8-2020

Querying recurrent convoys in a sliding window

Munkh-Erdene YADAMJAV

Zhifeng BAO

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

Farhana M. CHOUDHURY

Hanan SAMET

Citation

1

Querying Recurrent Convoys in a Sliding Window

MUNKH-ERDENE YADAMJAV, RMIT University, Australia
ZHIFENG BAO, RMIT University, Australia
BAIHUA ZHENG, Singapore Management University, Singapore
FARHANA M. CHOUDHURY, The University of Melbourne, Australia
HANAN SAMET, University of Maryland, USA

Moving objects equipped with location-positioning devices continuously generate a large amount of spatio-temporal trajectory data. An interesting finding over a trajectory stream is a group of objects that are travelling together for a certain period of time. We observe that existing studies on mining co-moving objects do not consider an important correlation between co-moving objects, which is the reoccurrence of the co-moving pattern. In this study, we propose the problem of finding recurrent co-moving patterns from streaming trajectories, enabling us to discover recent co-moving patterns that are repeated within a given time period. Experimental results on real-life trajectory data verify the efficiency and effectiveness of our method.

CCS Concepts: • **Theory of computation** → *Data structures and algorithms for data management*; • **Information systems** → *Data stream mining*.

Additional Key Words and Phrases: recurrent convoy query, co-moving pattern, spatio-temporal index

ACM Reference Format:
Munkh-Erdene Yadamjav, Zhifeng Bao, Baihua Zheng, Farhana M. Choudhury, and Hanan Samet. 2018. Querying Recurrent Convoys in a Sliding Window. *J. ACM* 37, 4, Article 111 (August 2018), 24 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

With the prevalence of location-positioning devices, a vast amount of spatio-temporal data of moving objects is being generated. Systematically analysing trajectory data of moving objects enables us to extract a variety of interesting patterns and knowledge that can lead to many real-life applications such as facility deployment [31] and urban computing [34].

One interesting finding in trajectory databases is the exploration of convoys [9]. Informally, a convoy refers to a group of spatially close-by objects moving together for a specific period of time. In essence, a convoy of interest is defined by the number of objects (τ) and the time duration of moving together (k), where τ and k are user-specified parameters.

A number of variations of the convoy have been proposed in the literature, which consider either offline [6, 7, 11, 15, 16, 28, 32] or online [2, 12, 14, 26, 33] trajectory data processing. The definitions and techniques to mine patterns of co-moving objects vary depending on the parameters and scenarios of consideration. However, existing literature on mining patterns of co-moving

Authors' addresses: Munkh-Erdene Yadamjav, RMIT University, Australia, munkh-erdene.yadamjav@rmit.edu.au; Zhifeng Bao, RMIT University, Australia, zhifeng.bao@rmit.edu.au; Baihua Zheng, Singapore Management University, Singapore, bhzheng@smu.edu.sg; Farhana M. Choudhury, The University of Melbourne, Australia, farhana.choudhury@unimelb.edu.au; Hanan Samet, University of Maryland, USA, hjs@cs.umd.edu.

0004-5411/2018/8-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

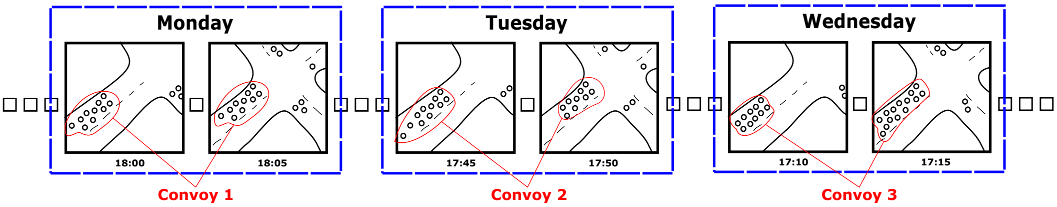


Fig. 1. A motivating example

objects usually treats each mined pattern independently while ignoring the possible correlation between them. Taking the correlations between convoys into consideration enables us to evaluate the importance of each convoy effectively. Moreover, in many real-life applications, an in-time analysis is required for incoming trajectory data [33]. Thus, it is critical to mine convoys that occur in the recent time window and take corresponding measures based on the historic occurrences of them. Figure 1 illustrates an example in transportation management application. The scenario shows data of traffic congestion occuring in a road segment during a weekday (i.e., occurrence of a convoy with more than a threshold number of vehicles in close-by distance), and similar congestion patterns being repeated during the other weekdays. These patterns are found to be instances of a pattern that reoccurs daily in terms of timespan, group size, and spatial closeness. Our aim is to find such recurrent convoys. A number of applications can benefit from exploring convoys with historic occurences in a sliding window. To name a few:

- **Scenario 1: Transport management system.** A transportation application can distinguish abnormal traffic congestions caused by accidents from recurrent regular traffic congestions during rush hours.
- **Scenario 2: Military surveillance.** A real-time military surveillance system can detect a co-moving pattern of suspicious groups with repeated occurrences.
- **Scenario 3: Parades and protests.** An occasional gathering of a larger number of people needs to be differentiated from common crowd of commuters.

Informally, a sequence of similar convoys forms a ***recurrent convoy***. The significance of a recurrent convoy varies w.r.t. the parameters, such as the number of objects that form the convoy ***(prominence)***, the duration of the convoy ***(timespan)***, and the time interval between two successive convoy occurrences ***(recurrence)***. The values of the parameters that define the interestingness of the convoy may change over time or domains/contexts. Therefore, the exploration of recurrent convoys of interest is an iterative process as the distribution of the parameters' values is not uniform across the whole search space. Setting appropriate values as a query input gives us a new insightful explanation about the dataset.

In this paper, given a sliding time window and thresholds for prominence, timespan, and recurrence, we study the problem of finding the *recurrent convoys in the sliding window* that satisfy the given thresholds. Our main focus is to propose a general approach to compute the similarity between convoys and to store them in a structure that facilitates the mining effort since the mining task that has a one-off parameter setting might not achieve the goal of extracting all interesting convoys.

The main challenges in identifying recurrent convoys in a trajectory database are three-fold. *First*, a convoy might not repeat itself at a regular pace, i.e., at the same timestamp with the same objects. The features that form the convoy may vary from time to time. As a result, the similarity metric that uses common objects to find the co-moving pattern in most of the related work [2, 6, 7, 11, 12, 14–16, 26, 28, 32, 33] is not suitable to find the correlation between convoys. *Second*, we can find a number of convoys in a sliding window that satisfy the query parameters. However, the number of times each result convoy is repeated w.r.t. the recurrence threshold is not known. Thus, the search for previous occurrences of each convoy within the recurrence threshold varies among the resulted convoys. An effective algorithm that only retrieves potential candidate convoys is required to speed up the mining task. *Third*, users might not know the proper value for each threshold to find the convoys of interest; it is an iterative process of exploration rather than a task of one-off parameter settings. Thus, we need an efficient indexing structure that facilitates the mining effort to explore the recurrent convoys of interest.

Recurrent convoy query was first proposed in a short paper [29]. In this paper, we extend it and make the following fresh contributions.

- We propose an indexing structure that organizes clusters effectively and efficiently [29] and develop an enhanced algorithm to mine convoys in a sliding window by using an in-memory lookup table (Section 4.2).
- We implement two similarity metrics to measure the cluster similarity and to evaluate their effectiveness on mining recurrent convoys (Section 5.3) and accelerate the similarity computation between convoys using the corresponding minimum bounding rectangles (Section 4.4.1).
- We conduct comprehensive experiments using real-life trajectory datasets to evaluate the efficiency and the effectiveness of our method (Section 5).

2 RELATED WORK

A number of approaches have been proposed to mine patterns of moving objects in a spatio-temporal database. Depending on the definition of a pattern, these approaches fall into two categories: *co-moving pattern* [6, 7, 9, 11, 15, 16, 28, 32] and *periodic pattern* [3, 10, 17–20, 35]. The studies of co-moving patterns generally differ in the way they compute the relationship between clusters of objects in a pattern. In addition, the co-moving pattern studies are not applicable to finding recurrent co-moving patterns because they consider each pattern as independent. More details on the co-moving pattern mining and their differences with our work are presented in Section 2.1. In contrast, periodic pattern mining techniques define a pattern as a sequence of regions that are visited by objects within regular time intervals (details in Section 2.2).

2.1 Co-moving pattern mining

A set of objects that move close-by for a certain period of time is considered as a co-moving pattern. Two main parameters that define the pattern are (i) the timespan of a pattern; and (ii) the number of objects that constitute a pattern. As we focus on both online and offline processing of the problem, we present the literature on co-movement pattern as: (i) offline co-moving pattern mining over historical trajectory data; and (ii) online co-moving pattern mining over steaming trajectory data.

2.1.1 Offline co-moving pattern mining. Many studies [7, 9, 15, 16, 23, 28] propose additional constraints on top of those two parameters, such as different spatial clustering techniques, local temporal consecutiveness, and temporal gap. Objects at each timestamp of a co-moving pattern are contained in a circle of a pre-defined radius in [7, 28], whereas a density-based clustering is used in other works. Timestamps in *swarm* [16] are not necessarily consecutive. A local temporal consecutiveness threshold is introduced to allow a temporal gap in a co-moving pattern [6, 15, 28]. In contrast to our problem, these works require objects in a co-moving pattern to be observed at all timestamps during database timespan. Orakzai et al. [23] proposed a sequential algorithm that clusters only convoy members corresponding to certain timestamps by pruning objects that are not part of the convoys. A moving cluster [11] does not often contain the same objects during its timespan. Thereby, a threshold for the percentage of common objects of two consecutive clusters is used to mine moving clusters. A gathering pattern is proposed in [32] where a set of objects called *dedicated members* travels for at least a certain period of time to be considered as a gathering.

The similarity of these works lies in the goal of discovering a set of objects moving together over a certain period, regardless of the parameters that have been considered. Co-moving patterns proposed in [7, 9, 11, 23, 32] discover patterns that occur at consecutive timestamps without any temporal gap between two consecutive clusters.

2.1.2 Online co-moving pattern mining. The streaming case of finding co-moving patterns in a trajectory database has been considered in [2, 12, 14, 25, 26, 33]. However, all these works consider different definitions of the co-moving patterns. Thus, each work designs its mining algorithms based on its specific definition of a co-moving pattern. Objects are required to record their locations at every timestamp throughout the database timespan in [26, 33]. However, we consider an object to record its location at every timestamp during its timespan.

2.2 Periodic pattern mining

A pattern is defined as a sequence of locations that are frequently visited by moving objects (e.g., taxis). It is noteworthy that each location in the sequence is represented as a region rather than an individual point. This representation is called a dense region [20], spatial region [3], or reference spot [17, 18]. Such a relaxation helps in finding patterns in a spatio-temporal database because an object is not likely to be at exactly the same location due to the limitations of GPS-equipped devices and other interferences that may affect the data collection process. A periodicity parameter is given as an input to segment the database to facilitate the mining process.

In [3, 20], the authors defined a periodic pattern as a sequence of spatial regions within T, where T is a time interval (e.g., day, week). The support of a pattern is defined by the number of periodic sequences in an object trajectory. In addition, a sequence of locations is divided into T spatial datasets and frequent patterns are mined. Finally, there is no time constraint between two consecutive occurrences of a periodic pattern. In contrast, we mine recurrent co-moving patterns that repeat themselves within periods of length T.

[17, 18] defined a periodic behaviour as an object visiting reference spots repeatedly. Reference spots are generated using a kernel method. Periodicity detection for each reference spot is performed by transforming a sequence of locations into a binary sequence. [10] defined a periodic pattern as a set of speed camera stations that are spatially close-by and exhibit the same periodic behaviours in terms of vehicle speed. Speed data is transformed into four discretized levels. Periodic behaviours for each station are detected by transforming discretized speed data into a binary sequence for each speed range. However, the occurrence of each convoy in a recurrent convoy differs, depending on the parameters of interest. Thus, it is time-consuming to apply methods proposed in [10, 17] for every query input over the filtered dataset to solve our problem.

Generally, a trajectory periodic pattern mines a sequence of locations that are periodically visited by a number of trajectories. In contrast, our problem defines a convoy to be generated by a set of objects that are spatially close-by at each of the k consecutive timestamps. Thus, our problem considers a densely populated area over time as a convoy whereas previously mentioned studies consider the correlation between different regions that are frequently visited by objects in the same order.

2.3 Other related areas

In addition to the aforementioned literature, there are other research areas that are weakly related to our work, such as distributed pattern mining [4, 6, 22, 24] and trajectory clustering [1, 13, 27]. Distributed co-moving pattern mining algorithms were proposed in [6, 22, 24] by using the

Notation	Definition
O^t	Set of objects whose locations are recorded at timestamp t
C^t	Set of clusters generated over O^t
c_i^t	Cluster with identifier i that is generated at timestamp t
W_I	Sliding window of length I
τ	Prominence threshold
k	Timespan threshold
ρ	Recurrence threshold
g	Convoy
$\mathcal{G}$	Set of convoys
$SIM()$	Similarity measure to compare convoys
p	Recurrent convoy
$\mathcal{P}$	Set of recurrent convoys

Table 1. Summary of notations

MapReduce framework to increase the efficiency and scalability of the mining process. Chen et al. [4] proposed a framework to mine co-movement patterns using Apache Flink that is designed to process data efficiently in a distributed manner. The trajectory clustering methods emphasize the spatial closeness of moving objects [27] and hence, the result cluster might contain moving objects that are not aligned w.r.t. the time dimension.

Remark. To the best of our knowledge, none of the existing work finds recurrent convoys since querying recurrent convoys over a sliding window considers both online and historic convoy generations simultaneously w.r.t. given thresholds.

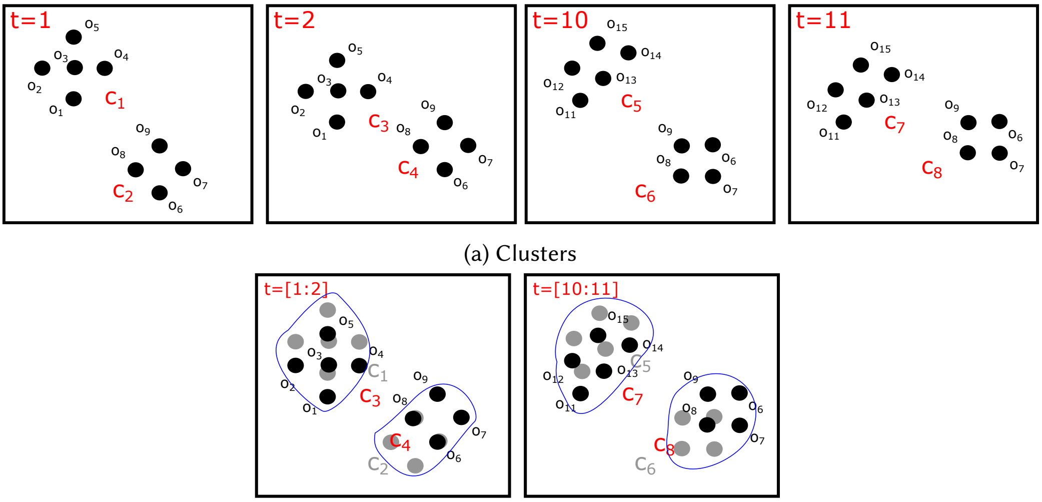
3 PROBLEM FORMULATION

In this section, we first present the necessary preliminaries and then give the formal problem definitions.

Given a set of moving objects $O = \{o_1, o_2, \ldots, o_{|O|}\}$ in a trajectory database with time domain $\mathcal{T} = \{t_1, t_2, \ldots, t_\infty\}$, a trajectory of moving object $o \in O$ is represented as a finite sequence of location samples within time interval $[t_i, t_j] \subseteq \mathcal{T}$, i.e., $o = \{loc_i, loc_{i+1}, \ldots, loc_j\}$, where loc_a is a recorded position of o in a two-dimensional space at timestamp t_a. We assume that the trajectory of object o is recorded at *every* timestamp during its lifetime $[t_i, t_j]$. Trajectories of different objects may have varying lengths.

Let $C = \{C^{t_1}, C^{t_2}, \ldots, C^{t_{|C|}}\}$ be the set of clusters generated by applying a chosen clustering algorithm over the trajectory database at different timestamps. Here, $C^t (\in C) = \{c_1^t, c_2^t, \ldots, c_{|C^t|}^t\}$ represents the set of clusters obtained at timestamp t, where $c^t \in C^t$ is a non-empty cluster of objects in O that satisfies the clustering conditions. Since we assume that each trajectory of an object only corresponds to a finite time interval $[t_i, t_j]$, it is possible that there is no cluster for some timestamps. Table 1 summarizes the frequently used symbols throughout the paper. A time-based query sliding window W_I of length I shifts at a time. A running example shown in Figure 2 is used to explain the definitions used in our problem formulation.

Example 1. Let the timespan of a trajectory database be $[1, 11]$ as shown in Figure 2a. Assume we find a total of eight clusters $C = \{C^1\{c_1, c_2\}, C^2\{c_3, c_4\}, C^{10}\{c_5, c_6\}, C^{11}\{c_7, c_8\}\}$ in the database at four different timestamps $\{t = 1, t = 2, t = 10, t = 11\}$ and no cluster is found from $t = 3$ to $t = 9$.



(a) Clusters

(b) Convoys

Fig. 2. A Running Example

The goal of our approach is to find the recurrent convoys that satisfy the thresholds given by the user. We adopt a well-recognized definition of convoy, originally defined by Jeung et al. [9], in our work.

Definition 1. ***(Convoy).*** Given a set of clusters C and thresholds for prominence (τ) and timespan (k), a convoy $g = \{c^{t_i}, c^{t_{i+1}}, \ldots, c^{t_j}\}$ is defined as a sequence of clusters at consecutive timestamps that satisfy the following constraints: (i) $\forall c^{t_a} \in g$, $\exists C^{t_a} \in C$ such that $c^{t_a} \in C^{t_a}$; (ii) the number of common objects shared by all clusters, denoted as $g.\tau$, is no less than τ, i.e., $g.\tau = |c^{t_i} \cap c^{t_{i+1}} \cap \cdots \cap c^{t_j}| \geq \tau$; and (iii) the time duration, denoted as $g.k$, is no less than k, e.g., $g.k = |t_j - t_i + 1| \geq k$, where $k > 1$.

Example 2. Let $\tau = 4$ and $k = 2$. Assume we generate the convoys that satisfy the given thresholds by using the clusters found in Example 1. We find four convoys as shown in Figure 2b: $\mathcal{G} = \{g_1\{c_1^1, c_3^2\}, g_2\{c_2^1, c_4^2\}, g_3\{c_5^{11}, c_7^{12}\}, g_4\{c_6^{11}, c_8^{12}\}\}$ that satisfy the thresholds. Note, we purposely include the timestamp t in each cluster c_i in the form of c_i^t to denote the timestamp that the cluster was formed.

Next, we define similar convoys w.r.t. the thresholds of interest in Definition 2. The boolean function SIM(c_1, c_2, δ) returns 1 if the similarity between two object clusters c_1 and c_2 meets the minimum similarity score δ. There are multiple similarity metrics available to quantify the similarity between two object clusters and the selection of the similarity metric is application dependent. For illustration purposes, we adopt the Hausdorff distance [21] to compute the similarity between clusters in convoys, similar to the trajectory pattern mining work by [32]. Nonetheless, the problem definition and our approach could be easily adjusted to other similarity metrics.

Definition 2. ***(Similar convoys).*** *Given the thresholds τ, k and δ and two convoys $g_a = \{c^{t_i}, c^{t_{i+1}}, \ldots, c^{t_{i+u-1}}\}$ and $g_b = \{c^{t_j}, c^{t_{j+1}}, \ldots, c^{t_{j+v-1}}\}$, convoy g_a is similar to convoy g_b w.r.t. τ and k iff (i) MIN($g_a.\tau$, $g_b.\tau$) $\geq \tau$; (ii) MIN($g_a.k$, $g_b.k$) $\geq k$; and (iii) $\exists g'_a = \{c^{t_a}, c^{t_a+1}, \ldots, c^{t_a+k-1}\} \subseteq g_a$, $\exists g'_b = \{c^{t_b}, c^{t_b+1}, \ldots, c^{t_b+k-1}\} \subseteq g_b$ such that $\forall l \in [0, k-1]$, SIM($g'_a.c^{t_a+l}$, $g'_b.c^{t_b+l}$, δ) = 1.*

As presented in Definition 2, two similar convoys contain at least τ objects and last at least k timestamps. Moreover, the corresponding clusters in two k-length subsequences of similar convoys satisfy the given similarity metrics. Now we are ready to introduce a *recurrent convoy* in Definition 3.

Definition 3. ***(ρ-Recurrent Convoy).*** Given a sequence of similar convoys $p_{i,j} = \{g_i, \ldots, g_j\}$ where each convoy satisfies the thresholds τ, k, and δ (by Definitions 1 and 2) and a recurrence threshold ρ, $p_{i,j}$ is a *ρ-recurrent convoy* iff $\forall a \in [i, j-1]$, two successive convoys $g_a, g_{a+1} \in p_{i,j}$ are similar and the difference between their starting timestamps is no larger than ρ, i.e., $|g_a.t_s - g_{a+1}.t_s| \leq \rho$ where $g.t_s$ denotes the starting timestamp of convoy g.

Example 3. Assume that we search for recurrent convoys w.r.t. $\tau = 5$, $k = 2$, and $\rho = 10$ *(shown in Figure 2b).* We found two convoys that satisfy the given thresholds, i.e., $g_1\{c_1, c_3\}$ and $g_3\{c_5, c_7\}$. If we assume that convoy g_1 is similar to convoy g_3, then they form a ρ-recurrent convoy $p_1 = \{g_1, g_3\}$, as g_1 and g_3 start at timestamps $t = 1$ and $t = 10$ respectively, and $g_3.t_s - g_1.t_s = 10 - 1 = 9 \leq \rho$.

Mining recurrent convoys is a one-off task from the data mining perspective. However, as we accumulate more data, the parameters that define the interestingness (timespan, prominence, recurrence) of the convoy might change over time. Thus, we present the *recurrent convoy query* over a sliding window in Definition 4 that takes varying parameters as input and only considers convoys in the sliding window (note, we skip parameter δ in Definition 4).

Definition 4. ***(Recurrent Convoy Query(RCQ)).*** Given a trajectory database that is continuously updated and a current sliding window of length I, the *RCQ*$\langle k, \tau, \rho\rangle$ finds a set of ρ-recurrent convoys $\mathcal{P}$, where $\forall p_i = \{g_a, \ldots, g_b\} \in \mathcal{P}$ satisfies the recurrence constraint ρ (by Definition 3), $\forall g_b$ is within the sliding window I, and $\forall g_j \in p_i$ is a valid convoy w.r.t. τ and k.

4 METHODOLOGY

In this section, we first outline the baseline steps to answer the *recurrent convoy query* and then present three enhancements to further improve the search performance.

4.1 Baseline

A general approach to answer the *recurrent convoy query* consists of three phases: (i) *cluster generation* that generates clusters using object locations at the current timestamp; (ii) *convoy generation* that extends existing convoy candidates in a sliding window using newly identified clusters or generates new convoy candidates from newly identified clusters; and (iii) *historic convoy generation* that searches for the historical occurrences of each convoy in a sliding window that satisfies the given thresholds. In the following, we explain these three steps in detail.

Cluster generation. Once we receive a set of objects whose locations are recorded at the current timestamp, we apply the chosen clustering algorithm on the object set to generate the object clusters. As we consider the streaming case of trajectory database where we need to process location updates immediately, we could not apply the proposed methods in the literature [9, 26] for convoy generation over historical data. However, the clustering step does not account for a substantial amount of the total query execution time compared with other parts of the recurrent convoy query. Thus, we leave the choice of the clustering algorithm to the user. For example, if we use DBSCAN [5], the clustering parameters *minPts* and ϵ are configured differently depending on the application requirement. Clusters that are obtained after the clustering step vary in the number of objects. Since τ is user-specified and unknown a priori, we define a parameter $\tau_{min} \leq \tau$ which serves as a lower bound of τ to accelerate the query processing. Clusters with at least τ_{min} objects are indexed using a traditional R-tree [8] structure in a 2-dimensional space, with each cluster c_i represented as a point of (t, $|c_i|$). Here, t refers to the timestamp of the cluster and $|c_i|$ refers to the number of objects in the cluster ($|c_i| \geq \tau_{min}$). The parameter τ_{min} is set once for each application.

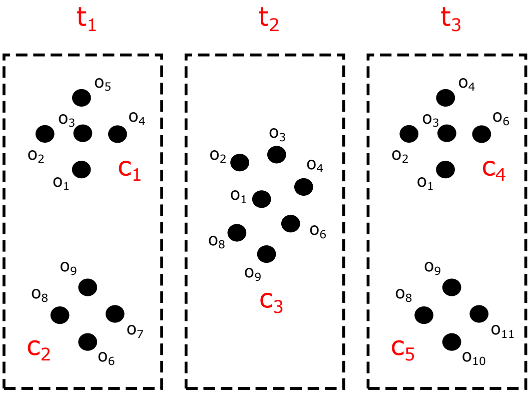


Fig. 3. Clusters

Example 4. Given a clustering algorithm, assume we find two clusters $C^{t_1} = \{c_1, c_2\}$ at timestamp t_1 as shown in Figure 3. If we set the value τ_{min} to 3 in our problem setting, both c_1 and c_2 are stored in the index as the clusters contain at least τ_{min} objects.

Convoy generation. Since the convoy clusters are consecutive in time, the incoming clusters at timestamp t are only compared to the candidate convoys that include clusters generated at timestamp $t - 1$. We are only interested in a convoy that contains τ number of common objects throughout its timespan. Thus, the incoming clusters are filtered by the number of objects and only the clusters that satisfy τ threshold are checked against the candidate convoys. In this way, the candidates are gradually refined and the ones that do not contain τ common objects are removed from the candidate set and added to the result set if the time-consecutiveness threshold k is satisfied. Note that all incoming clusters containing at least τ_{min} objects are passed to the indexing step, to support the discovery of recurrent convoys required by future queries.

Example 5. Assume we generate convoys w.r.t. thresholds $\tau = 4$ and $k = 3$ using the clusters obtained during the timespan of $[t_1 : t_3]$ in Figure 3. At timestamp t_1, we identify two clusters c_1 and c_2 and both clusters satisfy the threshold τ. Accordingly, we form two convoy candidates $g_1 = \{c_1\}$ and $g_2 = \{c_2\}$. At the timestamp t_2, we generate a new cluster c_3 which could extend the candidate convoy g_1 (i.e., $g_1 = \{c_1, c_3\}$ at t_2) but not g_2. Accordingly, candidate convoy g_2 is removed from the candidate set. At timestamp t_3, with the newly identified clusters c_4 and c_5, g_1 is further extended by cluster c_4 (i.e., $g_1 = \{c_1, c_3, c_4\}$ at t_3) and satisfies the timespan threshold $k = 3$. Consequently, g_1 becomes a result convoy. On the other hand, the newly identified cluster c_5 forms a new candidate convoy $g_3 = \{c_5\}$.

Historic convoy generation. Once we find the convoys of interest that fall inside the sliding window, we search for the previous occurrences of each convoy by checking historical clusters stored in the index w.r.t. the given thresholds. As we search for historic occurrences of each convoy within ρ time period, we load clusters within the time interval of length ρ at each iteration. Convoys are created using the same procedure that we described previously in the step of convoy generation based on the clusters w.r.t. the thresholds. The similarities between historical convoys and convoys in a sliding window are computed using the given similarity function and thresholds. Convoys in a sliding window that are extended by historic convoys retrieved within the time interval of length ρ are sent to the next iteration to check for another occurrence.

Discussion. We use the Coherent moving cluster (CMC) algorithm proposed in the original definition of the convoy [9] as a baseline algorithm. This algorithm is applicable in our problem setting, which considers streaming trajectory data. However, further optimizations proposed in [9] to reduce the clustering effort are not applicable in our problem setting, as the trajectory database is continuously being updated. In addition, the effort to generate the clustering based on the location updates reported by objects at timestamp t is not expensive as compared with the cost of the other two steps.

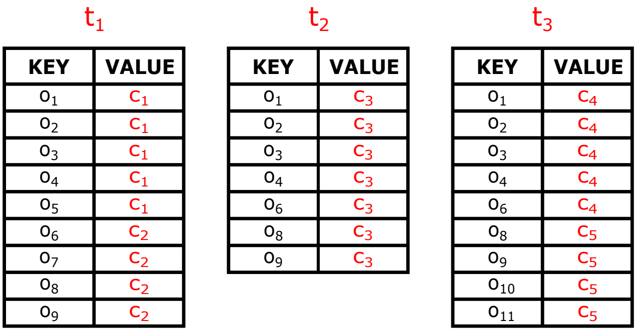
The computational overhead of the baseline to generate convoys is substantially high as the clusters at the current timestamp need to be checked against all candidate convoys that end at the previous timestamp. The historic convoy generation takes even more time if we set the recurrence threshold (ρ) to a higher value. To address the efficiency issue, we propose three enhancements with corresponding index structures over the baseline to accelerate the *recurrent convoy query* processing. To be more specific, we propose an in-memory lookup table to significantly accelerate the convoy generation, and two index structures, namely *intersection index* and *convoy index*, to speed up the historic convoy generation. These three enhancements are detailed below.

4.2 Lookup table

The computation cost of the convoy generation in a sliding window is significantly lower than the historic convoy generation over the clusters that fall within the specified time interval w.r.t. the given recurrence threshold. It only matches incoming clusters against the existing convoy candidates that end at the previous timestamp. However, we can improve the performance of the convoy generation algorithm by avoiding checking every incoming cluster against each candidate convoy.

Objects that are moving close-by are likely to be observed in the same cluster for a few consecutive timestamps rather than scattered into different clusters abruptly at the next timestamp [26]. Thus, we optimize the convoy generation algorithm by considering the overlaps between clusters at consecutive timestamps.

We create an in-memory lookup table that stores objects as a key and the corresponding cluster identifier as a value for incoming clusters at timestamp t as shown in Figure 4. By using the temporary lookup table at each timestamp, we compute the number of objects from each candidate convoy that are still observed in one cluster. If the number is no smaller than the threshold τ, the corresponding convoy is extended by the cluster. Clusters that do not extend any candidate convoys form new candidate convoys and hence are added into the candidate convoy set.



t_1

KEY	VALUE
o_1	c_1
o_2	c_1
o_3	c_1
o_4	c_1
o_5	c_1
o_6	c_2
o_7	c_2
o_8	c_2
o_9	c_2

t_2

KEY	VALUE
o_1	c_3
o_2	c_3
o_3	c_3
o_4	c_3
o_6	c_3
o_8	c_3
o_9	c_3

t_3

KEY	VALUE
o_1	c_4
o_2	c_4
o_3	c_4
o_4	c_4
o_6	c_4
o_8	c_5
o_9	c_5
o_{10}	c_5
o_{11}	c_5

Fig. 4. Lookup table states for different timestamps

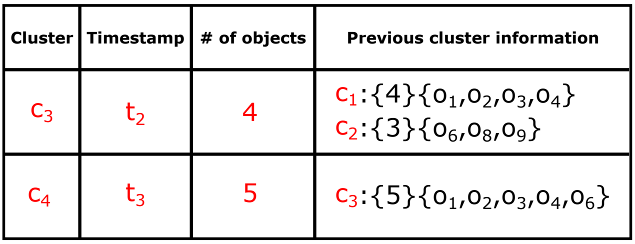
Example 6. Assume we generate convoys w.r.t. thresholds $\tau = 4$ and $k = 3$ using the clusters obtained during the timespan of $[t_1 : t_3]$ in Figure 3. Figure 4 shows the state of the lookup table for each timestamp. At timestamp t_1, we find two convoy candidates $g_1\{c_1 : [o_1, o_2, o_3, o_4, o_5]\}$, $g_2\{c_2 : [o_6, o_7, o_8, o_9]\}$ in a sliding window. Note, we do not use the lookup table at timestamp t_1 as there is no convoy in its previous timestamp. At timestamp t_2, we find out that four objects of c_1 still appear in the same cluster (i.e., c_3) based on the lookup table. Accordingly, g_1 is extended by cluster c_3 and becomes $g_1\{c_1, c_3 : [o_1, o_2, o_3, o_4]\}$. After four objects in c_3 are used to extend g_1, there are only three objects from cluster c_3 left for possible extension of candidate convoy g_2. However, there is no need to perform the intersection as the number of objects is smaller than $\tau = 4$. At timestamp t_3, g_1 is further extended by cluster c_4 using the lookup table. In addition, a new convoy candidate $g_3\{c_5 : [o_8, o_9, o_{10}, o_{11}]\}$ is created. As g_1 satisfies thresholds τ and k, the historic occurrences of that convoy are checked using the proposed index, as to be detailed next.

4.3 Intersection index

As soon as we find a convoy at the current timestamp that satisfies the thresholds τ and k, we check for historic occurrences of that convoy in the past within the recurrence threshold ρ. We can accelerate this process using an index on historical data.

We use a traditional R-tree [8] for the baseline algorithm to index each cluster, using its timestamp and the number of objects inside as the dimensions of the *Minimum Bounding Rectangle (MBR).*

The index enables us to filter clusters within a time interval of length ρ that contain at least τ number of objects in one operation for each query input. Retrieved clusters are fed into a convoy generation stage. However, a cluster with at least τ objects is not guaranteed to share at least τ common objects with any cluster at the next timestamp. To avoid checking each cluster against all candidate convoys, we propose an *intersection index* that considers the object overlaps between clusters at consecutive timestamps. To be more specific, a cluster c_i in the index stores all the clusters C' at the previous timestamp that share at least τ_{min} common objects with the cluster (shown in Figure 5 where $\tau_{min} = 3$) as data embedded in the node. One cluster might share τ objects with multiple clusters at the previous timestamp. Thus, we capture the maximum number of common objects between c_i and any cluster in C' using column "# of objects" in the *intersection index*. For our example in Figure 3, c_3 has 4 common objects with c_1, and 3 common objects with c_2, so 4 is captured by "# of objects" in the *intersection index*. This index has two advantages over the baseline index that stores clusters independently using R-tree only. First, it eliminates the extra check of two clusters at consecutive timestamps with τ number of objects if they do not share τ common objects. Second, each retrieved cluster is only checked against the convoys that end with a cluster in an embedded set of the previous timestamp. Figure 5 shows the information about embeddings in the *intersection index* for clusters in Figure 3. The *intersection index* is essentially



Cluster	Timestamp	# of objects	Previous cluster information
c_3	t_2	4	c_1:{4}{o_1,o_2,o_3,o_4} c_2:{3}{o_6,o_8,o_9}
c_4	t_3	5	c_3:{5}{o_1,o_2,o_3,o_4,o_6}

Fig. 5. Intersection index ($\tau_{min} = 3$)

an R-tree, hence the update process and the update costs are the same as that of the R-tree. Each cluster is inserted to the *intersection index* as soon as it becomes available.

Example 7. Assume five clusters (c_1 to c_5) observed from t_1 to t_3 are given in Figure 3 and $\tau_{min} = 3$. The *intersection index* contains two entries for clusters c_3 and c_4. Clusters c_1 and c_2 are not indexed as they correspond to t_1 without any previous timestamp, and cluster c_5, identified at timestamp t_3, is not indexed as it does not share at least $\tau_{min} = 3$ objects with any cluster identified at timestamp t_2 (e.g., it only shares two common objects with c_3, which is less than τ_{min}). For index entry of c_3, clusters c_1 and c_2 are embedded as each of them shares at least $\tau_{min} = 3$ objects with cluster c_3; for index entry of c_4, cluster c_3 is embedded as it shares 5 common objects with c_4, as shown in Figure 5. Assume we search for convoys w.r.t. $\tau = 4$ and $k = 3$. At timestamp t_2, we find a candidate convoy $g_1 = \{c_1, c_3\}$; at timestamp t_3, incoming cluster c_4 further extends g_1 to $\{c_1, c_3, c_4\}$ that satisfies the given thresholds.

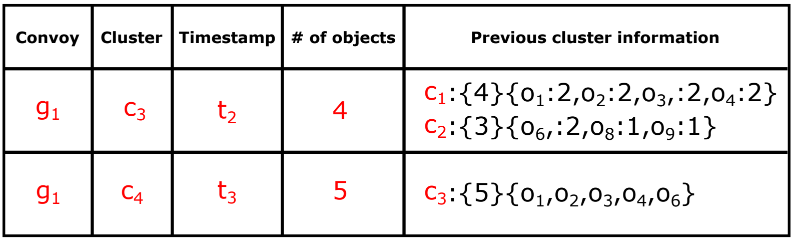
4.4 Convoy index

The embedding in the *intersection index* structure that we proposed previously avoids certain checking between the candidate convoys at timestamp $t - 1$ and clusters identified at timestamp t. However, we still need to intersect the clusters $k - 2$ times to generate a convoy with a length of k. If we search for convoys that contain a large number of objects, the intersection of clusters at consecutive timestamps still leads to poor performance. Moreover, the number of historical convoys that satisfy the query thresholds is likely to increase substantially when the recurrence threshold is set to a larger value. Thus, we propose another index structure to tackle this problem and to improve the efficiency of the historic convoy generation.

The two common scenarios of convoys are *converge* and *diverge*. As shown in Figure 3, objects in clusters c_1 and c_2 converge into cluster c_3 from t_1 to t_2 whereas objects in cluster c_3 diverge into two clusters from t_2 to t_3.

Based on this observation, we propose a *convoy index*, which groups incoming clusters into a set of distinctive convoys where each cluster is only assigned to one convoy. For each convoy, we compute and capture the number of consecutive appearances of each object in a cluster that belongs to the convoy. Each cluster is indexed with the embedded information about the corresponding convoy identifier and object timespans that appear in the convoy. The index convoys are generated w.r.t. the parameter τ_{min}. Thus, the convoys, generated for indexing purposes, contain at least τ_{min} objects. The timespan value for each object in a convoy indicates the consecutive time duration the object is observed in the convoy clusters. Splitting convoys at the diverging and converging timestamps enables us to assign a cluster to only one convoy, which leads to less information storage.

The *convoy index* enables a retrieval of clusters that satisfy both the time interval and number of common objects requirements similar to the *intersection index*. A sequence of retrieved clusters



Convoy	Cluster	Timestamp	# of objects	Previous cluster information
g_1	c_3	t_2	4	c_1:{4}{o_1:2,o_2:2,o_3,:2,o_4:2} c_2:{3}{o_6,:2,o_8:1,o_9:1}
g_1	c_4	t_3	5	c_3:{5}{o_1,o_2,o_3,o_4,o_6}

Fig. 6. Convoy Index ($\tau_{min} = 3$)

with the same convoy identifier is merged without performing actual intersections between clusters based on the objects' timespan information. The candidate convoy length is determined by the pre-computed timespan of objects w.r.t. the τ threshold, as shown in Figure 6.

Example 8. Let us generate convoys to be indexed that contain at least $\tau_{min} = 3$ common objects during their timespans using the clusters shown in Figure 3. Timestamp t_1 is a splitting point as clusters c_1 and c_2 converge into a cluster c_3. However, there is no previous timestamp for timestamp t_1 in the example. Thus, clusters c_1 and c_2 are embedded into the index entry of cluster c_3 w.r.t. τ_{min}, same as the *intersection index*. Although cluster c_3 diverges into two clusters c_4 and c_5 at the next timestamp, we only generate one convoy that contains at least $\tau_{min} = 3$ objects, that's $g_1[o_1, o_2, o_3, o_4, o_6] = \{c_3, c_4\}$. This is because clusters c_3 and c_5 only share two common objects, which is smaller than $\tau_{min} = 3$.

Once a convoy could not be extended by any incoming clusters, we compute the objects' consecutive appearances in the corresponding cluster and embed that information in the index as shown in Figure 6. We store objects in a cluster without timespan information if the cluster does not share enough common objects with any subsequent clusters w.r.t. the threshold τ_{min}.

Example 9. Assume we search for convoys w.r.t. $\tau = 4$ and $k = 3$. At timestamp t_2, we find a candidate convoy $g = \{c_1, c_3\}$ with objects o_1, o_2, o_3, and o_4. At timestamp t_3, we check the timespan value for each object and find out that the candidate convoy can be extended by one more cluster (i.e., c_4) that has the same convoy identifier in the index. We then add c_4 to the current candidate convoy $g = \{c_1, c_3, c_4\}$ satisfying the given thresholds without actually performing the intersection between convoy g and cluster c_4.

4.4.1 Similarity Computation. In case a threshold-based similarity metric is used for *recurrent convoy query*, we can accelerate the process of checking the similarity between two convoys by using the minimum bounding rectangles (MBRs) of the convoys. If the distance between the MBRs of two convoys that are being compared is greater than the given threshold δ, we can safely skip the similarity computation between the corresponding clusters in the convoys. No cluster in one convoy is within the distance threshold from any cluster in another convoy.

4.4.2 Index update. The *convoy index* is updated continuously with convoys generated by incoming clusters w.r.t. the threshold τ_{min} for indexing purpose. Here, the threshold τ_{min} represents the minimum number of objects that can be given as a query parameter. These convoys, namely *index convoys*, are different from the convoys generated w.r.t. the query parameters. At every timestamp, a set of index convoys stored in the main memory is checked against the newly generated clusters at the current timestamp w.r.t. τ_{min}. In order to assign a cluster to only one index convoy, we check the following conditions: (i) If a cluster c_i extends more than one convoy, we do not extend the corresponding index convoys by cluster c_i and generate a new index convoy with that cluster c_i; (ii) If an index convoy is extended by multiple clusters C', we do not extend that convoy by any cluster in C' as well and generate new index convoys with the corresponding clusters in C'. The index convoys that are not extended by any cluster at the current timestamp are inserted to the *convoy index* with the necessary timespan information for each object inside any cluster of the convoys. The extended or newly generated convoys at the current timestamp are stored in the main memory to check for possible extensions using the incoming clusters at the next timestamp. That means we run two different convoy generation algorithms, with one being query convoy generation w.r.t. the given threshold and the other for indexing purpose w.r.t. τ_{min}. The *convoy index* is only updated when we find an index convoy that is not extended at the current timestamp.

4.5 Algorithm

In the following, we present the *RCQ* processing algorithm over the streaming trajectory data using the *convoy index*. The query processing (i.e. Algorithm 3) includes two stages: (i) Existing convoy candidates that satisfy the prominence threshold are checked against the clusters generated at the current timestamp. Convoys that are extended and satisfy the timespan threshold (i.e. Algorithm 1) are passed to the second stage. (ii) Each result convoy received from the first stage is checked for its historic occurrences w.r.t. given time interval and thresholds using the *convoy index* (i.e. Algorithm 2). Thus, the historic convoy generation is executed only if we find convoys that satisfy τ and k in the sliding window.

Algorithm 1: Online convoy generation using a lookup table

```
     Input: set of convoys $G^{t-1}$ at timestamp $t-1$, clusters $C^t$ identified at timestamp $t$, and prominence
            threshold $\tau$
     Output: set of convoys $G^t$ extended at time $t$
1.1  $G^t \leftarrow \emptyset, C_{ext} \leftarrow \emptyset$
1.2  $lookupTable \leftarrow$ mapObjectsToCluster($C^t$)
1.3  foreach $g \in G^{t-1}$ do
1.4      $clusterMap \leftarrow \emptyset$
1.5      $O \leftarrow getConvoyObjects(g)$
1.6      foreach object $o \in O$ do
1.7          $c \leftarrow$ getObjectCluster($lookupTable$)
1.8          $clusterMap.push(c, clusterMap.get(c) + 1)$
1.9      foreach $pair(c, count) \in clusterMap$ do
1.10         if $count \geq \tau$ then
1.11             Update objects of $g$ with $O \cap C^t.get(c)$
1.12             Add $c$ to $C_{ext}$
1.13             Push $pair(g, c)$ to $G^t$
1.14 Add $C^t \setminus C_{ext}$ to $G^t$ as new convoys
1.15 return $G^t$
```

Algorithm 1 outlines the process of the convoy generation in a sliding window using the lookup table w.r.t. the given thresholds of τ and k. C_{ext} is declared to store the clusters from C^t that extend the candidate convoys in G^{t-1} (Line 1.1). First, a lookup table is created to store objects as a key and the corresponding cluster identifier as a value using clusters obtained at timestamp t (Line 1.2). Objects in each candidate convoy are grouped by the cluster identifiers obtained at timestamp t and stored in *clusterMap*. Note that, there is no cluster identifier for the object that is not recorded at the current timestamp t (Lines 1.3 - 1.8). *clusterMap* for each candidate convoy contains the cluster information of its objects at the timestamp t. As the convoy can be diverged into multiple convoys, we exam each cluster c in the *clusterMap* that satisfies the threshold τ, and add a pair of the candidate convoy g and the cluster c to G^t (Lines 1.9 - 1.13). Clusters that do not extend any candidate convoy are added as new convoys in the candidate set (Line 1.14).

Algorithm 2 presents the pseudo-code to generate historic convoys w.r.t. the given thresholds within the time interval $[t_{end} - \rho + 1, t_{end}]$, using the *convoy index*. Lookup table *skip* in Line 2.1 keeps track of convoy timespans that are currently under evaluation. The key in the lookup table indicates the convoy identifier while the value denotes the number of timestamps that convoy lasts. We retrieve sets of clusters ordered by the timestamp within the given time interval in Line 2.2. Each cluster set for a specific timestamp retrieved from the index contains a set of preceding clusters,

Algorithm 2: Historic convoy generation using the convoy index

```
Input: R-tree index $tree$, prominence threshold $\tau$, timespan threshold $k$, periodicity threshold $\rho$, time
       interval offset $t_{end}$
Output: set of historic convoys $S$
2.1  $skip \leftarrow \emptyset, G \leftarrow \emptyset$
2.2  $\cup(C^t, t) \leftarrow retrieveClusters(tree, \rho, t_{end}, \tau)$
2.3  foreach $C^t \leftarrow$ cluster set are accessed in ascending order of timestamp $t$ do
2.4      $newSkip \leftarrow \emptyset, G^t \leftarrow \emptyset$
2.5      foreach $c^t \in C^t$ do
2.6          foreach convoy $g \in G$ that intersects $c^t$ do
2.7              if skip contains $g$ then
2.8                  push $c^t$ to $g$, push $g$ to $G^t$
2.9                  $newSkip.put(g, skip.get(g) - 1)$
2.10         if $c^t$ extends nothing then
2.11             foreach previous cluster $c$ of $c^t$ do
2.12                 $g_{new} \leftarrow$ generate a convoy using clusters $c$ and $c^t$ if $\tau$
2.13                 push $g_{new}$ to $G^t$
2.14                 $count \leftarrow$ derive the value using the embedding of $c^t$
2.15                 $newSkip.put(g_{new}, count)$
2.16     $S \leftarrow S \cup \{g \in (G \setminus G^t) | timespan(g) \geq k\}$
2.17     $G \leftarrow G^t, skip \leftarrow newSkip$
2.18 return $S$
```

objects in common, and timespans of objects. We then generate the convoys from the clusters by evaluating the clusters according to ascending order of their timestamp t (Line 2.3). Assume we are currently evaluating cluster set C^t corresponding to timestamp t. For each cluster $c^t \in C^t$, we evaluate whether c^t could extend any candidate convoy g (preserved by G) under evaluation (Lines 2.5 - 2.6). For each retrieved convoy g that intersects c^t, if it is indexed by the lookup table *skip*, we know the common objects of g and their timespans with the help of *convoy index* and hence we can skip the detailed intersection operation between c^t and g. Without performing the intersection, we can expand the convoy g by c^t and preserve the expanded convoy g as one of the candidate convoys for the next iteration at $t + 1$ in G^t (Lines 2.7 - 2.8). In addition, we insert expanded convoy g as a new entry in *newSkip*, the lookup table to be used in the next iteration at timestamp $t + 1$ (Line 2.9). Note, the *count* value of g is decreased by one, to reflect the fact that g has been expanded by a cluster c^t corresponding to timestamp t and hence the remaining timespan of un-retrieved clusters is reduced. If cluster c^t does not expand any of the candidate convoys, we add that cluster with its previous clusters as new candidate convoys if they satisfy the threshold τ (Lines 2.10-2.15). Note that *count* value denotes the number of timestamps where at least τ objects of cluster c^t are observed in same cluster during subsequent timestamps (Lines 2.14). Those convoys not extended at timestamp t are added to the result set if satisfying k (Line 2.16), while convoys extended/generated by any cluster at timestamp t are passed to the next iteration at timestamp $t + 1$ (Line 2.17).

Algorithm 3 outlines the full process of the *recurrent convoy query* as the sliding window shifts at a time and a set of objects' location updates arrives. Location updates of the objects at the current timestamp t are clustered by a chosen clustering algorithm (Line 3.1). The generated clusters are filtered by the threshold τ. Existing candidate convoys G^{t-1} that end at the timestamp $t - 1$ are checked against the filtered cluster set C^t for a possible extension using the idea proposed in

Algorithm 3: Recurrent convoy query in a sliding window

```
     Input: sliding window $W_I$ of length $I$, R-tree index $tree$, convoys $G^{t-1}$ at timestamp $t-1$, objects $O_t$
            recorded at timestamp $t$, prominence threshold $\tau$, timespan threshold $k$, periodicity threshold $\rho$
     Output: set of recurrent convoys $P$
3.1  $C^t \leftarrow getClusters(O_t, \tau)$
3.2  $G^t \leftarrow generateConvoys(G^{t-1}, C^t, \tau)$ using Algorithm 1
3.3  $G \leftarrow filterConvoys(G^t, k)$
3.4  $t_{end} \leftarrow t - k, P \leftarrow \emptyset$
3.5  while $G$ is not empty do
3.6      $G_{remove} \leftarrow \emptyset$
3.7      $G_{past} \leftarrow getHistoricConvoys(tree, \rho, t_{end}, \tau, k)$ using Algorithm 2
3.8      foreach $g \in G$ do
3.9          $boolean$ found $\leftarrow false$
3.10         foreach $g_{past} \in G_{past}$ do
3.11             if $isSimilar(g, g_{past}, \tau, k)$ then
3.12                 push $(g, g_{past})$ to $P$, $found \leftarrow true$
3.13         if $found = false$ then
3.14             push $g$ to $G_{remove}$
3.15     $G \leftarrow G \setminus G_{remove}$
3.16     $t_{end} \leftarrow t_{end} - \rho$
3.17 return $P$
```

Section 4.2 (Line 3.2). Candidate convoys that satisfy the threshold k are added to a result convoy set G (Line 3.3). The variable t_{end} in Line 3.4 defines the upper bound of the time interval to search for historic convoys. Thereafter, we evaluate whether any convoy in the result convoy set G is actually a recurrent convoy satisfying the requirements specified (Lines 3.6 - 3.16). To be more specific, it initializes G_{remove}, a temporal set that stores the result convoys that no longer require any searching for reoccurrences in the next time interval (Line 3.6). It next retrieves historic convoys within the time interval $[t_{end} - \rho + 1, t_{end}]$ using Algorithm 2 over the *convoy index* (Line 3.7). It then evaluates the similarity between historic convoys and the result convoys in G (Lines 3.8-3.11) and adds historic occurences of each result convoy to the corresponding list (Line 3.12). If a result convoy is not extended by any historic convoys within the time interval of search, it is added to G_{remove} so it will not be evaluated in the next iteration (Lines 3.13-3.14). Thereafter, the result convoy set G is updated by removing those in G_{remove}, and the next time interval for historic convoy search is also updated by shifting another ρ timestamps (Lines 3.15-3.16). The process repeats until the result convoy set G becomes empty. Finally, P is returned to terminate the query (Line 3.17).

5 EXPERIMENTS

In this section, we compare our proposed algorithm with the baseline approach (presented in Section 4.1) through an experimental evaluation using real datasets. Further, we conduct a case study to show the effectiveness of our approach.

5.1 Experimental Settings

All algorithms are implemented in JAVA. Experiments were run on a 24 core Intel Xeon E5-2630 2.3 GHz using 256GB RAM, and 1TB 6G SAS 7.2K rpm SFF (2.5-inch) SC Midline disk drives running Red Hat Enterprise Linux Server release 7.5. We test the following methods to answer the recurrent convoy queries on real-life datasets: (1) **CMC:** The baseline *RCQ* algorithm consists two Coherent Moving Cluster [9] algorithms, where **CMC-S** and **CMC-H** are used to generate convoys in a sliding window and historic convoys on top of an R-tree index, respectively; (2) **CLT:** Convoy generation algorithm over a sliding window using a in-memory lookup table; (3) **RCI:** *RCQ* algorithm based on the *intersection index*; (4) **RCC:** *RCQ* algorithm based on the *convoy index*; and (5) **RCC+:** *RCQ* algorithm based on the *convoy index* with the optimized convoy similarity computation proposed in Section 4.4.1. Note that RCI, RCC, and RCC++ algorithms use CLT to discover convoys in a sliding window.

Datasets. All experiments were conducted using two real datasets, (i) T-drive dataset [30] and (ii) Beijing dataset. The T-drive dataset contains the raw trajectories of 10,357 taxis in Beijing, China, collected for a week in Feb 2008. The Beijing dataset contains 28,162 raw trajectories in Beijing, collected for a month in March 2009. Each trajectory in the dataset is a sequence of GPS locations (latitude and longitude) and the corresponding timestamps. We obtained clusters by running DBSCAN with the parameter settings listed in Table 2.

	T-drive	***Beijing***
Cluster density *minPts*	4	4
Cluster radius ϵ	100m	100m
# of cluster points	2,048,088	25,882,012
# of clusters	455,891	2,788,174
τ_{min}	4	5
Hausdorff distance threshold (δ_{HD})	100m	100m

Table 2. Dataset statistics

Query Generation. To ensure that at least one recurrent convoy is obtained as a result of the query, we generate all recurrent convoys that satisfy the maximum values of τ and k and the minimum value of ρ defined in Table 3. As we find multiple convoys that satisfy the query parameters, we randomly choose 100 of such queries for each dataset.

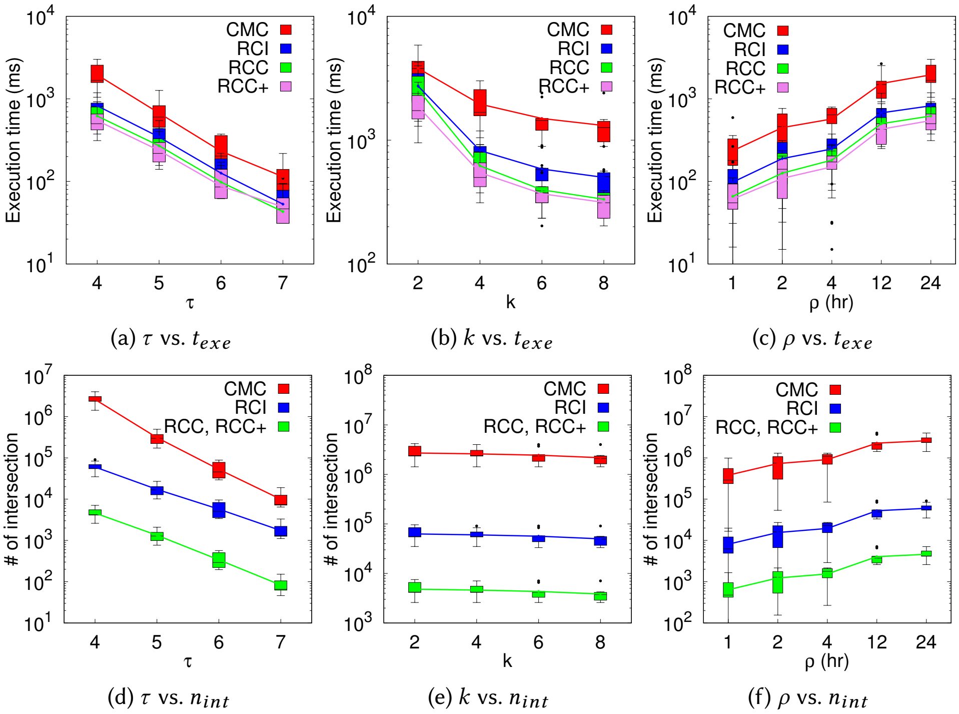
Evaluation and Parameterization. We compared the performance of the baseline and our proposed approaches by varying the query input parameters as shown in Table 3, where the values in bold represent the default values. The fanout of the R-tree index is set to 100. For all experiments, a single parameter is varied while other parameters are set to their default values.

Parameter	***Description***	***Dataset***	***Values***
τ	# of objects	T-drive	**4**, 5, 6, 7
		Beijing	**6**, 7, 8, 9
k	Timespan (sec)	T-drive	2 **4**, 6, 8
		Beijing	**6**, 7, 8, 9
ρ	Periodicity (hr)	Both	1, 2, 4, 12, **24**

Table 3. Experimental parameters

5.2 Efficiency Study

In this section, we conducted an experiment to evaluate the efficiency of our proposed algorithms against the baseline. We study the impact of each parameter by running 100 queries and report the average query execution time and the average number of intersections between convoy candidates and clusters evaluated, denoted as t_{exe} and n_{int} respectively. The performance for multiple runs is shown in boxplots, where the bounding box shows the first and third quartiles; the whiskers show



(a) τ vs. t_{exe}

(b) k vs. t_{exe}

(c) ρ vs. t_{exe}

(d) τ vs. n_{int}

(e) k vs. n_{int}

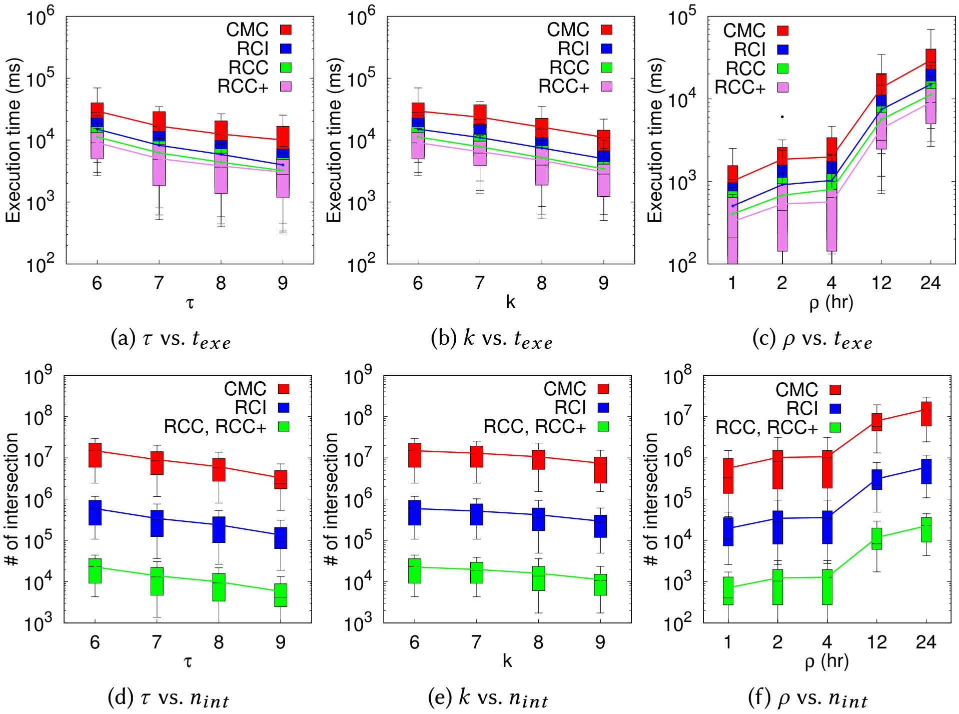
(f) ρ vs. n_{int}

Fig. 7. Effect of Varying Parameters on the T-drive Dataset

the range, up to 1.5 times of the interquartile range; and the outliers beyond this value are shown as separate points. The average values are shown as connecting lines.

Effect of τ. The effect of the parameter τ that controls the number of objects in a convoy on the query performance is presented in Figure 7a and Figure 8a, for the T-drive dataset and the Beijing dataset respectively. As we search for larger convoys by increasing the threshold τ, the overall query execution time decreases due to the distribution of objects in the clusters. Both RCI and RCC perform up to three times faster than the baseline. The performance gap between RCI and RCC shows a small margin, as shown in Figures 7d and 8d. RCC computes up to one order of magnitude fewer intersections than RCI. However, there is no substantial difference in the query performance, which is likely due to the distribution of the convoy sizes and lengths. RCC+ computes the similarity between convoys faster than other algorithms, resulting in shorter query execution time. However, they share the same number of intersections, so we use one line when reporting the number of intersections for RCC and RCC+.

Effect of k. The effect of the parameter k that controls the duration of the convoys on the query performance is presented in Figures 7b and 8b. As we increase the threshold k for the convoy, the number of historic convoys that satisfy the threshold declines, resulting in a smaller number of clusters to be retrieved from the *convoy index*. RCI and RCC perform up to five times faster than the baseline when varying timespan thresholds. The margin between RCI and RCC increases with the increase of k, leading to fewer intersections as shown in Figures 7e and 8e. Longer convoys have a higher chance of using information in the *convoy index* instead of performing intersections. This confirms that the convoy index works *better for searching convoys that last for longer timespans.*



(a) τ vs. t_{exe} (b) k vs. t_{exe} (c) ρ vs. t_{exe}

(d) τ vs. n_{int} (e) k vs. n_{int} (f) ρ vs. n_{int}

Fig. 8. Effect of Varying Parameters on the Beijing Dataset

Effect of ρ. The effect of the parameter ρ that controls the recurrence of the convoys on the query performance is presented in Figures 7c and 8c. As we increase the threshold ρ, the number of clusters that fall in the time interval of search increases, leading to longer time to generate convoys based on retrieved clusters. The number of historical convoys that satisfy the thresholds also increases, leading to longer query execution time. RCC performs up to four times faster than the baseline for varying settings of ρ threshold. The increase in the number of historical clusters leads to more intersections to be performed for historic convoy generation, as shown in Figures 7f and 8f.

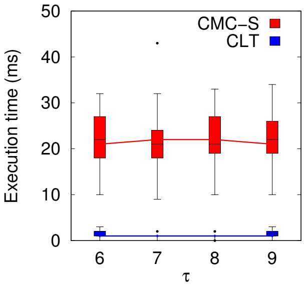
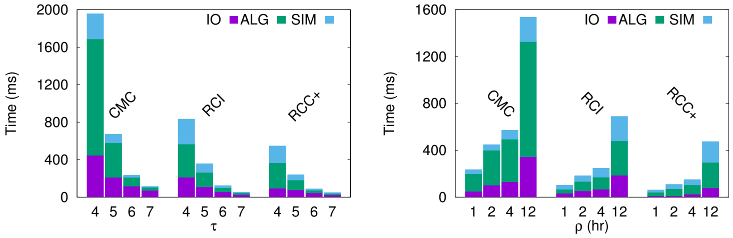


Fig. 9. Convoy generation in a sliding window on the Beijing Dataset

Online convoy generation. We report the query performance of **CMC** and **CLT** aglorithms for online convoy generations using the Beijing dataset in Figure 9. The Beijing dataset has more clusters per timestamp than the T-drive dataset and the clusters are dense in terms of the number of points inside. Thus, it clearly shows the efficiency of our proposed algorithms over the baseline.



(a) Prominence

(b) Periodicity

Fig. 10. Query Performance Breakdown on the T-drive Dataset

Performance breakdown. We further present the performance breakdown of algorithms for varying τ and ρ on the T-drive dataset in Figures 10a and 10b. Each group of bars reports the performance of one algorithm (i.e., CMC, RCI, and RCC+) when answering recurrent convoy queries. Each bar is split into three parts, namely IO, ALG, and SIM that represents I/O time to retrieve the clusters from the index, the execution time of the recurrent convoy search algorithm, and similarity computation between historic and online convoys in a sliding window, respectively. As soon as we find convoys that satisfy the given query threshold, we search for historic occurrences of each convoy within ρ time interval. This historic convoy search occupies most of the query execution time compared to the cluster and convoy generations of objects at the current timestamp. As shown in Figures 10a, our approach uses less time in all stages of the algorithm (i.e., retrieve clusters from the index, generate historic convoys, and compute similarity between convoys).

Index. The sizes of the indexes built on top of R-tree structure are reported in Table 4. The *baseline index* uses the timestamp of the cluster and the number of objects in a cluster as two dimensions of the MBR. Objects in a cluster are embedded as data into the node. Thus, the index size is directly proportional to the number of clusters to be indexed.

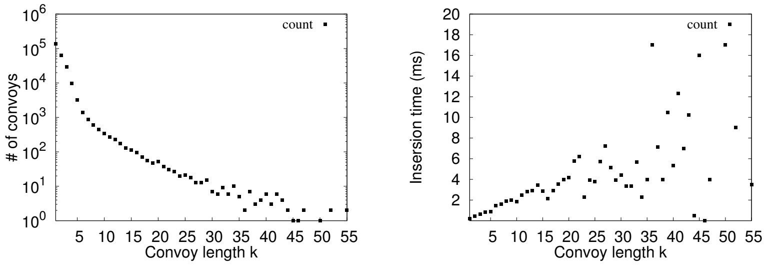
Dataset	*Baseline index*	*Intersection index*	*Convoy index*
T-drive	55	41	42
Beijing	339	342	339

Table 4. Index size of our method (unit: MB)

The *intersection index* contains clusters that have at least τ_{min} number of common objects with preceding clusters. Thus, the objects not observed in the previous clusters are not stored in the node. However, we embed extra information about the preceding clusters that share common objects with the current cluster. The size of the *intersection index* is smaller than the *baseline index* (as shown in Table 4) for T-drive dataset. This could be because there are some single clusters, which cannot form convoys. In contrast, the size of the *intersection index* is larger than that of the *baseline index* for the Beijing dataset, which indicates that the clusters in this dataset are highly inter-related w.r.t. the common objects.

The *convoy index* contains clusters with embedded information about the timespan of each object in the corresponding convoy. It can be seen from Table 4 that the sizes of *intersection index* and *convoy index* are almost similar for the T-drive dataset. This implies that most of the convoys have a length of two timestamps. In contrast, *intersection index* accounts for 342MB in the Beijing dataset; whereas *convoy index* accounts for 339MB. Longer convoys require more information about timespans of the objects in the convoys, which accelerates the intersection in the query processing.

Next, we study the *convoy index* update time for convoys of varying timespans of occurrence. Figure 11a shows the length distribution of indexed convoys' timespans in the T-drive dataset. A



(a) Convoy timespan distribution (b) Index update time distribution

Fig. 11. Convoy index update performance on the T-drive dataset

majority of convoys generated for indexing purpose last less than five timestamps as shown in Figure 11a. One of the reasons to find index convoys that last shorter timespans is the common occurrence of convergence and divergence between clusters at consecutive timestamps. Convoys occuring for longer timespans take more time to be inserted in the *convoy index* as the number of clusters grows. Nonetheless, the update time is still much faster as compared to the execution of the *recurrent convoy query*.

5.3 Effectiveness study

We choose the T-drive dataset to study the effectiveness of our method for mining recurrent convoys. DBSCAN [5] with parameter settings shown in Table 2 is applied to generate clusters at each timestamp. The generated clusters have their sizes varied from 4 to 18 objects. However, almost 70% of the clusters contain only 4 objects. We generate convoys based on different parameters settings listed in Table 3 over the T-drive dataset. *"Total convoys"* row in Table 5 shows the number of convoys mined for each pair of (τ, k). For example, we find 109,818 convoys that contain at least $\tau = 4$ common objects and last at least $k = 2$ timestamps. In contrast, the number of convoys for the thresholds $\tau = 7$ and $k = 4$ is the smallest.

Parameters	***Metric***	τ=4	τ=5	τ=6	τ=7	τ=4		
		k=4				***k=2***	***k=6***	***k=8***
Total convoys		18,216	5,703	2,095	778	109,818	5,181	2,941
Recurrent convoys	HD	2,265	869	365	138	4,205	926	589
Historic occurrences		14,185	3,992	1,253	343	103,773	3,031	1,390
Unique convoys		1,766	842	477	297	1,840	1,224	962
Recurrent convoys	τ-HD	1,772	612	272	130	3,694	818	543
Historic occurrences		15,202	4,579	1,583	501	104,481	3,520	1,783
Unique convoys		1,242	512	240	147	1,643	843	615

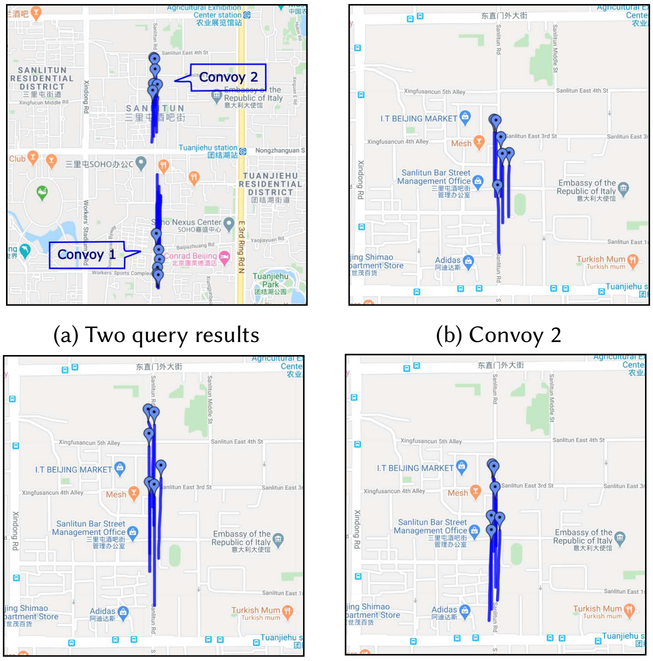
Table 5. Convoys in the T-drive dataset on varying parameter settings

Cluster similarity metric. As mentioned in Section 3, the similarity between convoys depends on the similarity between matching clusters. Thus, a proper choice of the metric to evaluate the similarity between clusters is crucial for mining recurrent convoys of interest. However, the choice of metric highly depends on the application scenario, which is out of the scope of this paper. In this paper, we adopt the commonly used similarity metrics that could be applied to point sets: *Hausdorff distance (HD)* [21]. HD measures the distance of two clusters by computing the max-min distance of containing points. Clusters are considered similar if the computed distance is within the threshold δ. Further, we use a tailored HD measure, namely τ-HD, that considers the query threshold τ to compute the cluster similarity. The original HD considers all points in two clusters while τ-HD computes the similarity between τ points from two clusters. The clusters are considered similar if τ points from each cluster are within δ Euclidean distance. Both δ_{HD} and $\delta_{\tau-HD}$ are set to 100 meters.

Recurrent convoys. Next, we search for recurrent convoys over the mined convoys from T-drive dataset without specifying the recurrence threshold ρ. Table 5 shows the number of recurrent convoys and the number of unique convoys w.r.t. two cluster similarity metrics and different convoy thresholds. Here, historic occurrences represent the set of convoys that are similar to recurrent convoys w.r.t. the thresholds τ and k. In contrast, a unique convoy does not have any previous occurrence w.r.t. the thresholds. As can be seen from the table, we find a small number of unique or recurrent convoys. The majority of the convoys in the dataset are historic occurrences of the recurrent convoys. For example, we find 5,181 total convoys w.r.t. the thresholds $\tau = 4$ and $k = 6$. However, 58% and 67% of the mined convoys are historic occurrences of other convoys for HD and τ-HD, respectively. As we increase the prominence and timespan thresholds, the total number of convoys also decreases by eliminating convoys with less objects or shorter timespans. We observe that we find more similar clusters by using the tailored τ-HD than HD. The reason is that HD considers all points to compute the similarity whereas the tailored τ-HD only considers τ similar points from each cluster.

Case study. The effectiveness of mined recurrent convoys can be demonstrated from the sample query result visualized using Google Maps API[1]. We run a *recurrent convoy query* with the following parameters: $\tau = 6, k = 25, \rho = 24hr$ and a sliding window of 30-minute length. Two recurrent convoys that satisfy the thresholds $\tau = 6$ and $k = 25$ are found at timestamp 04 Feb 2008 18:08:55,

[1] http://maps.googleapis.com



(a) Two query results

(b) Convoy 2

(c) 1^{st} historic occurrence of Convoy 2

(d) 2^{nd} historic occurrence of Convoy 2

Fig. 12. Recurrent convoy query result in a sliding window (04 Feb 2008 18:00:00-18:30:00)

as shown in Figure 12a. We then search for historic occurrences of those two convoys w.r.t. the recurrence threshold ρ.

Convoy #2, which occurs at the timestamp 04 Feb 2008 18:08:30 lasting 25 seconds as shown in Figure 12b, has two historic occurrences. The first one started at 03 Feb 2008 18:09:15 lasting 35 seconds (as shown in Figure 12c), and the second one started at 02 Feb 2008 19:48:55 lasting 50 seconds (as shown in Figure 12d). *Convoy #2* and its previous occurrences are observed in Sanlitun area that is located in Chaoyang District, Beijing as shown in Figure 12. This area is a popular destination for locals and foreigners containing many bars, restaurants and shopping malls [2]. Thus, *Convoy #2* is likely to be a convoy of taxis picking up or dropping off passengers along the street based on the location and time interval of occurrences. Since we use the Hausdorff distance to compute the similarity between convoys, it can be seen that the shapes of convoys are similar w.r.t. the given similarity threshold.

In case we do not find any result for certain values of thresholds, it is possible to further search for convoys by changing the threshold (incrementally). That is why we need efficient index structures that can facilitate the mining effort to query recurrent convoys, as we expect the users to frequently tune the parameters in order to explore convoys of interest.

6 CONCLUSION

In this paper, we studied the problem of finding a pattern of co-moving objects that repeats itself within given time interval. We formally defined the problem of finding *recurrent convoy query* in a sliding window and proposed algorithms and data structures that improve the efficiency in the mining process. Experimental study on real-life dataset shows the efficiency and effectiveness of our approach. Considering the correlations between convoys enables us to distinguish unique patterns from recurring patterns. In the future, we plan to extend this work by giving safe ranges of values (τ, k, ρ) where the current query result does not change. This facilitates the mining effort by guiding the user to the next value of interest to find different results.

Acknowledgements. This work was partially supported by ARC under Grants DP180102050, and DP200102611, a Google Faculty Research Award, the NSFC grant 91646204, the National Research Foundation, Prime Minister's Office, Singapore under its International Research Centres in Singapore Funding Initiative, and the National Science Foundation of the US under grant IIS-1816889. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of National Research Foundation, Singapore. Zhifeng Bao is the corresponding author.

REFERENCES

[1] Pankaj K Agarwal, Kyle Fox, Kamesh Munagala, Abhinandan Nath, Jiangwei Pan, and Erin Taylor. 2018. Subtrajectory clustering: Models and algorithms. In *PODS*. 75–87.

[2] Alka Bhushan, Umesh Bellur, Kuldeep Sharma, Srijay Deshpande, and Nandlal L Sarda. 2017. Mining swarm patterns in sliding windows over moving object data streams. In *SIGSPATIAL*. ACM, 60.

[3] Huiping Cao, Nikos Mamoulis, and David W Cheung. 2007. Discovery of periodic patterns in spatiotemporal sequences. *TIST* 19, 4 (2007), 453–467.

[4] Lu Chen, Yunjun Gao, Ziquan Fang, Xiaoye Miao, Christian S Jensen, and Chenjuan Guo. 2019. Real-time distributed co-movement pattern detection on streaming trajectories. *VLDB* 12, 10 (2019), 1208–1220.

[5] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*. 226–231.

[2] https://en.wikipedia.org/wiki/Sanlitun

[6] Qi Fan, Dongxiang Zhang, Huayu Wu, and Kian-Lee Tan. 2016. A general and parallel platform for mining co-movement patterns over large-scale trajectories. *VLDB* 10, 4 (2016), 313–324.

[7] Joachim Gudmundsson and Marc van Kreveld. 2006. Computing longest duration flocks in trajectory data. In *SIGSPATIAL*. ACM, 35–42.

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

[9] Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, Christian S Jensen, and Heng Tao Shen. 2008. Discovery of convoys in trajectory databases. *VLDB* 1, 1 (2008), 1068–1080.

[10] Tanvi Jindal, Prasanna Giridhar, Lu An Tang, Jun Li, and Jiawei Han. 2013. Spatiotemporal periodical pattern mining in traffic data. In *SIGKDD*. 11:1–11:8.

[11] Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. 2005. On discovering moving clusters in spatio-temporal data. In *SSTD*. Springer, 364–381.

[12] Ruoshan Lan, Yanwei Yu, Lei Cao, Peng Song, and Yingjie Wang. 2017. Discovering evolving moving object groups from massive-scale trajectory streams. In *MDM*. IEEE, 256–265.

[13] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. [n.d.]. Trajectory clustering: a partition-and-group framework. In *SIGMOD*.

[14] Xiaohui Li, Vaida Ceikute, Christian S Jensen, and Kian-Lee Tan. 2013. Effective online group discovery in trajectory databases. *TKDE* 12 (2013), 2752–2766.

[15] Yuxuan Li, James Bailey, and Lars Kulik. 2015. Efficient mining of platoon patterns in trajectory databases. *Data & Knowledge Engineering* 100 (2015), 167–187.

[16] Zhenhui Li, Bolin Ding, Jiawei Han, and Roland Kays. 2010. Swarm: mining relaxed temporal moving object clusters. *VLDB* 3, 1-2 (2010), 723–734.

[17] Zhenhui Li, Bolin Ding, Jiawei Han, Roland Kays, and Peter Nye. 2010. Mining periodic behaviors for moving objects. In *SIGKDD*. ACM, 1099–1108.

[18] Zhenhui Li, Jiawei Han, Bolin Ding, and Roland Kays. 2012. Mining periodic behaviors of object movements for animal and biological sustainability studies. *Data Mining & Knowledge Discovery* 24, 2 (2012), 355–386.

[19] Zhenhui Li, Jiawei Han, Ming Ji, Lu-An Tang, Yintao Yu, Bolin Ding, Jae-Gil Lee, and Roland Kays. 2011. Movemine: Mining moving object data for discovery of animal movement patterns. *TIST* 2, 4 (2011), 1–32.

[20] Nikos Mamoulis, Huiping Cao, George Kollios, Marios Hadjieleftheriou, Yufei Tao, and David W. Cheung. 2004. Mining, indexing, and querying historical spatiotemporal data. In *SIGKDD*. 236–245.

[21] Sarana Nutanong, Edwin H Jacox, and Hanan Samet. 2011. An incremental hausdorff distance calculation algorithm. *VLDB* 4, 8 (2011), 506–517.

[22] Faisal Orakzai, Toon Calders, and Torben Bach Pedersen. 2016. Distributed convoy pattern mining. In *MDM*, Vol. 1. IEEE, 122–131.

[23] Faisal Orakzai, Toon Calders, and Torben Bach Pedersen. 2019. k/2-hop: fast mining of convoy patterns with effective pruning. *VLDB* 12, 9 (2019), 948–960.

[24] Faisal Orakzai, Thomas Devogele, and Toon Calders. 2015. Towards distributed convoy pattern mining. In *SIGSPATIAL*. 1–4.

[25] Sutheera Puntheeranurak, Thi Thi Shein, and Makoto Imamura. 2018. Efficient discovery of traveling companion from evolving trajectory data stream. In *COMPSAC*, Vol. 1. IEEE, 448–453.

[26] Lu-An Tang, Yu Zheng, Jing Yuan, Jiawei Han, Alice Leung, Chih-Chieh Hung, and Wen-Chih Peng. 2012. On discovery of traveling companions from streaming trajectories. In *ICDE*. IEEE, 186–197.

[27] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Timos Sellis, and Xiaolin Qin. 2019. Fast large-scale trajectory clustering. *VLDB* 13, 1 (2019), 29–42.

[28] Yida Wang, Ee-Peng Lim, and San-Yih Hwang. 2006. Efficient mining of group patterns from user movement data. *Data & Knowledge Engineering* 57, 3 (2006), 240–282.

[29] Munkh-Erdene Yadamjav, Zhifeng Bao, Farhana M Chowdhury, Hanan Samet, and Baihua Zheng. 2019. Querying continuous recurrent convoys of interest. In *SIGSPATIAL*. ACM, 60.

[30] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-drive: driving directions based on taxi trajectories. In *SIGSPATIAL*. ACM, 99–108.

[31] Yipeng Zhang, Yuchen Li, Zhifeng Bao, Songsong Mo, and Ping Zhang. 2019. Optimizing impression counts for outdoor advertising. In *SIGKDD*. ACM, 1205–1215.

[32] Kai Zheng, Yu Zheng, Nicholas Jing Yuan, and Shuo Shang. 2013. On discovery of gathering patterns from trajectories. In *ICDE*. IEEE, 242–253.

[33] Kai Zheng, Yu Zheng, Nicholas J Yuan, Shuo Shang, and Xiaofang Zhou. 2014. Online discovery of gathering patterns over trajectories. *TKDE* 26, 8 (2014), 1974–1988.

[34] Yu Zheng. 2015. Trajectory data mining: an overview. *TIST* 6, 3 (2015), 29.

[35] Kaichun Zhou, Zongshun Tian, and Yuanwei Yang. 2019. Periodic Pattern Detection Algorithms for Personal Trajectory Data Based on Spatiotemporal Multi-Granularity. *IEEE Access* 7 (2019), 99683–99693.