of prime significance in this work.

C. Interval Queries

In Section V, we reduce the problem of temporal range pattern queries to the problem of finding sets of time-intervals that are contained in a query time-interval. [8] and [18] propose similarity measures and efficient algorithms for finding similar interval sequences in a given dataset. However, they do not consider the frequency of the intervals, where in co-movement pattern mining, temporal frequency of the intervals must be considered. Moreover, they look for similar intervals in a dataset of time-intervals while we look for time-intervals that are fully contained in the query interval.

D. Mobility Pattern Queries on Trajectory Datasets

Most of the work in this body of literature is focused on mining *known* mobility patterns (such as co-location [1], flock [19], co-occurrence [2], moving cluster [7], etc.) from trajectory datasets. Such problems are often reduced to corresponding queries, where given the apriori knowledge about spatiotemporal constraints of such patterns, the proposed query solutions can benefit from significant search space reduction. However, we do not assume any apriori knowledge about co-movement patterns. In addition, co-movement patterns are not necessarily predefined to follow any form (shape) with specific constraints, nor pattern of movement for each individual object in a co-movement pattern must necessarily have any similarity to movement patterns of other objects. In [12], authors propose a query

language for mobility patterns. Similarly, in [16] 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 [16], 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 ([6] and [5]). 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.

E. Queries on Frequent Pattern Datasets

Data structures to index frequent pattern datasets are mostly dedicated to the problem of similarity search such as k-nearest pattern queries (e.g., see [13]), or retrieving subsequences of a given pattern form a set of frequent itemsets (e.g., [9]). However, they neither are developed for trajectory pattern queries, nor allow temporal range pattern queries which should account for cases where some frequent patterns become infrequent during a given time-interval. [10] is the closest work to our work in this category, in which authors propose an index structure for spatiotemporal range queries by re-purposing the traditional R-trees in order to reduce the number of disk accesses. However, the proposed approach only works when the given dataset is highly periodic, which significantly limits its applicability. Moreover, it only considers spatiotemporal intersection of patterns with the given spatiotemporal query range, and ignores temporal frequency of the patterns in the intersection. In this paper, we present the problem of pattern query processing for the first time and present efficient solutions for processing such queries along with extensive evaluation of these solutions with real datasets.

III. Problem Definition

Let $O = \{O_1, O_2, \ldots, O_n\}$ be a set of n moving objects; in Figure 1, $O = \{O_1, O_2, O_3, O_4, O_5, O_6\}$. Let S_i denote the trajectory of object O_i capturing the spatial path that it follows over time. S_i can be captured as a sequence of locations in space, ordered by time: $S_i = l_{t_1} l_{t_2} \ldots l_{t_m}$, where l_{t_j} shows the location of O_i at the j_{th} timestamp. Without loss of generality, we assume a discrete spatial model, namely a grid, which divides the entire region of interest R into smaller equisized cells/regions. With this model, the absolute locations of objects are replaced by their corresponding regions. For example, in Figure 1, the trajectory of Object 1 in discrete space is $S_1 = r_{20} r_{15} r_{10}$. It is important to note that we do not make any assumptions on the size of the grid in this paper; hence, our solution is applicable independent of the size of grid chosen by the user during pattern mining. Finally, we define a multivariate spatial event sequence dataset (*MVS*) as a set of n moving object trajectories denoting by $D = \{S_1, S_2, \ldots, S_n\}$.

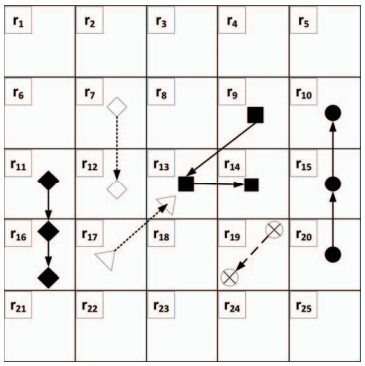

Figure 1. An MVS dataset including three co-movement patterns. α_1 is shown with solid lines including Object 1 (●), Object 2 (■), and Object 3 ♦; α_2 is shown with dotted lines including Object 4 (◇) and Object 5 (▷); α_3 is shown with dashed lines including Object 6 (⊗)

$occ(\alpha_1)$ = [2,4] [5,7] [12,14] [16,18]
$occ(\alpha_2)$ = [3,4] [12,13] [15,16] [20,21]
$occ(\alpha_3)$ = [2,2] [12,12] [16,16]

Figure 2. Occurrences of the co-movement patterns depicted in Figure 1

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 the rest of this section, we provide necessary definitions to define pattern queries on a given dataset of frequent co-movement patterns that are assumed to be previously mined from an MVS dataset using existing pattern mining solutions [5], [6].

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.
For example, $\alpha_1 = \{(r_{20} r_{15} r_{10})_{S_1}, (r_9 r_{13} r_{14})_{S_2}, (r_{11} r_{16} r_{16})_{S_3}\}$ is a co-movement pattern in Figure 1.

Definition 2 (Sub-Pattern and Super-Pattern):
We say α is a sub-pattern of the co-movement pattern α' (or α' is a super-pattern of α) denoted by $\alpha \subset \alpha'$, 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.
For example, if $\alpha = \{(r_{20} r_{15})_{S_1}\}$, $\alpha' = \{(r_{20} r_{15} r_{10})_{S_1}\}$, and $\alpha'' = \{(r_{20} r_{15} r_{10})_{S_1}, (r_9 r_{13} r_{14})_{S_2}\}$ then $\alpha \subset \alpha' \subset \alpha''$.

Definition 3 (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 D, if $\alpha \subset D$. We denote all

occurrences of α in D as $occ(\alpha) = \lambda_1\lambda_2...\lambda_f$, where $\lambda_j = [t_s, t_e]$ captures the start and end time of each occurrence. Note that, $T(\lambda_j) = \lambda.t_e - \lambda.t_s + 1$ captures the time-span of j_{th} occurrence. Finally, $sup(\alpha) = |occ(\alpha)|$ denotes the *support* of α in D.

The occurrences of α_1, α_2 , and α_3 in D are provided in Figure 2, where $sup(\alpha_1) = sup(\alpha_2) = 4$ and $sup(\alpha_3) = 3$.

Definition 4 (Valid Co-Movement Pattern):
To focus only on "interesting" co-movement patterns, 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 cannot 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/cells in their sequence, and iv) $\forall\lambda_j \in occ(\alpha)$, $T(\lambda_j) \leq T_{max}$, where T_{max} is user-defined maximum allowed time-span for a co-movement pattern.

The aforementioned validity conditions are natural and capture the features of a family of patterns that we are interested in studying in this paper. Studying other possible families of patterns is out of scope. Also note that the choice of conditions may affect mining of the patterns but not querying of the patterns, where the latter is the focus of this paper. Below we further elaborate on our choice of validity conditions. Condition *i* is self-explanatory. Condition *ii* ensures we only consider patterns across different objects. For instance, consider sports (e.g. soccer) analytics where a frequent co-movement pattern of a player with itself can be *"whenever the player goes from point A to point B, in the near future he will move from Point C to point D"*. Such a pattern is redundant, because $\{(AB)_{S_1}\} \longmapsto \{(CD)_{S_1}\}$ is an association rule and can be derived from the mined movement patterns of the player. Condition *iii* limits the patterns to those that include same size subsequences for each object included in the pattern to limit mined patterns to synchronous movements. 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. Going back to the soccer example, we are not interested in a co-movement pattern which captures the movement of Player 1 in the first minute of the game and captures the movement of Player 2 in the last minute of the game.

For the sake of simplicity, in the rest of this paper we use co-movement pattern and valid co-movement pattern interchangeably.

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 of n moving objects D, we assume F denotes the set of all frequent co-movement patterns mined from D. Accordingly, we denote each frequent co-movement pattern $p \in F$ be a ternary $p = \langle id, \alpha, occ(\alpha)\rangle$, where $id(\alpha)$ is a unique id assigned to the pattern α. We refer to the individual elements of p as $p.id$, $p.\alpha$ and $p.occ(\alpha)$, respectively.

Given the above definitions, we define the focus problems in this paper, i.e., spatial range pattern queries and temporal range pattern queries as follows.

Definition 6 (Spatial Range Pattern Query):
Given a dataset of frequent co-movement patterns F, a spatial range Q_r and a minimum support threshold μ, a Spatial Range Pattern Query (SRPQ), $Q_S = \langle F, Q_r, \mu\rangle$, returns a set P containing all co-movement patters $p \in F$ such that p frequently occurs in Q_r and p is fully contained in Q_r. Without loss of generality, we assume Q_r is rectangular.

Definition 7 (Temporal Range Pattern Query):
Given a dataset of frequent co-movement patterns F, a temporal range $Q_t = [t_s, t_e]$ which is a continuous time-interval, and a minimum support threshold μ, a temporal range pattern query (TRPQ), $Q_T = \langle F, Q_t, \mu\rangle$, returns a set P containing all co-movement patters $p \in F$ such that p frequently occurs during Q_t.

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

IV. Spatial Range Pattern Query

In this section, first we reduce the problem of Spatial Range Pattern Queries (SRPQ) to the problem of exact region-set matching, including a basic algorithm as baseline. Then, we propose two different algorithms that significantly improve the efficiency of the basic algorithm. A spatial query range Q_r can be mapped to a set of regions (from the grid superimposed on R) that are completely contained in Q_r (see Figure 3). Similarly, each pattern p can be mapped to a set of regions that are traversed by one or more sub-trajectories in $p.\alpha$. We denote this region-set of p as $p.r$. For example in Figure 3, $Q_r = \{r_7, r_8, r_9, r_{12}, r_{13}, r_{14}, r_{17}, r_{18}, r_{19}\}$ and $p_2.r = \{r_7, r_{12}, r_{13}, r_{17}\}$. Hence, with SRPQ we aim at finding all patterns in frequent co-movement pattern dataset F that have exact-matching regions-sets with Q_r, i.e., patterns whose regions-sets are subsets of Q_r. According to the aforementioned definitions, we formally re-define the SRPQ as follows:

Given a dataset of frequent co-movement patterns F, a region-set Q_r, and a minimum support threshold μ, a SRPQ $Q_S = \langle F, Q_r, \mu\rangle$ returns a set P containing all co-movement patters $p \in F$ such that $p.r \subseteq Q_r$ and $sup(p.\alpha) \geq \mu$.

A. Basic SRPQ Algorithm

The *Basic Spatial Range Pattern Query (B-SRPQ)* algorithm is straightforward. First, it finds C, which is the set all of patterns whose region-set intersect with Q_r, and C', which is the set of all patterns whose region-set intersect with $\overline{Q_r} = R - Q_r$. Then, the result-set P can be computed

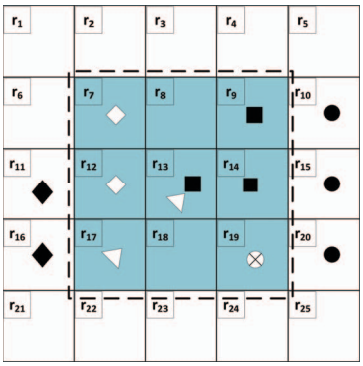

Figure 3. The MVS of Figure 1 where actual co-movement patterns are replaced with their region-sets. Q_r is shown with a dashed rectangle and its corresponding point-set is highlighted with shaded (blue) background.

as $C - C'$.

We formalized B-SRPQ below. Meanwhile, it is important to note that if the region-set of a frequent co-movement pattern is fully contained in Q_r, it can be inferred that all of its occurrences are also fully contained in Q_r. Moreover, with SRPQ (unlike TRPQ) support/frequency of patterns is irrelevant as all retrieved patterns will satisfy the user-defined support in the query.

To implement B-SRPQ, we maintain an inverted index F_R, where each key is a region $r \in R$ and its corresponding value (denoted as F_r) contains the set of pattern ids for patterns that contain r in their region-sets, i.e., $F_r = \{p.id \mid r \in p.r\}$. For example, in Figure 3 $F_{r_{13}} = \{p_1.id, p_2.id\}$.

As outlined in Algorithm 1, the B-SRPQ algorithm first generates the candidate set C by iterating through F_r for all $r \in Q_r$ and adding patterns ids that are in F_r to C (Lines 4-6). Next it generates C' by repeating the same process and adding those pattern ids that are in $F_{r'}$, where $r' \in \overline{Q_r}$ (Lines 7-9). Then it refines C by removing those patterns ids from C that exist in C', i.e., $C = C - C'$ (Line 10). Finally, the patterns whose ids remain in C are fetched from F (the dataset stored on disk) into P using the *fetch* function and P is returned (Line 11).

As an example, consider the given F and Q_r illustrated in Figure 3. B-SRPQ first goes through the shaded regions and adds $p_1.id$, $p_2.id$, and $p_3.id$ to C. Then it iterates through the remaining regions (white cells) and adds $p_1.id$ to C'. Afterwards, the difference of two sets are computed $C = \{p_1.id, p_2.id, p_3.id\} - \{p_1.id\} = \{p_2.id, p_3.id\}$ and $P = \{p_2, p_3\}$ is returned.

Algorithm 1 B-SRPQ

1: **Input**: $Q_S = \langle F, Q_r, \mu \rangle$
2: **Output**: P frequent co-movement patterns in Q_r
3: $P \leftarrow \emptyset, C \leftarrow \emptyset, C' \leftarrow \emptyset$
4: **for each** $r \in Q_r$ **do**
5: **for each** $p.id \in F_r$ **do**
6: $C.add(p.id)$
7: **for each** $r' \in \overline{Q_r}$ **do**
8: **for each** $p.id \in F_{r'}$ **do**
9: $C'.add(p.id)$
10: $C \leftarrow C - C'$
11: $P \leftarrow fetch(F, C)$, **return** P

B. Topological SRPQ Algorithm

The drawback of the B-SRPQ is that it needs to check all regions that are outside of the query range to filter out patterns that are not fully contained in Q_r. However, in most real-world applications, Q_r is small compared to R. Therefore, in this section, we introduce the *Topological Spatial Range Pattern Query (T-SRPQ)* algorithm to improve the complexity of the SRPQ processing.

Similar to the B-SRPQ algorithm, T-SRPQ first computes C, which contains the set of all pattern ids that intersect with Q_r. However, instead of computing C' by iterating through the regions that are not fully contained in Q_r, T-SRPQ checks the region-set $p.r$ for each pattern that its id exists in C; then, those patterns that are not fully in Q_r (i.e., $p.r \not\subseteq Q_r$) will be removed from C and the remaining patterns, which are exact-matching with Q_r, will be fetched into P from F. To expedite retrieving the region-sets $p.r$ for all patterns that are in C, we maintain a separate inverted index for patterns (denoted as φ), such that each key is a pattern id and its corresponding value, which is referred to by $\varphi_{p.id}$, contains the region-set $p.r$ for the pattern (see Figure 4(b)).

Moreover, if we maintain local topological information for each pattern in F_r, fetching $p.r$ from $\varphi_{p.id}$ can be avoided in two cases: i) if a co-movement pattern p is contained only in one region; or ii) if p is located in a boundary region of Q_r (that is adjacent to another region in $\overline{Q_r}$), we can use the topological information to determine if p is located in a region in $\overline{Q_r}$. The local topological information is stored as a quaternary $p.t = \langle l, t, r, b \rangle$, where flags l, t, r, and b are set to true if the region-set of the pattern p also covers a region r', such that r' is relatively placed on the left, top, right, or bottom of r, respectively. For example, for $p_1 \in F_{r_9}$ in Figure 3, l is true since p_1 covers r_{11} and r_{16}, which are relatively located on the left side of r_9; t is set to false, since the region-set of p_1 is not located in any region that is relatively located above the r_9. Also, for $p_3 \in F_{r_{19}}$, $p_3.t = \langle F, F, F, F \rangle$, since p_3 is fully contained in r_{19}. The T-SRPQ algorithm is outlined in Algorithm 2. It only iterates through F_r for $r \in Q_r$ (Line 4). For each pattern $p \in F_r$ it first checks if p is fully contained in r, i.e., $p.t = \langle F, F, F, F \rangle$ (Line 6). In this case $p.id$ is added to C. Otherwise, if r is a boundary region, the *out* function is called, which uses $p.t$ to determines if p is partially located outside of Q_r. If *out* returns true, then p can be discarded (Line 7). Otherwise, $\varphi_{p.id}$ must be fetched to determine if p is fully contained in Q_r. If it is, then $p.id$ is added to

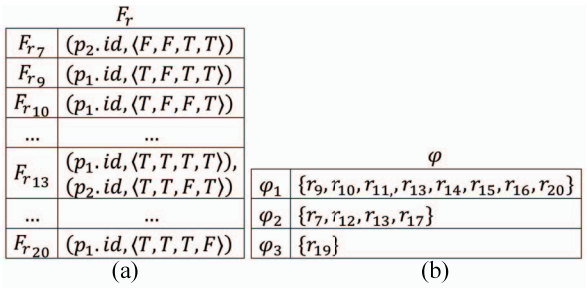

Figure 4. (a) Extended F_R and (b) φ for the pattern-set F of Figure 3

C (Lines 8-9). Finally, patterns that their ids are in C are fetched into P and P is returned (Line 10).

For example, in Figure 3 $p_3.id$ is added to C since all of its topological flags in r_{19} are false; p_1 is discarded since $out(p_3, r_9)$ is true. Since the containment of $p_2 \in Q_r$ cannot be decided using the topological information, $\varphi_{p_2.id}$ must be checked; since $p_2.r \subseteq Q_r$, $p_2.id$ is added to C. Finally, p_2 and p_3 are fetched from F and added to P.

Algorithm 2 T-SRPQ

1: **Input**: $Q_S = \langle F, Q_r, \mu \rangle$
2: **Output**: P frequent co-movement patterns in Q_r
3: $P \leftarrow \emptyset$, $C \leftarrow \emptyset$
4: **for each** $r \in Q_r$ **do**
5: **for each** $(p.id, p.t) \in F_r$ **do**
6: **if** $(p.t = \langle F,F,F,F\rangle)$ **then** $C.add(p.id)$
7: **else if** $(boundary(r) = T \wedge out(p.t, r) = T)$ **then** $discard\ p$
8: **else** {
9: **if** $(r \in Q_r \mid \forall r \in \varphi_{p.id})$ **then** $C.add(p.id)$ }
10: $P \leftarrow fetch(F, C)$, **return** P

C. Longest Match SRPQ Algorithm

Drawback of the T-SRPQ algorithm is that it does not consider the similarities among the region-sets of patterns. In this section, we present the *Longest Match SRPQ (LM-SRPQ)* algorithm that introduces a ranking among patterns which in turn defines locality of the patterns. LM-SRPQ leverages this pattern locality to effectively navigate and narrow down the search space in answering SRPQ in order. In particular, with the proposed ranking, the patterns that have similar region-sets, i.e., pattern that contains the same set of regions appearing in their movement patterns, will be co-located in the search space. Benefiting from this locality, for a given Q_r, LM-SRPQ can efficiently identify all pattern localities for the patterns that their region-sets are subsets of Q_r, i.e., $P = \{p | p \in F \wedge p.r \in \wp(Q_r)\}$, where $\wp(Q_r)$ shows the powerset of Q_r.

We decide the ranking of patterns as follows. First, we assume the region-set of a pattern $p.r$ to be an ordered set, where regions in $p.r$ are ordered based on a sorting criteria, e.g., their lexicographical order. Then, we assume $p_i \prec p_j$ if $|p_i.r| < |p_j.r|$; further, if $|p_i.r| = |p_j.r|$, then we apply

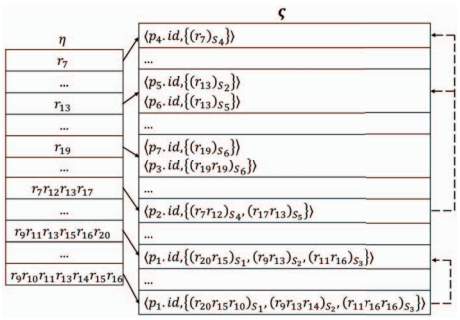

Figure 5. Examples of η and ς

the same sorting criteria used for regions in a region-set, e.g., the lexicographical order of the regions in the region-set. For example, given $p_i.r = r_1r_2$, $p_j.r = r_1r_2r_3$, and $p_k.r = r_1r_3r_4$, then $p_i \prec p_j$ since $|p_i.r| < |p_j.r|$ and $p_j \prec p_k$ since $|p_j.r| = |p_k.r|$ but $r_1r_2r_3$ comes before $r_1r_3r_4$ in the lexicographical order.

The LM-SRPQ index structure is defined based on the aforementioned ranking/organization of the patterns and consists of two data structures (see Figure 5). First, the index maintains an inverted index on region-sets in F denoted as ς, such that each key is a region-set κ, and its corresponding value $\varsigma_\kappa = \{\langle p.id, p.\alpha, p.r\rangle | p.r = \kappa\}$ (note that the keys of ς are omitted from the Figure 5 to save space). For a given Q_r, we need to look up all $\kappa \in \wp(Q_r)$ from ς. To speed up the lookup process, we also build a secondary index on top of ς, denoted as η, where η_κ points to ς_κ. The two structures ς and κ allow effective look-up of all patterns whose region-sets are subsets of $\wp(Q_r)$.

However, note that, the cardinality of $\wp(Q_r)$ exponentially grows with $|Q_r|$; hence, looking up all subsets of $\wp(Q_r)$ becomes time-consuming. The LM-SRPQ algorithm only looks up the longest region-set in $\wp(Q_r)$ (i.e., $\varsigma_{\kappa=Q_r}$) and then recursively looks up κ' such that κ' is a direct subset of κ (i.e., $\kappa \subset \kappa' \wedge |\kappa'| = |\kappa| - 1$). To further reduce the time-complexity of LM-SRPQ, we augment the ς data structure by adding pointers from parent region-sets to their immediate subsets. We call these pointers the secondary pointers of ς, which are shown with dashed arrows in Figure 5. Hence, as soon as a relevant region-set in $\wp(Q_r)$ is looked up and found in ς, all other qualifying patterns with subset region-sets can be located and fetched directly using the secondary pointers in ς.

The implementation of the LM-SRPQ algorithm is presented in Algorithm 3. First, Q_r is pushed into to the lookup queue q (Line 3). Then, until q is not empty, the head of q is popped into κ to implement the aforementioned recursive process (Line 5). η is used to checks whether κ exists in ς (Line 6). If it does, the *fetch* function is used to recursively retrieve patterns in $\varsigma_{\kappa'}$ such that $\kappa' \in \wp(\kappa)$. If κ does not exist in ς, then LM-SRPQ calls the *subset* function, which generates all direct subsets of κ, and pushes them to q (Line 7). Finally, P is returned (Line 8).

For example, in Figure 5 given $\kappa = Q_r = \{r_7, r_{12}, r_{13}, r_{17}\}$,

the LM-SRPQ algorithm first uses η to find ς_κ and fetches its patterns; then, using the secondary pointers in ς, the direct subsets of κ, which are $\{r_7\}$ and $\{r_{13}\}$, are located and their corresponding patterns are fetched. Now assume $\kappa = Q_r = \{r_{19}, r_{30}\}$. Since $\kappa = \{r_{19}, r_{30}\}$ does not exist in η, LM-SRPQ looks up $\{r_{13}\}$ and $\{r_{19}\}$ in η. $\{r_{19}\}$ exists in η, therefore its corresponding patterns are located and fetched from ς. On the other hand, $\{r_{30}\}$ neither exists in η nor has a further subset; hence it is discarded.

Algorithm 3 LM-SRPQ

1: **Input:** $Q_S = \langle F, Q_r, \mu \rangle$
2: **Output**: P: all frequent co-movement patterns contained in Q_r
3: $P \leftarrow \emptyset, q.push(Q_r)$
4: **while** $(q \neq \emptyset)$ **do**
5: $\quad \kappa \leftarrow q.pop()$
6: $\quad$ **if** $(\kappa \in \eta)$ **then** $P \leftarrow P \cup fetch(\varsigma_{\kappa'}) | \kappa' \in \wp(\kappa))$
7: $\quad$ **else** $q.push(subset(\kappa))$
8: **return** P

V. Temporal Range Pattern Query

In this section, first we reduce the problem of Temporal Range Pattern Queries (TRPQ) to the problem of finding the intervals in an interval-set that are contained in a query interval. Then we will propose two algorithms to this problem which significantly improve upon the basic algorithm (that is also briefly discussed in Section V.A).

Note that each occurrence $\lambda \in p.occ(\alpha)$ is a time-interval; hence, $p.occ(\alpha)$ is a set of time-intervals and accordingly, the set of occurrences of all patterns in F, is set of interval-sets. With TRPQ, we need to find those interval sets having at least μ time-intervals contained in the query interval Q_t. We formally define interval containment:

$$contain(I_1, I_2) = \begin{cases} 1, & \text{if } I_1.t_s \geq I_2.t_s \wedge I_1.t_e \leq I_2.t_e \\ 0, & \text{otherwise} \end{cases}$$

Using the aforementioned definition, we redefine the TRPQ as follows:

Given a dataset of frequent co-movement patterns F, a time-interval $Q_t = [t_s, t_e]$, and a minimum support threshold μ, a TRPQ $Q_T = \langle F, Q_t, \mu \rangle$ returns a set P containing all co-movement patters $p \in F$ such that the occurrence set of p contains at least μ time-intervals that are contained in Q_t, i.e.:

$$P = \{p | \Big(\sum_{i=1}^{|p.occ(\alpha)|} contain\big(\lambda_i \in p.occ(\alpha), Q_t\big) \Big) \geq \mu\}.$$

A. Minimum Support Interval TRPQ Algorithm

A basic TRPQ processing approach sequentially retrieves patterns from F, and for each pattern p, counts the number of its occurrences that are contained in Q_t. If the support of p in Q_t is greater than μ, then p is frequent during Q_t and is added to the result set. However, fetching all patterns and computing their support during Q_t is very time-consuming, if at all feasible. In this section, we propose the *Minimum Support Interval TRPQ (MSI-TRPQ)* algorithm which uses the concept of minimum support interval to avoid redundant computations.

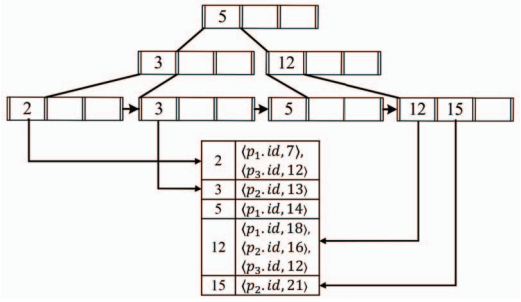

Figure 6. The τ of Figure 2 given $\mu = 2$

Definition 8 (Minimum Support Interval): Given a timestamp t_i, we denote the minimum support interval of a frequent co-movement pattern p at t_i as $p.min(t_i) = [t_j, t_k]$, where p is a frequent pattern in any interval $I = [t_i, t_k]$ if $t_j \geq t_k$, and p is infrequent in any interval $I' = [t_i, t'_k]$ if $t'_k < t_j$.

For example, in Figure 2 given $\mu = 2$ and $t_i = 10$, $p_1.min(10) = [10, 18]$, $p_2.min(10) = [10, 16]$, and $p_3.min(10) = [10, 16]$.

With the MSI-TRPQ algorithm, we maintain an inverted index on time τ, where each key is a timestamp t_i and its corresponding entry τ_{t_i} includes all patterns $p \in F$ that have an occurrence starting at t_i. For each such pattern, the binary value-set $\langle p.id, p.min(t_i) \rangle$ is stored in τ_{t_i}. The corresponding τ for the dataset depicted in Figure 2 is illustrated in Figure 6.

To process a TRPQ query $Q_T = \langle F, Q_t, \mu \rangle$, the MSI-TRPQ algorithm first finds t_b, which is equal to $Q_t.t_s$ if $Q_t.t_s$ exists in τ, and otherwise is equal to the earliest timestamp after $Q_t.t_s$ in τ. In order to reduce the lookup time for t_b, we apply a B^+-tree on top of the inverted index τ that is constructed on key values (timestamps) in τ. After finding t_b, for each $t_i \in [t_b, Q_t.t_e]$ in τ, the MSI-TRPQ algorithm retrieves τ_{t_i}. For each tuple $\langle p.id, p.min(t_i) \rangle \in \tau_{t_i}$, if $p.min(t_i) \leq Q_t.t_e$, then p is frequent during Q_t and $p.id$ is added to the result set P. After finding the id of all frequent co-movement patterns during Q_t, the actual patterns must be fetched from F, which is stored on disk. To reduce the fetch time, MSI-TRPQ also maintains an inverted index on F such that for each pattern $p \in F$ the key is $p.id$ and its corresponding value is $p.\alpha$. We denote this inverted index as F_p. The pseudocode for the MSI-TRPQ algorithm is provided in Algorithm 5. The algorithm is self-descriptive, but note that the *find* method finds and returns t_b (Line 4), and the *fetch* method fetches all patterns from F_p that their ids are in the candidate list C.

As an example, consider the τ in Figure 6. Given $Q_t = [4, 14]$, $t_b = 5$ since $\tau_4 \notin \tau$. Starting form τ_5, $p_1.id$ is added to C since $p_1.min(5) = 14 \leq Q_t.e = 14$. $p_2.min(12) = 16$ and $p_3.min(12) = 16$ in τ_{12}, which are both greater than $Q_t.e = 14$; hence, p_2 and p_3 are discarded. Finally, p_1 is fetched from F_p and returned.

Algorithm 4 MSI-TRPQ

1: **Input:** $Q_T = \langle F, Q_t, \mu \rangle$
2: **Output**: P: all frequent co-movement patterns in Q_t
3: $P \leftarrow \emptyset, C \leftarrow \emptyset$
4: $t_b \leftarrow find(\tau, Q_t.t_s)$
5: **for each** $t \in [t_b, Q_t.t_e]$ **do**
6: **for each** $\langle p.id, p.min(t_i) \rangle \in \tau_{t_i}$ **do**
7: **if** $(p.min(t_i).t_e \leq Q_t.t_e)$ **then** $C.add(p.id)$
8: $P \leftarrow fetch(F_p, C)$**, return** P

B. Reverse Support TRPQ Algorithm

Although the MSI-TRPQ algorithm is very efficient for processing temporal queries, index τ assumes a predetermined constant μ (which is usually set to the minimum support value used in the mining process). However, one may want to query patterns with different support values using the pre-generated pattern dataset that is mined from raw data given a fixed support. To this end, for each given μ a new τ must be constructed, or patterns must be mined using the new μ which is not efficient, if possible at all. To address this requirement, we propose the *Reverse Support TRPQ (RS-TRPQ)* query processing algorithm, which is similar to MSI-TRPQ, but benefits from the concept of reverse support, that results in an improved index structure which can be constructed and used independent of μ. In another words, we let the user query the given dataset with multiple values for μ, which is equal or higher than the same value used for mining co-movement patterns from an multivariate spatiotemporal dataset, while there is no need to mine patterns with the new μ or re-constructing a new τ per μ and desired pattern can be directly find from the index structure.

Definition 9 (Reverse Support): Given a timestamp t_i, we denote the reverse support of a pattern p at time t_i as $p.rs(t_i)$ and define it as the number of occurrences of p in F that start at or after t_i.

For example in Figure 2, $p_1.rs(3) = 3$ and $p_1.rs(10) = 2$.

For the RS-TRPQ algorithm we modify the index structure τ such that for each pattern $p \in \tau_{t_i}$, $\langle p.id, p.\lambda_{t_i}.t_e, p.ri(t_i) \rangle$ is stored, where $p.\lambda_{t_i}.t_e$. denotes the end time of the occurrence of p that starts at t_i. Moreover, suppose $p.cs(I)$ denotes the support of p in a given interval $I = [t_s, t_e]$. As outline in Algorithm 6, after finding t_b, RS-TRPQ iterates through all entries of τ from τ_{t_b} to τ_{t_e}. For each $\langle p.id, p.\lambda_{t_i}.t_e, p.ri(t_i) \rangle$ if $p.ri(t_i) \geq \mu$ and $p.\lambda_{t_i}.t_e \leq Q_t.t_e$, $\langle p.id, p.sc(t_i) = 1 \rangle$ is added to C, which is an ordered list based on pattern ids and each entry contains the current support of p in $[t_b, p.\lambda_{t_i}.t_e]$. If $p.id$ already exists in C, $p.ci(t_i)$ is not checked and its current support increases by 1 (i.e., $C[p.id] + +$) . Finally, patterns $p \in C$ having $C[p.id] < \mu$, are removed from C and the remaining entries are fetched from F_p into the result set P.

As an example, consider the occurrences provided in Figure 2. Given $Q_t = [4, 14]$ and $\mu = 2$, $t_b = 5$ since $\tau_4 \notin \tau$. Starting form τ_5, $\langle p_1.id, 1 \rangle$ is added to C since $p_1.ri(5) = 3 \geq \mu = 2$ and $p_1.\lambda_{t_5} = 7 \leq Q_t.t_e = 15$. For τ_{12}, $p_1.id$ is both in τ_{12} and C, then $C[p_1.id]$ increases by 1. Moreover, $p_2.id \in \tau_{12}$ and $p_2.ri(t_{12}) = 3 \geq 2$ and $p_2.\lambda_{t_{12}}.t_e = 13 \leq 14$; hence $\langle p_2.id, 1 \rangle$ is add to C. Similarly, $\langle p_3.id, 1 \rangle$ is added to C as well. Finally, p_2 and p_3 are removed from C since $C[p_1.id] = C[p_2.id] = 1 < \mu$ and p_1 is fetched from F_p and added to P.

Algorithm 5 RS-TRPQ

1: **Input:** $Q_T = \langle F, Q_t, \mu \rangle$
2: **Output**: P: all frequent co-movement patterns in Q_t
3: $P \leftarrow \emptyset, C \leftarrow \emptyset$
4: $t_b \leftarrow find(\tau, Q_t.t_s)$
5: **for each** $t_i \in [t_b, Q_t.t_e]$ **do**
6: **for each** $\langle p.id, p.\lambda_{t_i}.t_e, p.ri(t_i) \in \tau_{t_i}$ **do**
7: **if** $(p.id \in C)$ **then**
8: **if** $(p.\lambda_{t_i}.t_e \leq Q_t.t_e)$ **then** $C[p.id] + +$
9: **else if** $(p.ri(t_i) \geq \mu \wedge p.\lambda_{t_i}.t_e \leq Q_t.t_e)$ **then**
10: $C.add(\langle p.id, 1 \rangle)$
11: **for each** $\langle p.id, p.ci(Q_t) \rangle \in C$ **do**
12: **if** $(C[p.id] < \mu)$ **then**
13: $C.remove(p.id)$
14: $P \leftarrow fetch(F_p, C)$**, return** P

VI. Experiments

The CPM framework is implemented in C# and experiments were carried out on a workstation with Intel Core-i7 3.6GHz CPU and 16GB of memory running Windows 10. Below, we briefly describe the datasets used for our experiments. The selected datasets vary in number of objects included in the dataset (few vs. many), spatial extent of the dataset (small vs. large), and spatial sparsity of the datasets (sparse vs. dense).

- Soccer Dataset [14]: This dataset was captured during a soccer game in late 2013 at the Alfheim Stadium in Troms, Norway, and includes players' field position, heading, and speed, sampled at 20 Hz using the highly accurate ZXY sport tracking system.
- PortoTaxi Dataset [11]: This dataset contains trajectories of 442 taxis navigating the city of Porto in Portugal. The dataset was captured during a period of 11 months and includes many features including latitude and longitude, call type, date, etc.

We used the methods proposed in our prior work [5], [6] to mine all frequent co-movement patterns. The default parameter values for the mining phase are provided in Table 1, where $|O|$, $|S_i|$, $|R|$, $|\omega|$, denote the number of objects (trajectories) in D, number of data points for each trajectory, number of regions in the grid applied on R, and the minimum support threshold, respectively. Moreover, $|F|$, $\overline{\lambda}$, and $\overline{\kappa}$ denote the number of mined patterns from D, the average number of occurrences, and the average cardinality of the region-sets in the mined frequent pattern datasets F, respectively.

Table I
DATASETS FEATURES

D	$\|O\|$	$\|S_i\|$	$\|R\|$	μ	$\|F\|$	$\overline{\lambda}$	$\overline{\kappa}$
Soccer	7	2000	80	6	11K	10	5
PortoTaxi	50	1000	400	15	130K	17	7

A. Index Construction

In this section, we study the performance of the index structures proposed for pattern query processing by measuring their construction times as well as their compression ratio to the size of corresponding F, as $|F|$ grows. The results of this experiment are provided in Figures 7. In all charts, the x-axis shows $|F|$. In the top charts, the y-axis shows the construction time in milliseconds, while in the remaining charts it shows the compression ratios of the proposed index structures in percentage. All charts on the left side illustrate the results for the Soccer dataset, while charts on the right side show the results for the PortoTaxi dataset. Note that in these charts we refer to B-SRPQ, T-SRPQ, LM-SRPQ, MSI-TRPQ, and RS-TRPQ as *SB*, *ST*, *LM*, *MSI*, and *RI*, respectively.

As it can be observed in Figure 7, the construction time of ς for LM-SRPQ is significantly larger than the construction time for other index structures. The reason is that LM-SRPQ scans the dataset for each unique region-set once to find all patterns with the same region-sets. Moreover, the size of the T-SRPQ is significantly larger than other index structures, since T-SRPQ constructs and maintains two different data structures on top of F. Furthermore, RS-TRPQ consumes slightly more storage than MSI-TRPQ since it stores more information for each entry. Finally, the construction times (the compression ratios) grows linearly (remains steady) as $|F|$ increases, as expected.

B. Query Processing

In this section, we discuss the performance of the proposed query processing algorithms in terms of the number of required IO operations to process queries. Note that this metric is proportional to the average query response time and the actual time for query processing (in milliseconds) can be computed as follows: $|IO| \times (l + \frac{1}{tr})$, where $|IO|$, l, and tr denote the number of IO operations, average disk access-time, and disk transfer-rate, respectively. For

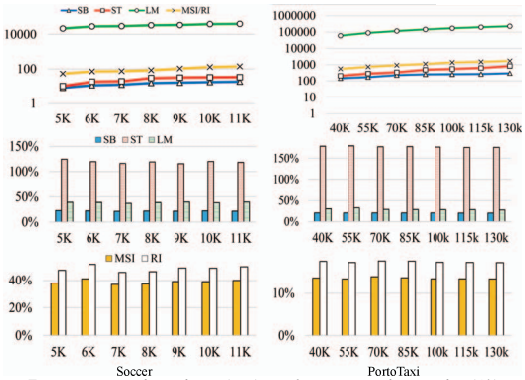

Figure 7. construction time (ms) and compression ratio (%) vs. $|F|$

example, with the workstation used for these experiments $tr = 260\ blockes/second$, and $l = 13.6\ ms$. Finally, for all of the following charts the y-axis shows the number of IO requests made by each algorithm, and the x-axis shows the value of the test parameter, where the rest of parameters are set to their default values provided in Table-1. In Sections VI.B.1 to VI.B.3, we evaluate sensitivity of the the proposed query processing algorithms to the size of query range, number of regions in the grid applied to R, and the minimum support threshold. Note that in these charts we refer to B-SRPQ, T-SRPQ, LM-SRPQ, MSI-TRPQ, and RS-TRPQ as *SB*, *ST*, *LM*, *MSI*, and *RI*, respectively.

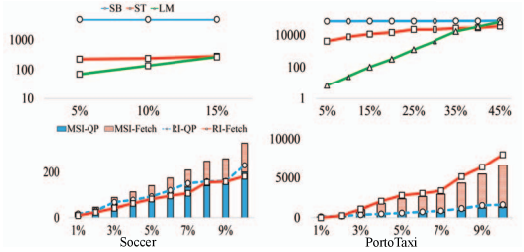

Figure 8. IO vs. $|Q_r|$ and $|Q_t|$

1) IO vs. $|Q_t|$ and $|Q_r|$: In this section, we evaluate the performance of the proposed query processing algorithms with those of the basic approaches as the spatial and temporal range of the query ($|Q_r|$ and $|Q_t|$) grow. The results for this experiment are illustrated in Figure 8. The x-axis shows $\frac{|Q_r|}{|R|}$ and $\frac{|Q_t|}{|S_i|}$ in percentage for SRPQs and TRPQs, respectively. For the TRPQ algorithms, the IO requests made for the query processing and by the *Fetch* function are shown with the *QP* and *Fetch* post-fixes, respectively.

As illustrated in Figure 8, LM-SRPQ outperforms B-SRPQ by four orders of magnitude, when $|Q_r|$ covers only 1% of $|R|$. However, the number of IO requests made by LM-SRPQ exponentially grows when Q_r becomes greater than $\overline{\kappa}$ due to significant increase in the number of subsets of Q_r that are to be looked up in τ, when the first lookup fails. The number of IO requests made by TS shows a gradual-linear growth with Q_r and outperforms LM-SRPQ for larger range queries.

The number of IQ requests made by MSI-TRPQ and RS-

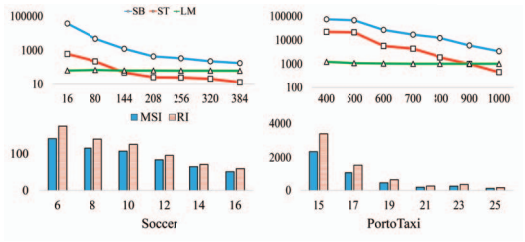

Figure 9. IO vs. μ and $|R|$

TRPQ linearly grow with Q_t for TRPQs. The reason is that they only need to iterate through $t_i \in [Q_t.t_s, Q_t.t.e]$. MSI-TRPQ slightly outperforms RS-TRPQ since it maintains less information but only works for a constant μ, while RS-TRPQ works for an arbitrary μ. For the soccer dataset, most of the IO requests are made by τ, while for the PortoTaxi dataset fetching patterns from F_p results in higher number of IO requests. The reason is that $|F|$ is significantly larger for the PortoTaxi dataset compared to the Soccer dataset; hence, more patterns are fetched from F_p. The number of IO requests made by the basic algorithm is 2972 and 55390 for the Soccer and PortoTaxi datasets, respectively.

2) IO vs. $|R|$: For this experiment, we varied the number of regions in the region of interest $|R|$ from 16 to 384 for the Soccer dataset, and from 400 to 1000 for the PortoTaxi dataset (which covers a larger space) and monitored the number of IO requests made by the proposed algorithms. Since TRPQ algorithms are barely sensitive to $|R|$, we provide the results only for the SRPQ algorithms in the top half of Figure 9. Increasing R causes an exponential decrease in the number of frequent patterns in the dataset $|F|$; hence, the number of IO requests in B-SRPQ and T-SRPQ sharply decrease as $|R|$ increases, as expected. However, the number of IO operations by LM-SRPQ remains almost steady. The reason is that as R increases, the search space of ς exponentially grows, resulting in significantly more number of lookups.

3) IO Access vs. μ: For this experiment, we varied the minimum support threshold μ from 16 to 26 for the Soccer dataset, and from 15 to 25 for the PortoTaxi dataset and monitored the number of IO requests made by the proposed algorithms. Since SRPQ algorithms are independent of μ, we provide the results only for the TRPQ algorithms in the lower half of Figure 9. The results for this experiment match our expectations as the number of IO operations exponentially decrease as μ increases. The reason is that the number of frequent patterns in F have a negative exponential co-relation with the minimum support, which causes the τ to shrink in size. Moreover, fewer number of patterns are required to be fetched from F_p.

VII. Conclusions and Future Work

In this paper, for the first time we defined range pattern queries on *frequent co-movement pattern datasets*, and proposed novel indexing structures and query processing algorithms for spatial range pattern queries and temporal range pattern queries. As shown in Section VI, our proposed algorithms outperform the basic solutions by 4 orders of magnitude.

We intend to extend this work in many directions. However, as the very next steps, we will introduce a hybrid tree structure by integrating the index structures proposed for spatial and temporal range pattern queries in order to efficiently process spatiotemporal pattern queries.

References

[1] S. Barua and J. Sander. Mining statistically significant co-location and segregation patterns. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1185–1199, 2014.

[2] M. Celik, N. Azginoglu, and R. Terzi. Mining periodic spatio-temporal co-occurrence patterns: a summary of results. In *Innovations in Intelligent Systems and Applications (INISTA), 2012 International Symposium on*, pages 1–5. IEEE, 2012.

[3] V. P. Chakka, A. C. Everspaugh, and J. M. Patel. Indexing large trajectory data sets with SETI. *Ann Arbor*, 1001(48109-2122):12, 2003.

[4] C. Efstathiades, A. Belesiotis, D. Skoutas, and D. Pfoser. Similarity search on spatio-textual point sets. In *EDBT*, pages 329–340, 2016.

[5] S. Helmi and F. Banaei-Kashani. Mining frequent episodes from multivariate spatiotemporal event sequences. Under Review.

[6] 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.

[7] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *International Symposium on Spatial and Temporal Databases*, pages 364–381. Springer, 2005.

[8] O. Kostakis, P. Papapetrou, and J. Hollmén. Artemis: Assessing the similarity of event-interval sequences. *Machine Learning and Knowledge Discovery in Databases*, pages 229–244, 2011.

[9] G. Liu, A. Suchitra, and L. Wong. A performance study of three disk-based structures for indexing and querying frequent itemsets. *Proceedings of the VLDB Endowment*, 6(7):505–516, 2013.

[10] 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.

[11] 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.

[12] C. Mouza and P. Rigaux. Mobility patterns. *GeoInformatica*, 9(4):297–319, 2005.

[13] 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.

[14] S. A. Pettersen, D. Johansen, H. Johansen, V. Berg-Johansen, V. R. Gaddam, A. Mortensen, R. Langseth, C. Griwodz, H. K. Stensland, and P. Halvorsen. Soccer video and player position dataset. In *Proceedings of the 5th ACM Multimedia Systems Conference*, pages 18–23. ACM, 2014.

[15] 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.

[16] M. A. Sakr and R. H. Güting. Group spatiotemporal pattern queries. *GeoInformatica*, 18(4):699–746, 2014.

[17] 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.

[18] A. Zheng, X. Zhou, J. Ma, and M. Petridis. The optimal temporal common subsequence. In *Software Engineering and Data Mining (SEDM), 2010 2nd International Conference on*, pages 316–321. IEEE, 2010.

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