Efficient Mining of Platoon Patterns in Trajectory Databases[☆]

Yuxuan Li, James Bailey, Lars Kulik

Department of Computing and Information Systems
The University of Melbourne, VIC 3010, Australia

Abstract

The widespread use of localization technologies produces increasing quantities of trajectory data. An important task in the analysis of trajectory data is the discovery of moving object clusters, i.e., moving objects that travel together for a period of time. Algorithms for the discovery of moving object clusters operate by applying constraints on the consecutiveness of timestamps. However, existing approaches either use a very strict timestamp constraint, which may result in the loss of interesting patterns, or a very relaxed timestamp constraint, which risks discovering noisy patterns. To address this challenge, we introduce a new type of moving object pattern called the *platoon pattern*.

We propose a novel algorithm to efficiently retrieve platoon patterns in large trajectory databases, using several pruning techniques. Our experiments on both real data and synthetic data evaluate the effectiveness and efficiency of our approach and demonstrate that our algorithm is able to achieve several orders of magnitude improvement in running time, compared to an existing method for retrieving moving object clusters.

Keywords: spatial clustering, trajectory database, moving object cluster, spatial pattern mining, data mining

1. Introduction

With the increasing availability of position-aware devices such as GPS receivers and mobile phones, it is now possible to collect and analyze large volumes of location databases that describe the trajectories of moving objects. Well known examples include taxi position data [1], animal movement data [2] and eye tracking data [3].

We address an important data mining challenge for trajectory data: discovering groups of spatial objects that move together for a certain period. We propose a new type of patterns, *platoon patterns*, that describe object clusters that stay together for time segments, each with some minimum consecutive duration of time. Figure 1 (a) shows an example of a platoon pattern. Wedding party vehicles o_2, o_3, o_4 and o_5 move together as a platoon at consecutive timestamps t_1, t_2, as well as consecutive timestamps t_4 and t_5.

The discovery of platoon patterns has a range of real-world applications. The identification of common routes among convoys may lead to more effective traffic control and the early discovery of truck platoons may assist traffic planning to avoid congestion. In eye tracking applications [3], the identification of common areas being viewed by a group of viewers can be used in advertising design and movie filming. In ecology, platoon patterns may provide a deeper understanding of animal migrations and in security may assist police to identify suspicious crowd movements.

1.1. Current Techniques

Several recent approaches for discovering moving object clusters have been reported in the literature, but they are not directly applicable for mining platoon patterns. We use "moving object cluster" as a generic term in our paper.

[☆]This research is supported under the Australian Research Council's Discovery Projects funding schema (project number DP110100757).

Email addresses: yuxuan.li@unimelb.edu.au (Yuxuan Li), baileyj@unimelb.edu.au (James Bailey), lkulik@unimelb.edu.au (Lars Kulik)

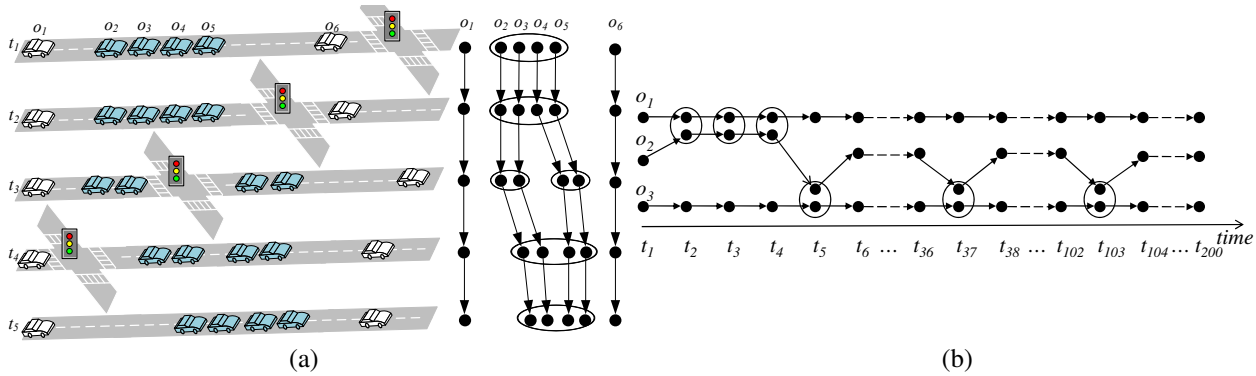


Figure 1: (a) A platoon pattern example. Vehicles o_2, o_3, o_4 and o_5 travel together as a platoon at timestamps t_1, t_2, t_4 and t_5. Existing patterns such as flock and convoy fail to capture the co-location behavior of this pattern due to their strict constraint on timestamp consecutiveness. (b) The pattern that moving objects o_2 and o_3 travel together at isolated and non-consecutive timestamps t_5, t_{37} and t_{103} is a swarm when $k = 3$.

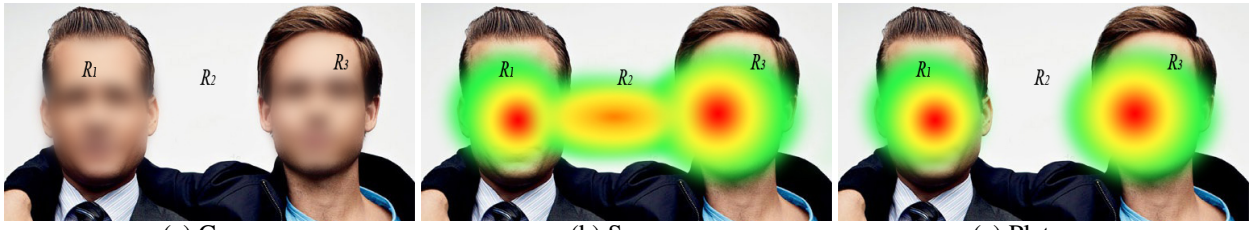
Previous work has proposed mining of moving objects that travel together for a minimum number of k consecutive timestamps such as flock [4, 5, 6] and convoy patterns [7, 8]. These patterns commonly require that all timestamps are strictly (or *globally*) consecutive. As pointed out in [9], enforcing timestamp consecutiveness may lead to the loss of interesting patterns. For instance, in Figure 1 (a) with $k = 3$, there are no convoy or flock patterns, since the four objects split into two clusters at t_3 due to a red traffic light, before coming together again at t_4. In our opinion, these four objects are an interesting moving object cluster.

Secondly, swarm patterns [9], take an opposite approach and remove any consecutiveness constraint on timestamps. Whilst this provides more latitude with regard to movement of clusters, it may also mine patterns that are overly "loose". Consider the example in Figure 1 (b) and assume we require at least $k = 3$ timestamps. Two vehicles (moving objects o_2 and o_3) might randomly encounter each other at some isolated and non-consecutive times (t_5, t_{37} and t_{103}), e.g. refilling fuel at the same petrol station, or stopping at the same car park. This does not imply the drivers have a strong association with each other. Although one might avoid outputting this type of pattern by imposing a larger threshold value for the minimum number of timestamps (e.g. $k = 4$ timestamps), this would risk missing patterns with two objects that do move together over shorter consecutive durations (such as t_2, t_3 and t_4). Another alternative would be to first mine all swarm patterns and then filter the interesting ones. Such an approach is time consuming, however, since the post processing constraints are not pushed inside the swarm mining task. Indeed, our experiments will show that the number of swarm patterns can be extremely large but contain only a small proportion of platoon patterns.

1.2. Platoon Patterns

Motivated by these issues, we propose a new definition for a moving object cluster called the *platoon pattern*, which allows the user to control the behavior of the consecutive time constraint to suit particular applications. Compared to the globally consecutive timestamp constraint of the convoy pattern [8], a platoon only requires that the timestamps are *locally* consecutive. Platoon patterns allow gap(s) in timestamps, but the consecutive time segments must have a minimum length (be locally consecutive). Given (1) a trajectory database with a timestamp-annotated history for moving objects, (2) a threshold for the minimum number of objects min_o that must appear in the platoon, (3) a threshold for the minimum number of timestamps min_t for which those objects travel together and (4) a threshold for the minimum number of consecutive timestamps min_c, a platoon pattern is an objectset and an associated timestamp sequence, denoted as $\{O : T\}$, such that $|O| \geq min_o$, $|T| \geq min_t$ and the timestamps in T are at least min_c locally consecutive. Intuitively, min_c denotes the minimum duration of a time segment in which objects stay together consecutively. In addition, platoon patterns do not rely on a particular clustering technique for deciding the spatial closeness of objects, which are instead modeled as preprocessing steps (c.f. Section 3 for our problem definition). The objects are required to be clustered.

Compared to the swarm query, with the combination of min_t and min_c, a platoon query is able to catch the patterns with consecutive timestamps without returning loose patterns. For example, if we set $min_o = 3$ and $min_t = 3$



(a) Convoy (b) Swarm (c) Platoon

Figure 2: Snapshot of a movie showing a dialog between two characters. Eye tracking data is represented as a heat map and eye movements of viewers focus on three dense regions: R_1, R_2 and R_3. (a) Convoy queries fail to identify interesting regions R_1 and R_3 due to the globally consecutive timestamp constraint. (b) Swarm queries erroneously consider R_2 to be interesting. (c) Platoon queries correctly identify R_1 and R_3 as interesting, using the locally consecutive timestamp constraint. Red color indicates high density of viewing, yellow indicates medium density and green indicates low density.

and $min_c = 2$, then Figure 1 (a) contains the platoon pattern $\{o_2, o_3, o_4, o_5 : t_1, t_2, t_4, t_5\}$. Objects are not considered forming a platoon pattern at timestamp t_3 since the spatial distance between o_3 and o_4 is greater than the maximum distance enforced by the used clustering algorithm. To avoid redundancy in the set of platoon patterns, we employ the notion of a *closed platoon pattern*. $(O : T)$ is a closed platoon if there is no platoon (O', T') for which either i) $O \subseteq O'$ and $T = T'$ or ii) $O = O'$ and $T \subseteq T'$. For example, $\{o_2, o_3, o_4 : t_1, t_2, t_4\}$ is not a closed platoon, since there is the platoon $\{o_2, o_3, o_4, o_5 : t_1, t_2, t_4, t_5\}$.

1.2.1. Motivational Example: Eye Tracking

Platoon patterns can capture the co-location behavior of moving objects for eye tracking datasets. We first explain the nature of an eye tracking dataset. Figure 2 (a) shows a snapshot of a movie containing a dialog between two characters. An eye tracking dataset records trajectories of the viewers eye movements during the movie. A heat map represents eye tracking data and omits time information. The density in the heat map indicates on which areas users focus their eyes and is shown in Figure 2 (b). Red (dark gray in B&W) areas are those where viewers looked most at the time, green (light gray in B&W) areas received little attention, and non-colored areas were not looked at.

For eye tracking data, the viewers' eye positions equate to objects, whilst the time dimension of the movie describes how the viewers' gaze varies (how the objects move). Figure 2 shows that there are three dense regions R_1, R_2 and R_3: where viewers frequently focus their attention. During a conversation between the two characters in a movie, the viewers switch their focus between these two characters. Since there is nothing interesting in the background, we would expect that R_1 and R_3 should be considered as the "interesting" regions. Region R_2 is unlikely to be of interest, as it is simply the result of eye movements between the two characters. The discovery of common eye movement patterns (moving object patterns) has applications in advertising, since they can guide product placement.

Compared to platoon patterns, convoy and swarm patterns are less suitable for eye tracking. Convoy patterns are determined by a globally consecutive timestamp constraint, and regions R_1 and R_3 would be missed, as it is unlikely that viewers look at the same region consecutively for the whole period (Figure 2 (a)). Swarm patterns have no time consecutiveness constraint, and region R_2 will be output (Figure 2 (b)), since it has been visited frequently (but not continuously). Platoon patterns use a local consecutive timestamp constraint, and only patterns in R_1 and R_3 are output (Figure 2 (c)), since they attract continuous focus.

1.2.2. Contributions

Efficient mining of platoon patterns in a large trajectory database is challenging. As the number of objects increases, the number of candidate patterns grows exponentially. We propose a platoon closed pattern mining algorithm called *PlatoonMiner* to address this issue. Four pruning techniques: Frequent-Consecutive pruning, Object pruning, Subset pruning and Common prefix pruning reduce the search space. The common prefix pruning rule is also able to directly extract closed platoons during the computation of platoon queries. Our experiments will demonstrate the effectiveness and the scalability of our proposed algorithm. In summary, we make the following contributions:

- We introduce a more flexible type of moving object cluster pattern, the *platoon* pattern.

- We propose a novel efficient algorithm *PlatoonMiner* for mining platoon patterns.
- We experimentally show the scalability of *PlatoonMiner* using real-world and synthetic datasets. Our algorithm can be several orders of magnitude faster compared to a swarm pattern mining algorithm.

2. Related work

We survey existing work on discovering moving cluster patterns and describe representative methods.

2.1. Approaches to Mine Moving Object Clusters

The *flock* pattern was proposed in [4]. A group of spatial objects moving together within a disk of a given radius r forms a flock. Later studies by Gudmundsson et al. [10, 5] introduced the minimum consecutive time period k as a parameter, instead of considering each time snapshot separately. Objects in the same group must stay together at all times during the period k (globally consecutive constraint): no object may leave or join the cluster. The disk shape constraint that is imposed for flock pattern may decrease its generality, e.g., a convoy of cars could travel in a line (instead of a disk). In comparison, the platoon pattern does not restrict the shape of moving object clusters.

A *moving cluster* [11] is a group of objects that are together for a certain time duration and fulfill two constraints: (1) there at least $MinPts$ objects in the group at all times and (2) objects together in the same set satisfy a spatial density value ε. A moving cluster also requires a minimum percentage of common objects between two consecutive timestamps θ, i.e., $\frac{|c_t \cap c_{t+1}|}{|c_t \cup c_{t+1}|} \geq \theta$, $0 < \theta \leq 1$, where c_t is a cluster at timestamp t. A moving cluster does not require objects from the same cluster to be present at all times in the cluster. Unlike platoon patterns, there is no constraint on the minimum number of timestamps (min_t).

Jeung et al. [7, 8] proposed the *convoy* pattern, which uses the number of common objects m, rather than the proportion θ between two consecutive timestamps, as a constraint for specifying the convoy pattern. In addition, convoy patterns enforce a minimum duration of consecutive timestamps, similar to the flock pattern.

A common property of flock, moving object and convoy patterns is that the timestamps are globally consecutive. In contrast, platoon patterns use a locally consecutive constraint: e.g. if we set $min_c = min_t \times 0.5$, only half of the timestamps of the minimum duration need to be consecutive (locally consecutive). On the other hand, a platoon query with $min_c = min_t$ is capable of retrieving the convoy patterns of fixed duration (all the timestamps in the fixed duration are consecutive, i.e., is globally consecutive).

The *swarm* pattern [9] is directly related to our work and it is connected with the MoveMine project [12]. In fact, swarm patterns may be considered as special case of platoon patterns when we set $min_c = 1$, i.e., the locally consecutive time constraint is removed. Swarm pattern mining considers the timestamps as an *unordered set*, rather than a *sequence*. There are some major differences between a platoon pattern and a swarm pattern query. The platoon pattern query can directly retrieve the moving object clusters with consecutive timestamps (Figure 2 showed why this can be important for eye tracking). It can also handle datasets with overlapping clusters (c.f. Section 4.6). Moreover, a platoon pattern query with $min_c = 1$ can retrieve all the swarm patterns, but not vice versa. We will use the swarm mining algorithm ObjectGrowth [9] as a baseline for evaluating the efficiency of *PlatoonMiner*.

Table 1 summarizes different moving object cluster patterns: only our approach can directly extract moving object clusters with locally consecutive timestamps. In addition, the flexibility of our approach enables users to mine different types of patterns proposed in the previous work. By setting $min_t = mint_c$ (global consecutive timestamps), PlatoonMiner can be used to discover convoy patterns. If we set $min_c = 1$ (no consecutiveness requirement), we also can use PlatoonMiner to mine swarm patterns. Therefore, PlatoonMiner can simulate the previous approaches by using different parameters but not vise versa.

2.2. Clustering of Spatial Trajectories and Moving Objects

In spatial trajectory clustering, Lee et al. [13] introduced a partition-and-group framework to find the common paths of a set of sub-trajectories. The trajectories are first partitioned into segments and then grouped into clusters according to distance. Compared to platoon mining, the focus is a geometric rather than a moving object perspective, and the temporal properties of trajectory data are not considered.

Table 1: Summary of moving object cluster patterns. Globally (locally): a pattern can be mined with a globally (locally) consecutive timestamp constraint.

Pattern	Globally	Locally	Minimum duration	Arbitrary shape
Flock	$\surd$	$\times$	$\surd$	$\times$
Moving cluster	$\surd$	$\times$	$\times$	$\surd$
Convoy	$\surd$	$\times$	$\surd$	$\surd$
Swarm	$\times$	$\times$	$\surd$	$\surd$
Platoon	$\surd$	$\surd$	$\surd$	$\surd$

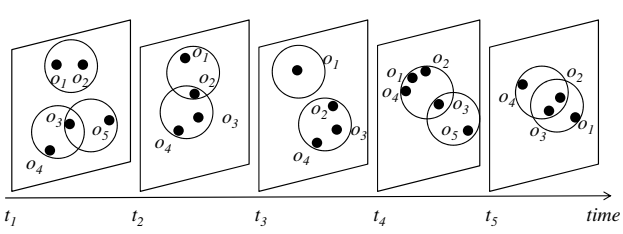
There are also studies on clustering moving objects [14, 15, 16]. In [14], micro-clustering is applied to group moving objects into clusters. Both current and near future positions of moving objects are considered during clustering. Kriegel et al. [15] also modified the DBSCAN algorithm [17] using fuzzy distance functions. Jensen et al. [16] proposed an approach for incrementally computing object clusters across a period of time. The major focus of these studies was about reducing the cost of computing and maintaining the object clusters, whilst the goal of platoon pattern mining is to discover the co-location patterns from the time changing object clusters for trajectory data.

2.3. Frequent Itemset Mining

Although we use a similar notation to the frequent itemset mining problem [18, 19, 20, 21], there are aspects that differentiate platoon pattern mining: (1) Platoon mining treats the sequential ordering of timestamps as significant, e.g., the number of consecutive timestamps in the temporal object cluster $C_1 = \{o_1, o_2 : t_1, t_2, t_3\}$ is three, while there are only two consecutive timestamps in the temporal object cluster $C_2 = \{o_1, o_2 : t_1, t_6, t_7\}$. If we set $min_c = 3$, the cluster C_2 is not a platoon. In contrast, support (frequency) is the only measurement for an itemset in frequent itemset mining. Treating a moving object as an item C_1 and C_2 are identical in the previous example, i.e., $\{o_1, o_2 : 3\}$. However, the objects in clusters could change over time. (2) In the spatial context, cluster overlapping is allowed, which means that measurement of support is not directly applicable. (3) There is a threshold for the minimum number of objects in the platoon mining problem, while the size of itemset is not a concern in the frequent itemset mining problem. For example, if we set $min_o = 2$, $\{o_1 : T\}$ cannot be a platoon regardless of $|T|$. (4) The popular pruning techniques used in frequent itemset mining are item merging, sub-itemset pruning and item skipping. However, the pruning rules used in platoon pattern mining are *Frequent-Consecutive pruning*, *Object pruning rule*, *Subset pruning rule* and *Common prefix pruning rule*. The former two pruning rules do not apply to frequent itemset mining. The Subset pruning rules can be seen as a generalized version of sub-itemset pruning, which can handle overlapping clusters. Compared to item skipping, the Common prefix pruning rule is implemented by subtree substitution and it also has a minimum object constraint. More details will be given in Section 4.3.

2.4. Frequent Sequential Pattern Mining

Another closely related research topic is the problem of frequent sequential pattern mining which was first introduced in [22], followed by later extensive studies in [23, 24, 25, 26, 27, 28]. Given a sequence database and a frequency threshold, the task is find all frequent subsequence patterns from the database. There were three algorithms [22] proposed to address this task. The algorithms *AprioriSome* and *DynamicSome* focus on solely mining the maximal frequent subsequence patterns whereas *AprioriAll* does not use the maximality constraint. A maximal subsequence pattern A is a pattern such that there is no other pattern B with $A \subset B$. AprioriAll works as follows. It first scans the database D once to compute all frequent single items (1-sequence). Then it combines every pair of frequent candidate a and b to generate 2-sequences ab and ba. Another scan on D is performed to obtain the frequent patterns of length 2. Next, the algorithm merges two frequent $(k-1)$-patterns A and B that share the first $k-2$ items, to generate a k-candidate. The first $k-1$ items of the candidate are the same as A and the k-th item is the same as the last item of B. AprioriAll generates all k-candidates in this way from frequent $k-1$-patterns and tests them against D to get all frequent k-patterns. The candidate generation step and database scan and check step are executed alternately until no new candidate can be generated.



t	object clusters
t_1	$\{o_1,o_2\},\{o_3,o_4\},\{o_3,o_5\}$
t_2	$\{o_1,o_2\},\{o_2,o_3,o_4\}$
t_3	$\{o_1\},\{o_2,o_3,o_4\}$
t_4	$\{o_1,o_2,o_3,o_4\},\{o_3,o_5\}$
t_5	$\{o_1,o_2,o_3\},\{o_2,o_3,o_4\}$

closed platoons
(o_3,o_4: t_1,t_2,t_3,t_4,t_5)
(o_2,o_3,o_4: t_2,t_3,t_4,t_5)
(o_1,o_2,o_3: t_4,t_5)
(o_1,o_2: t_1,t_2,t_4,t_5)

Figure 3: An example scenario.

These methods have been proven to work efficiently in classical sequence databases. Similar to the frequent itemset mining, the interestingness of a sequential pattern is only assessed by its occurrence frequency. The order of these occurrences is seen as irrelevant.

Moving object clusters can be considered as an extension of sequential patterns to trajectory databases. Compared with classical sequential patterns, moving object clusters take the spatial property of an object (item) into account. Compared to classical sequential pattern mining, only objects that are spatially close to each other are considered as interesting. This requires the computation of spatial closeness between objects. In particular, our approach allows users to control the duration of consecutive timestamps at which objects stay together. This can be described as the length of subsequences of a sequential pattern. Novel techniques are required to check the consecutiveness constraint in the context of timestamp-based data.

3. Problem definition

Let $T_S = \{t_1, t_2, ..., t_n\}$ be a linearly ordered set of timestamps of a trajectory history (called ***time space***). Let $O_S = \{o_1, o_2, ..., o_m\}$ be a collection of objects that appear in T_S (called ***object space***). An object $o_i \in O_S$ is observed at (possibly nonconsecutive) timestamps $T \subseteq T_S$. We refer to T as a timestamp sequence and its length is $|T|$. A ***trajectory database*** stores the trajectories of individual objects at distinct time points. A set of moving objects O (called *objectset*) that travel together as a cluster for a timestamp sequence T is denoted as $C = (O : T)$ and called a *temporal object cluster*, where $O \subseteq O_S$ and $T \subseteq T_S$. For each timestamp, an object o_i can belong to more than one cluster, i.e., overlapping clusters are allowed.

Given a minimum number of object threshold min_o, a temporal object cluster $C = (O : T)$ is *significant* if $|O| \geq min_o$. Two timestamps $t_i, t_j \in T$ are *consecutive* if $|j - i| = 1$. For $T' \subseteq T$ let t_{max} be the largest timestamp and t_{min} be the smallest timestamp in T'. T' is a consecutive timestamp sequence if $\forall t \in T, t_{min} \leq t \leq t_{max}, \exists t' \in T'$ such that t and t' are consecutive. We say $T' \subseteq T$ is maximally consecutive if $\nexists T'' \subseteq T$, such that $T' \subset T''$ and T'' is a consecutive timestamp sequence. Let $S_{l-con}(T) = \{T' \mid T' \subseteq T \wedge T' \text{ is maximally consecutive } \wedge |T'| \geq l\}$, i.e. $S_{l-con}(T)$ is the set of all maximally consecutive timestamp sequences T' of T with length at least l.

Given a minimum number of timestamps threshold min_t and minimum number of *consecutive* timestamps threshold min_c, $C = (O : T)$ is *frequent* if $|T| \geq min_t$; C is min_c *locally consecutive*, if $\forall t \in T$, there exists a T' such that $T' \in S_{min_c-con}(T)$ and $t \in T'$. i.e. T decomposes into consecutive segments, each of length at least min_c. Also, $C = (O : T)$ is min_c *globally consecutive*, if $T \in S_{min_c-con}(T)$.

For example, given a temporal object cluster $C = (O : T)$ and thresholds $min_t = 5$ and $min_c = 2$, where $T = \{t_1, t_2, t_4, t_5, t_6\}$. There are two maximally consecutive timestamp sequences in T: $T'_1 = \{t_1, t_2\}$ and $T'_2 = \{t_4, t_5, t_6\}$. $S_{2-con}(T) = T'_1 \cup T'_2 = \{\{t_1, t_2\}, \{t_4, t_5, t_6\}\}$. Now $|T| = 5 \geq min_t$ and C is frequent. Since $|T'_1|, |T'_2| \geq min_c$, C is locally consecutive. We formally define a platoon pattern. Therefore, if we set $min_c < min_t$, only partial timestamps of the minimum duration need to be consecutive (locally consecutive). On the other hand, a platoon query with $min_c = min_t$ can mine the platoon pattern that all the timestamps of the minimum duration are consecutive (globally consecutive).

Definition 1. *A platoon is a temporal object cluster $C = (O : T)$ that is significant, frequent and locally consecutive.*

Intuitively, a platoon is a cluster of a number of objects that travel together for some consecutive segments of time. In Figure 1, for $min_o = min_t = min_c = 2$ the platoon $C_1 = (o_2, o_3, o_4, o_5 : t_1, t_2, t_4, t_5)$ is returned. The derived platoons

$C_2 = (o_2, o_3 : t_1, t_3, t_4)$ and $C_3 = (o_2, o_3, o_4, o_5 : t_1, t_2)$ contain less information than C_1 since $O_2 \subseteq O_1$ and $T_3 \subseteq T_1$. A platoon $C = (O : T)$ is considered as *object-maximal* if there is no other platoon $C' = (O' : T')$ such that $O \subset O'$ and $T = T'$; C is considered as *time-maximal* if there is no other platoon $C' = (O' : T')$ such that $T \subset T'$ and $O = O'$. The maximal objectset and maximal timestamp sequence of C are denoted as $O_{max}(C)$ and $T_{max}(C)$, respectively.

Definition 2. *A platoon $C = (O : T)$ is* closed *if and only if C is both object-maximal and time-maximal.*

Pre-processing of Input: Given a trajectory database, our problem is to mine the complete set of closed platoon patterns. As a preprocessing step, any spatial clustering algorithm (e.g. DBSCAN [17]) and distance metric (e.g. Euclidean distance) can be used to obtain the clusters at each snapshot of the trajectory database. The output is a *temporal object cluster database*, denoted as C_{DB}. Example scenario 1 is used throughout the paper.

Example 1. *Figure 3 shows the example scenario will be using throughout the paper, where $T_S = \{t_1, t_2, t_3, t_4, t_5\}$ and $O_S = \{o_1, o_2, o_3, o_4, o_5\}$. At each timestamp, objects are assigned to different clusters with some maximum diameter and cluster overlapping is allowed. e.g. o_3 belongs to two different clusters at t_1. Our task is to retrieve the complete set of closed platoons where $min_o = min_t = min_c = 2$.*

Definition 3. *(Problem definition) Given a pre-processed trajectory database D and thresholds min_o, min_t and min_c, our task is to mine the complete set of closed platoon patterns from D.*

4. Retrieval of Closed Platoons

The definition of closed platoons suggests a simple way to retrieve all closed platoon patterns. First build an enumeration tree of either the object or the time space, and then traverse this tree. The tree contains every combination of objects (or timestamps) in depth-first search order (DFS) or breadth-first search (BFS) order. The enumeration tree has $2^{|O_S|}$ (or $2^{|T_S|}$) nodes and this exhaustive search has time complexity of $O(2^{|O_S|} \cdot |T_S| \cdot |O_S|)$, since at each node we need to scan T_S (O_S) to calculate T_{max} (O_{max}). Additional time is also needed to filter out non-closed patterns from the pattern output set quadratic in the number of candidate patterns).

As the naive (brute force) approach is impractical for large datasets, we propose four pruning rules to narrow the search space. We expect the number of timestamps to be larger than the number of moving objects and the clustering process to be used for grouping moving objects instead of timestamps. We thus construct the enumeration tree based on the object space and traverse it depth first. The first pruning rule is *Frequent-Consecutive pruning* and removes patterns that are not frequent and/or locally consecutive. The *Object pruning rule* prunes patterns that are not significant. The *Subset pruning rule* avoids unnecessary extensions of the current objectset. The *Common prefix pruning rule* directly extracts the closed platoon based on a subtree substitution technique, avoiding the need for post processing of patterns.

4.1. Main Ideas

Figure 4 provides an overview of our approach. It contains two modules.

- In *preprocessing*, objects at each timestamp of the trajectory database are clustered into groups, yielding clusters for each timepoint. As mentioned before, platoon patterns do not rely on a particular clustering technique for deciding the spatial closeness of objects. This new representation is denoted as C_{DB}.
- *PlatoonMiner* retrieves the complete set of closed platoons from C_{DB} using depth first search in the object space. We use prefix tables (c.f. Section 4.2) to efficiently store candidates at each step. The upper-left part of Figure 5 shows an example of the search tree. The search proceeds from left to right and top to bottom. In each iteration, each node is associated with a candidate C that has an objectset O and a timestamp sequence T_{max} at which the objectset occurs. Four punning rules are used to speed up the mining process (c.f. Section 4.3). Unqualified Candidates will be removed and the search will not continue down to their subtrees. Any closed platoon found in current iteration will be directly extracted.

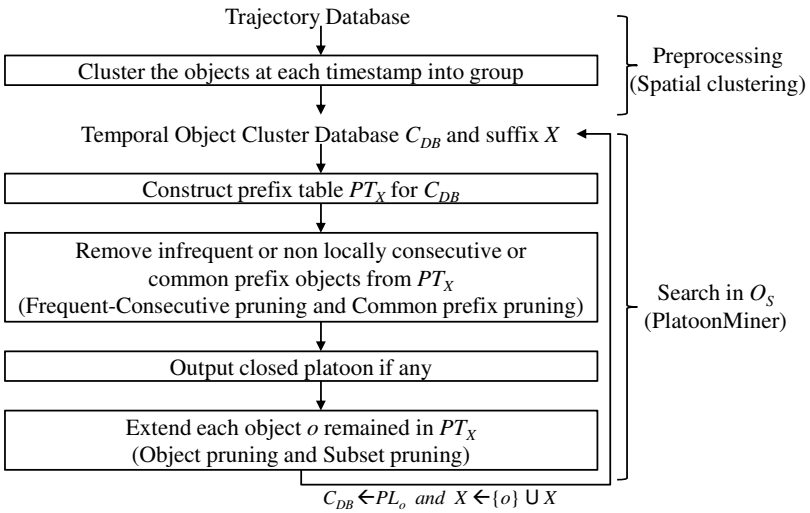


Figure 4: Structure of the PlatoonMiner algorithm.

The use of notations is summarized in Table 2.

In the PlatoonMiner algorithm, we consider an objectset as an object string, ordered according to lexicographical order. Consider an objectset O and an object o_i in O. Then O' is called the prefix of o_i, if $\forall o_j \in O, j < i$ we have $o_j \in O'$. For example, for the objectset $O = \{o_1, o_2, o_3\}$, the set $\{o_1, o_2\}$ is the prefix of o_3 (and likewise o_3 is the suffix of $\{o_1, o_2\}$). Each objectset in the node of the search tree is the suffix X of its children. Therefore, each objectset O consists of two parts: the current object o and its suffix X, where $O = \{o\} \cup X$ and X is the objectset of its parent node. For example, in Figure 5, the current object o and the suffix X of objectset $\{o_3, o_4, o_5\}$ are o_3 and $\{o_4, o_5\}$ respectively. In order to use our pruning algorithms, we also store the number of occurrences of a cluster C, which is denoted as N ($N \geq |T|$). In Figure 3, objectset $\{o_2, o_3\}$ at timestamp t_5 is in two different clusters and hence counted twice, but there is only one actual cluster $\{o_2, o_3\}$. A temporal object cluster C can be written as $(O : T : N)$, in situations where it is important to specify the number of occurrences.

During enumeration, one challenging task is to calculate T_{max} of the current candidate C with objectset O. A naive approach is to perform a *full* scan on C_{DB} every time to obtain the timestamps that the objectset O appears. Therefore, the number of points that need to be scanned for computing the T_{max} is $(|T_S| \cdot |O_S|)$ (if all moving objects exist in the whole timestamp history). As shown below, in our approach, only one full scan on C_{DB} is needed.

4.2. Prefix Table

In PlatoonMiner, T_{max} of the current objectset $O = \{o\} \cup X$ is obtained by the *prefix table* of their parent X, where $\forall o \in (O_S - X)$. The prefix table is a data structure with a two-level hash index which allows fast computing for T_{max}. Each prefix table is associated with a suffix X, denoted as PT_X. The prefix table stores T_{max} (later updated as $S_{min_c-con}(T_{max})$ via Algorithm 3), N_{con} (number of occurrence of locally consecutive timestamps) and PrefixList PL_o of objectset $O = \{o\} \cup X$ ($\forall o \in (O_S - X)$), where o is the first level hash index. In addition, PL_o records the set of prefix $\{P\}$ of objectset O as well as the variables T_p and N_p. P is the second level hash index and T_p and N_p are the timestamp sequence and the number of occurrences of objectset $P \cup O$ in T_S respectively. For example, in Figure 5, PT_{o_4} records $S_{min_c-con}(T_{max})$ and N_{con} of objectsets $\{o_1, o_4\}$, $\{o_2, o_4\}$ and $\{o_3, o_4\}$, as well as their prefixes. $\{o_1\}$ is the prefix of $\{o_2, o_4\}$ in PT_{o_4}.

When the search commences, we first scan the input C_{DB} to count T_{max} for objectset $O = \{o\} \cup \phi$, where $X = \phi$ and $\forall o \in O_S$. Meanwhile, we collect the prefix of o of each object cluster. These results are stored in PT_ϕ. Next, we extend the object in PT_ϕ and construct PT_o where $\forall o \in PT_\phi$. The PrefixList of o in PT_ϕ then becomes the input C'_{DB} of PT_o. After that, we construct the prefix table for the object in PT_o in the same way. This process repeats recursively until there is no object to be extended in the last prefix table. Let P be the prefix of O, since $|P| < |O|$, we have $|C'_{DB}| < |C_{DB}|$. Thus, only one full scan on C_{DB} is needed for constructing the PT_ϕ.

Table 2: Summary of the use of notations.

O_S	Object space.
T_S	Timestamp history.
O, O'	Objectset that is contained in O_S.
T, T'	Timestamp sequence that is contained in T_S.
T_{max} (T_{max}(C))	The maximum timestamp sequence of C.
N	The number of occurrences of timestamps.
N_{con}	The number of occurrences of locally consecutive timestamps.
$S_{l-con}(T)$	The set of all maximally consecutive timestamp sequences T' of T with length at least l.
$S_{min_c-con}(T)$	The set of all maximally consecutive timestamp sequences T' of T with length at least min_c.
PT_O	The prefix table of O.
PL_o	PrefixList of $O = \{o\} \cup X$ ($\forall o \in (O_S - X)$).

4.3. Pruning Rules

As mentioned before, the number of nodes in the search tree based on the object space is in the worst case $|2^{|O_S|}|$, thus requiring pruning strategies to narrow down the search space. There are three well known pruning strategies in mining closed frequent itemsets [19, 20, 21]: item merging, sub-itemset pruning and item skipping. These pruning strategies have been proven to be effective in avoiding searching redundant candidates patterns but they do not directly support the closed platoon query as shown in the previous section. Instead, the four pruning rules used in the PlatoonMiner algorithm are as follows.

4.3.1. Frequent-Consecutive Pruning Rule

For suffix X, after construction of prefix table PT_X, we may derive T_{max} of each child of X from PT_X. Lemmas 1 and 2 show that if a child does not satisfy the min_t or min_c threshold, then its descendants cannot be platoons.

Lemma 1. *If a temporal object cluster $C = (O : T)$ is not frequent and $O \subseteq O'$, then $C' = (O' : T')$ is not frequent.*

The proofs of lemmas can be found in Appendix A.

Lemma 2. *If a temporal object cluster $C = (O : T)$ is not locally consecutive, then any $C' = (O' : T')$ such that $O \subseteq O'$ cannot be locally consecutive.*

Using these lemmas, we have the following pruning rule.

Rule 1. *If a current candidate $C = (O : T)$ is not frequent or not locally consecutive, then we can prune the subtree from O because there is no (closed) platoon for any descendant.*

Example 2. *In Figure 5, for the candidate pattern associated with objectset $\{o_5\}$, we have $|S_{2-con}(\{t_1, t_4\}) = \phi| < 2$ then the whole subtree of $\{o_5\}$ can be pruned.*

Frequent checking on $|T|$ requires constant time. To calculate $S_{min_c-con}(T)$, requires linear time to scan T from left to right. During calculation, we extract those T' such that, T' is consecutive $\wedge$ $|T'| \geq min_c$ from T (see Algorithm 3). If the threshold is not met, the whole subtree of current node can be pruned. In practice, we test the two thresholds by performing frequent checking on $S_{min_c-con}(T_{max})$ since it ensures all its timestamp segments are consecutive. Thus any record in PT_X such that $|S_{min_c-con}(T_{max})| < min_t$ can be removed to avoid redundant search.

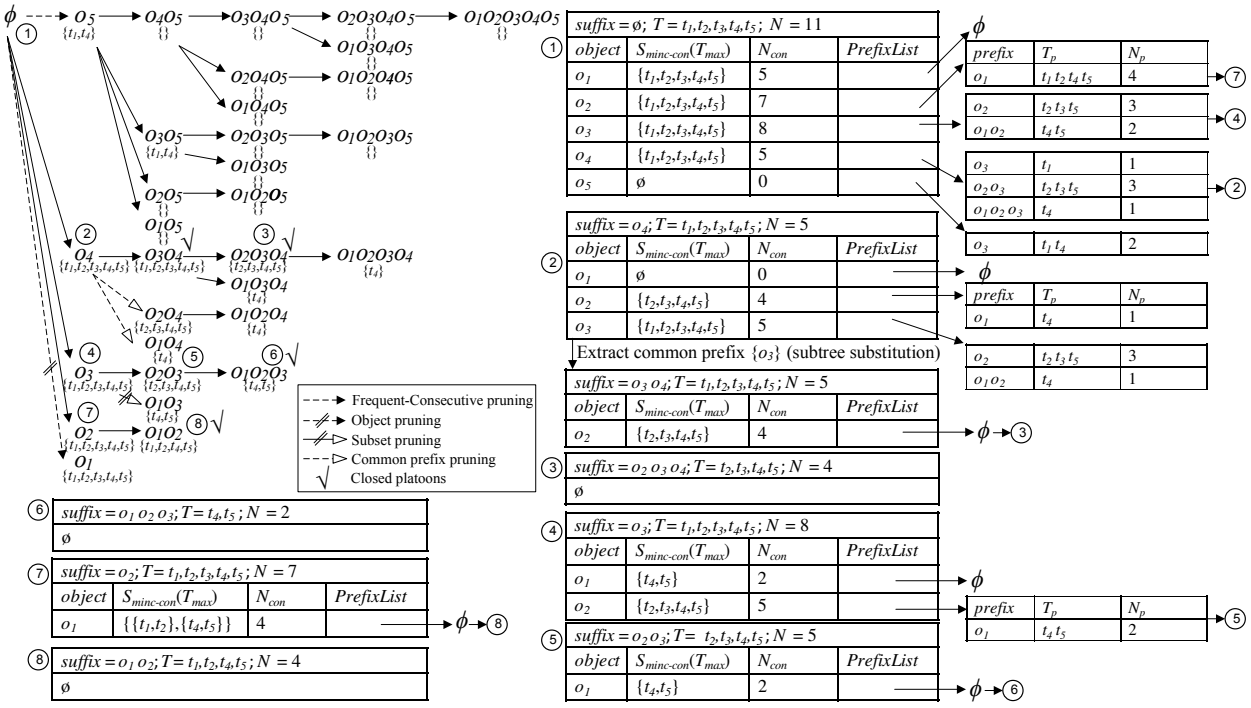


Figure 5: PlatoonMiner algorithm example scenario. 8 nodes are visited and 8 prefix tables are constructed. Each node is the suffix of its children, and each suffix X is associated with a prefix table. The closed platoon $(o_3, o_4 : t_1, t_2, t_3, t_4, t_5)$ is output in step 2 by Common prefix pruning rule (the subtree of $\{o_3, o_4\}$ substitutes the subtree of $\{o_3\}$).

4.3.2. Object Pruning Rule

Another constraint for a (closed) platoon is the minimum objects threshold min_o. Opposite to T_{max}, the objectset of a child node is always larger than the objectset of its parent. Thus, the Frequent-Consecutive Pruning rule is not applicable to the minimum object threshold. However, for any node $\{o\} \cup X$ of the search tree, the largest objectset of its descendant is determined by the index of first o and $|X|$. Lemma 3 proves the correctness of this rule. Similar to the frequent checking rule on T, the object pruning rule also only takes constant time.

Lemma 3. *In a depth-first search order tree, let $C = (O : T)$ be the current candidate and o_i be the first object of O, where $1 \leq i \leq |O_S|$. If C is not significant and $(i-1) + |O| < min_o$, then any descendant of C is not significant.*

Rule 2. *Let o_i with a suffix X to be an object under consideration to be extended, if $i + |X| < min_o$, then the whole subtree of $o_i \cup X$ can be pruned.*

Example 3. *In Figure 5, the suffix of objectset $\{o_2, o_5\}$ is $\{o_5\}$. The maximum number of objects for the descendent of $\{o_2, o_5\}$ is $2 + |\{o_5\}| = 3$, if we set $min_o > 3$, the subtree of $\{o_2, o_5\}$ can be pruned.*

4.3.3. Subset Pruning Rule

In the extreme case for a platoon query, we have $min_o = min_t = min_c = 1$ and the previous two pruning rules will have no effect on the enumeration tree. However, we can introduce another rule, the subset pruning rule, to shrink the search space.

Lemma 4. *In a depth-first search tree, if $O \subset O'$ and $T = T'$ and $N = N'$, where $C' = (O' : T' : N')$ is a platoon found previously. Then neither $C = (O : T : N)$ nor its descendants cannot be a closed platoon.*

Rule 3. *Let C' be a found previously platoon and $C = (O : T : N)$ be a candidate. If O is a proper subset of O', $T = T'$ and $N = N'$, then the subtree of O can be pruned.*

Example 4. *In Figure 5, $C' = (o_1, o_2, o_3 : t_4, t_5 : 2)$ is returned as a closed platoon in step 6. $C = (o_1, o_3 : t_4, t_5 : 2)$ is not a closed platoon. Assume that there is another object o_0 in Figure 5, and C's descendant $C'' = (o_0, o_1, o_3 : t_4, t_5 : 2)$, which means o_0 is always with $\{o_1, o_3\}$. Since $T = T'$ and $N = N'$, o_0 must be also always with $\{o_2\}$, thus $(o_0, o_1, o_2, o_3 : T : N)$ is also a platoon which have been found before (DFS order).*

When performing subset checking on the current candidate $C = (O : T : N)$, let R be the patterns found so far. There are three possible outcomes: (1) $\exists C' \in R$ such that $O \subset O'$ and $T = T'$ and $N = N'$; (2) $\exists C' \in R$ such that $O \subset O'$ and $T = T'$ and $N > N'$; (3) otherwise. In (1), the Subset pruning rule takes effect and the whole subtree of current node can be pruned. Case (2) may happen when there are overlapping clusters in the dataset. e.g. In Figure 5, when we perform the subset checking on $C = (o_2, o_3 : t_2, t_3, t_4, t_5 : 5)$, we have $C' = (o_2, o_3, o_4 : t_2, t_3, t_4, t_5 : 4)$, where $C' \in R$, $O \subset O'$, $T = T'$ and $N > N'$. In such a case, C is not a closed platoon according to the definition, but closed platoons may exist in the descendants. In this example, $N > N'$ suggests that there must exist another objectset O'' that contains O but not $\{o_4\} = O' - O$ in some timestamp of T. In fact, $O = \{o_2, o_3\}$ and $\{o_1\}$ are also in the same cluster at t_4 and t_5, where $\{o_1, o_2, o_3\}$ is the child of $\{o_2, o_3\}$ and $\{o_1, o_2, o_3 : t_4, t_5\}$ forms a closed platoon. In case (3), C is a closed platoon if $|O| \geq min_o$ and there is no common prefix of O (see the next section). To speed up subset checking, we can build a hash index on T for the patterns in R.

4.3.4. Common Prefix Pruning Rule

Let $R = \{C'\}$ be a set of patterns found so far, the subset checking ensures that there is no pattern C such that $O \subset O'$ and $T = T'$ will be added into R. However, we also need to ensure that $\nexists C' \in R$ such that $O \supset O'$ and $T = T'$. A naive approach is to perform the closure checking to remove the non-closed platoon after the search, which yields a time complexity of $O(R^2)$. Lemma 5 shows that we can directly extract closed platoons during the query computation.

Lemma 5. *Given a temporal object cluster $C = (O : T : N)$, if there is an objectset O' (and $\nexists O'' \supset O'$) occurs in the every prefix of O, then (1) $(O' \cup O : T : N)$ forms a closed platoon (if $|O' \cup O| > min_o$) and (2) there is no closed platoon in the subtree of O that does not contain O'.*

Rule 4. *In the prefix table PT_X, any object o that has the same number of occurrences as X should be added into CP (a set of common prefix) to forms a closed platoon $(CP \cup X : T : N)$ if $CP \cup X > min_o$. Additionally, any subtree of objectset X that does not contain CP can be pruned.*

Example 5. *In Figure 5, we have $N_{con}(o_3) = N = 5$ in the prefix table of $\{o_4\}$, thus $(o_3, o_4 : T : N)$ forms a closed platoon. Subtrees $\{o_1, o_4\}$ and $\{o_2, o_4\}$ can be pruned.*

The common prefix pruning rule can be seen as a modified version of the item-merging [19] by adding the minimum object constraint. However, in PlatoonMiner, this rule is implemented by subtree substitution. In Example 5, we extract common prefix $\{o_3\}$ to form a new suffix $\{o_3, o_4\}$. Then, we use the subtrees of $\{o_3, o_4\}$ to substitute the subtree of $\{o_4\}$. That is, we search $\{o_2, o_3, o_4\}$ and $\{o_1, o_3, o_4\}$ (and their descendants) instead of $\{o_2, o_4\}$ and $\{o_1, o_4\}$ (and their descendants). Comparing the number of occurrences requires $O(1)$ extra time.

4.4. PlatoonMiner Algorithm

The pseudocode of the PlatoonMiner algorithm based on the above lemmas is presented in Algorithms 1 - 5 (refer to Appendix C). Figure 5 illustrates the execution steps of the PlatoonMiner algorithm in our example scenario. The algorithm takes a temporal object cluster database as input and the number of occurrences of each cluster is initialized as 1. The entry of the algorithm is to call $PlatoonMiner(C_{DB}, \phi, T_S, |C_{DB}|, min_o, min_t, min_c, 0)$, where $min_o = min_t = min_c = 2$ for the example scenario.

In Algorithm 1, the first task is to build the prefix table PT for given C_{DB} with suffix X (line 1 - 5), which is done by Algorithm 2. First, the objectset of each temporal object cluster C in C_{DB} is scanned from left to right and inserted into the prefix table PT_X. For each object o in O, T_{max} is merged with T (line 3, Alg.2). Here we use symbol "$\oplus$" instead of "$\cup$" as we allow duplicate timestamps in T_{max} in order to use Algorithm 3 to extract $S_{min_c-con}(T_{max})$. We then update the PrefixList of o if PT contains a record for o (line 4 - 8, Alg.2). If o is new to PT, we insert o into PT and put the prefix of o into PL_o (line 9 - 11, Alg.2). After that, we call Algorithm 2

recursively if o is not the last object (rightmost) of O. Once PT is built, the common prefix of suffix X as well as the objects that cannot satisfy time constraints are detected (line 8 - 15, Alg.1). We first use Algorithm 3 to obtain the locally consecutive timestamps $S_{min_c-con}(T_{max})$ as well as its number of occurrence N_{con}. Consider an example where $T_{max} = \{t_1, t_2, t_2, t_4, t_7, t_8\}$ and $min_c = 2$. The computation is performed by scanning T_{max} from left to right (line 5 - 15, Alg.3). T_{con} keeps growing as long as the adjacent timestamps are consecutive. Meantime, c records the number of occurrence of consecutive timestamps including those are duplicate. When it encounters a gap, T_{con} is added into S_{min_c-con} and N_{con} increases by c if $|T_{con}| \geq min_c$ (line 11, Alg.3). T_{con} and c are then reset (line 12, Alg.3). For this example, we have $S_{min_c-con}(T_{max}) = \{\{t_1, t_2\}, \{t_7, t_8\}\}$. S_{min_c-con} and N_{con} are returned to Algorithm 1 for pruning tests. According to lemma 5, $(CP \cup X : T : N)$ forms a closed platoon if CP is not empty. The next step extends the prefix of the objects in PT by calling Algorithm 4 (line 23, Alg.1). We process the objects in PT in reversed order. If the current object cannot be extended to be a significant objectset, this object and the rest in PT will stop extending according to lemma 3 (line 2 - 4, Alg.4). Then, the suffix X is extended to $X' = \{o\} \cup X$ (line 5, Alg.4), where X' is the objectset of the child of X. After performing subset checking on candidate C, the whole subtree of X' can be pruned if $s = 0$ according to lemma 4. Otherwise, we call Algorithm 1 recursively using C'_{DB} and X' as input.

Appendix B shows how PlatoonMiner works on the example scenario (Figure 5).

Theorem 1. *The set of closed platoon patterns returned by the PlatoonMiner algorithm is correct and complete.*

4.5. Time and Space Complexities Analysis

For each node of search tree, the memory usage of PlatoonMiner to build the prefix table is $O(|O_S| \cdot |T_S|)$. The upper bound only holds for building PT_ϕ. Afterwards, the C_{DB} shrinks to the PrefixList of each object in the previous prefix table. PlatoonMiner requires $O(|O_S| \cdot |T_S|)$ to compute the T_{max} for current candidate.

4.6. Handling of Overlapping Clusters

Overlapping clusters are common in many real-world applications. Later, we compare our work to the fastest known algorithm for mining swarm patterns: ObjectGrowth [9]. The work of [9] implicitly assumes that cluster overlaps can only occur for the first object. In more detail: in ObjectGrowth, the maximal timeset for current objectset $O = \{o_i, \cdots, o_j, o_{j+1}\}$ is extracted from the maximal timeset of $O' = \{o_i, \cdots, o_j\}$ by removing the timestamps where o_{j+1} is not in O'. o_{j+1} is considered as getting together at timestamp t with O' if o_j (i.e. the last object of O') and o_{j+1} occurs in at least one cluster at t. This works correctly when there are no overlapping clusters or the overlap of clusters coincides with the first object of the search tree. However, the maximal timeset will be incorrectly calculated if the overlap of clusters contains other objects than the first object. For example, in Figure 5, the maximal timeset (timestamp sequence) of $O = \{o_1, o_2, o_3\}$ and $O' = \{o_1, o_2\}$ should be $\{t_4, t_5\}$ and $\{t_1, t_2, t_4, t_5\}$ respectively. In ObjectGrowth, the maximal timeset of $\{o_1, o_2, o_3\}$ is extracted from $\{t_1, t_2, t_4, t_5\}$. At t_2, o_2 and o_3 are in one cluster ($O'' = \{o_2, o_3\}$), and o_2 and o_1 also are in another cluster, but o_1 is not with o_3 in any cluster ($o_1 \in O'$ but $o_1 \notin O''$). However, objects o_1, o_2 and o_3 are still considered being together at t_2 since o_2 and o_3 are in the same cluster. Thus, the maximal timeset of $\{o_1, o_2, o_3\}$ computed by ObjectGrowth is $\{t_2, t_4, t_5\}$ instead of the correct answer $\{t_4, t_5\}$. The reason is that the use of the occurrence of o_j as the occurrence of O' causes problems when ObjectGrowth computes the maximal timeset. At timestamp t, if o_j and o_{j+1} are in another objectset O'' such that $\{\exists o \in O' : o \notin O''\}$, o_{j+1} is still considered in the same cluster with $O' = \{o_i, \cdots, o_j\}$ (denoted as $C_t(o_j) \cap C_t(o_{j+1}) \neq \phi$ in [9]), where $o_j = o_2$ and $o_{j+1} = o_3$ in this example. No problem will occur if $o_j = o_1$ because $O' = \{o_1\}$ and o_1 have the same occurrence. Otherwise, the maximal timeset is not correct. This issue can be addressed by considering the occurrence of the objectset rather than only the occurrence of the last object, but it potentially slows down the execution speed of ObjectGrowth since it requires extra time to match an objectset rather than an individual object. We call this modified version of ObjectGrowth as ObjectGrowth*. To compare the efficiency of PlatoonMiner against ObjectGrowth, we assign each moving object to at most one cluster at each timestamp in the datasets to ensure that ObjectGrowth correctly computes all (swarm) patterns. The datasets used in our experiment are non-overlapping by default unless we explicitly specify.

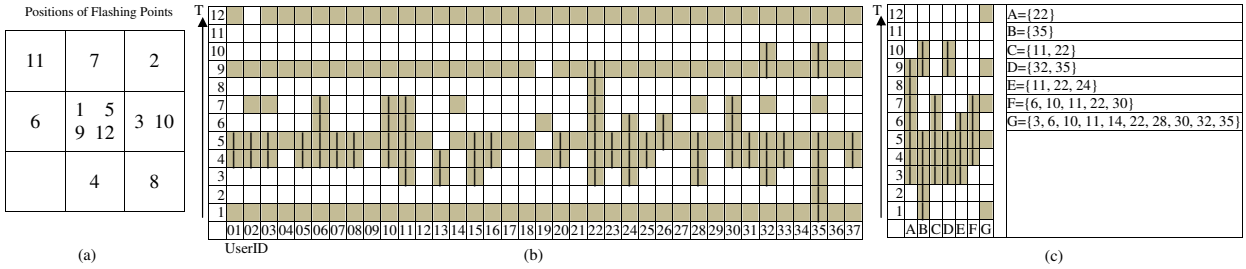


Figure 6: (a) Position of flashing point on the screen as a 3 by 3 matrix. The number in the cell represents the position of the flashing point for each of 12 timestamps. (b) Closed platoon patterns for each individual user (marked by the vertical line). Parameters: $min_o = 1$, $min_t = min_c = 2$. Each column of the matrix represents how an individual behaves during the whole period of the experiment. Swarm patterns are colored as grey. (c) Seven patterns (A-G) found based on all users except for cluster *G*. Each pattern (column) is a cluster of users having similar eye trajectories over a period of timestamps. Parameters: $min_o = 1$, $min_t = 4$ and $min_c = 2$. Patterns A-F are platoons and pattern G is a swarm.

5. Experiments

We conducted extensive experiments to evaluate the performance of PlatoonMiner by using both real-world and synthetic datasets. The efficiency of PlatoonMiner was mainly compared against ObjectGrowth [9] for non-overlapping datasets in Sections 5.1 and 5.2. ObjectGrowth is adopted as the baseline in our experiments as it is the fastest known algorithm that can mine swarm patterns. In Section 5.3, we compare PlatoonMiner versus ObjectGrowth* (c.f. Section 4.6) for overlapping datasets (and for traffic data in the non-overlapping case). We chose ObjectGrowth as our main baseline for two reasons: (1) both PlatoonMiner and ObjectGrowth require pre-clustered objects; (2) ObjectGrowth does not have any constraint on the consecutiveness of timestamps which means maximum patterns will be retrieved. This allows us to fully test the scalability of approach while performing a direct comparison with ObjectGrowth by setting $min_c = 1$. In Section 5.4, we further compare our approach with MC2 and CuTS* for mining moving object clusters with global consecutive timestamps. We did not compare PlatoonMiner with those algorithms that mine flock patterns due to their strict constraints on the shape of patterns (c.f. Section 2). The datasets we tested were not limited to having a disk shape which is required by flock patterns. All algorithms were implemented in Java (JDK 1.6) on MAC OS X 10.7.1 using an Intel Core i5 2.3 GHz machine with 8GB memory.

5.1. Evaluation On Real Datasets

We investigated the interpretability of Platoon patterns using eye movement data, and the efficiency of PlatoonMiner using traffic data.

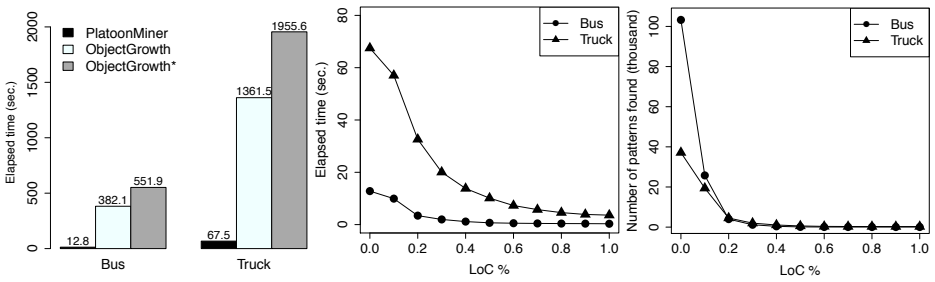
5.1.1. Eye Movement Data

We used an eye movement dataset [29] whose original task was to predict users' identities based on their eye movements. It consists of 652 labeled samples from 37 users. Eye tracking equipment obtained 2048 measures for each sample. Sampling frequency was 250Hz and measurements lasted 8192 ms. The screen the user looked at is modelled as a 3×3 matrix. A jumping flashing point on the blank screen was used as stimulus with a varying position, yielding a sequence of 12 point positions as shown in Figure 6 (a). The 8192 ms period was divided into 12 frames, with each frame lasting ~ 550 ms, except for the first and last frames which were 1600 ms and 1100 ms. At timestamps t_1, t_5, t_9 and t_{12}, the flashing point was in the center of the blank screen and directly ahead of the user's eyes. Ideally, a user's sequence of eye movements and the flashing point movements are similar trajectories. We call a *transition* of the flashing point between two consecutive timestamps a *local jump* if the change of position involves two adjacent cells in the 3×3 matrix, and otherwise, a *global jump*. Both $t_7 \rightarrow t_8$ and $t_{10} \rightarrow t_{11}$ are global jumps.

In our experiments, we consider user eyes as moving (objects) and each frame as a timestamp. Our aim is to discover how strictly the users follow the test, i.e., whether their eye positions follow the *transitions* of the flashing point. We calculate the average coordinates of each user's eye positions for each frame. At each timestamp we say that user followed the test, if at least half of the samples of the user are in the same cell as the flashing point. Although the eye movement data is a relatively small dataset, its primary propose is to show there are many significant patterns that cannot be directly obtained by existing methods.

	Bus	Truck
#objects	145	276
#timestamps	1713	2449
#data points	29202	75579
min_o	10	10
min_t	20	20
min_c	1~20	1~20
#Platoons closed; min_c =1	103247	37147
#Platoons closed; min_c =20	911	1282

(a) Configuration



(b) Elapsed time (c) Elapsed time w.r.t. *LoC* (d) #Patterns w.r.t. *LoC*

Figure 7: (a) & (b) Configuration and elapsed time for traffic datasets. (c) & (d) Effect of *LoC* on platoon queries.

To understand how individuals performed, we applied PlatoonMiner for each of the 37 users (its runtime was negligible due to the small size of the dataset). The result is shown in Figure 6 (b). We set the parameters as $min_o = 1$, $min_t = min_c = 2$. The returned platoon patterns are marked by vertical lines. To compare the Platoon patterns with swarm patterns, we also applied ObjectGrowth using $min_o = 1$ and $min_t = 2$. The returned swarm patterns are colored as grey. For these parameters, all the platoon patterns are swarm patterns but not vice versa. We see that about half of the cells in the matrix are colored (timestamps for swarm patterns), but less than half of the colored cells are marked with a cross. We observe that several of users were able to follow many transitions of the flashing point (contiguous series of crosses in a column). Notice that the swarm patterns often correspond to widely separated time points (users 4, 9, 12 and 34). Only one user obeys the test for the transition $t_7 \rightarrow t_8$ and no one follows on $t_{10} \rightarrow t_{11}$. Local jumps like $t_4 \rightarrow t_5$ and $t_5 \rightarrow t_6$ are more popular. An application that might benefit from this finding is placement of advertisements that would be best placed in adjacent cells. An analysis based solely on swarm patterns could not distinguish local jumps and global jumps, since patterns with non-consecutive time points are treated equally to patterns with consecutive time points.

In addition, Figure 6 (c) columns A-F are the complete set of closed platoons obtained by applying PlatoonMiner to all users, with the aim of discovering those behaving similarly. The right side of this figure shows the objects in each of the patterns. The parameters were set as $min_o = 1$, $min_t = 4$ and $min_c = 2$. Setting $min_o = 1$ allows us to detect a pattern consisting of a single user having many timestamps satisfying the constraint. We observe that users in patterns *C*, *D*, *E* and *F* have similar trajectories over some certain timestamps. Also, $user_{22}$ and $user_{35}$ obey the eye test the most, perhaps indicating they have the best attention span.

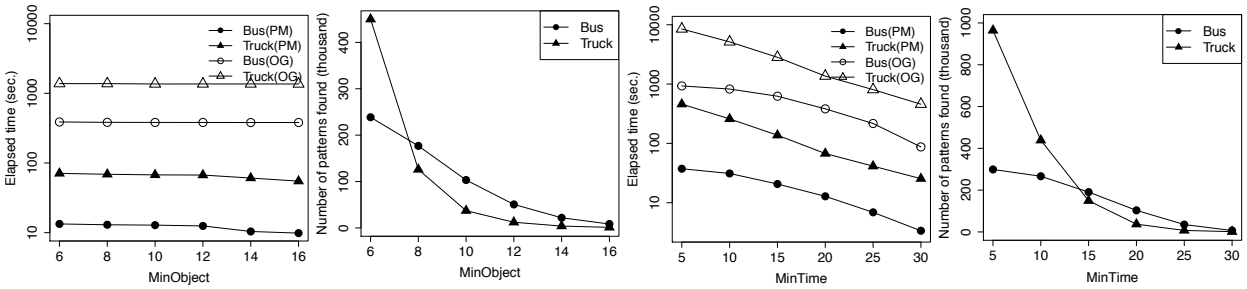
It is unlikely users would follow the test on the entire sequence of consecutive timestamps, as required by the convoy pattern. Lowering the threshold to $min_t = 4$ as the "same" setting as PlatoonMiner, the timestamp sequence for cluster *A* becomes $t_1 \rightarrow t_5$ which loses the information that they follow the transition of $t_9 \rightarrow t_{10}$. Furthermore, cluster *B* will be ignored entirely, as there are fewer than 4 consecutive timestamps in this pattern. On the other hand, some loose patterns like users in cluster *G* (which is a swarm pattern) that follow the flashing point at five isolated timestamps would be returned. Such a pattern is less informative, since isolated timestamps do not contain information about the transition of the jumping point.

5.1.2. Traffic Data

Two real-world vehicle traffic datasets were used[1]. (1) A bus dataset recording 2 school buses collecting (and delivering) students around Athens for 108 days and consisting of 145 trajectories. (2) A truck dataset recording 50 trucks delivering concrete to construction sites around Athens over 33 days and consisting of 276 trajectories.

To increase the size of moving objects, we considered each distinct trajectory as the ID of an object, yielding 145 buses and 276 trucks. This is a common method and has been used elsewhere [8]. The timestamp update frequency was set to every 30 seconds. Any second of a timestamp falling into the range [0", 30") was normalized to 15". Otherwise, it was normalized to 45". For example, the timestamp 23:22:22 gets normalized to 23:22:15, while 23:22:58 gets normalized to 23:22:45. The clusters at each timestamp are obtained by DBSCAN [17] with

[1] http://www.rtreeportal.org



(a) Elapsed time w.r.t. min_o (b) #Patterns w.r.t. min_o (c) Elapsed time w.r.t. min_t (d) #Patterns w.r.t. min_t

Figure 8: Effect of min_o and min_c on elapsed time and number of patterns (PM = PlatoonMiner, OG = ObjectGrowth). Y-axis of elapsed time is logarithmic.

$MinPoints = 3$ and $\varepsilon = 0.05$, where $MinPoints$ denotes the minimum number of objects in a cluster with a radius of ε. Figure 7 (a) details the information about these two datasets. To compare with ObjectGrowth, we use the consecutive timestamp constraint of $min_c = 1$. Thus, both methods return the complete set of platoon/swarm patterns. We set $min_o = 10$ and $min_t = 20$. The elapsed times of PlatoonMiner and ObjectGrowth for the bus and truck datasets are in Figure 7 (b). The number of closed platoons/swarms returned is equal for both methods. Compared to ObjectGrowth, PlatoonMiner is at least 20 times faster as shown in Figure 7 (b). ObjectGrowth* has relatively small overhead compared to ObjectGrowth, due to the extra time for objectset matchings.

Explanation for the performance difference

The key reason that is responsible for the performance difference between PlatoonMiner and ObjectGrowth when mining swarm patterns is the following: ObjectGrowth computes T_{max} for an objectset $O = O' \cup \{o_j\}$ by exhaustively enumerating each timestamp of the maximum timeset of O' and calculating the intersection of clusters that contain o_j and clusters that contain o_i (where o_i is the last object of O'). In PlatoonMiner, however, prefix tables are used to incrementally obtain the maximum timeset for an objectset. Other reasons including different pruning techniques are also contributing the performance difference. For example, ObjectGrowth has no pruning rule based on the threshold for the minimum number of objects. We will see in Section 5.2 that the performance difference between PlatoonMiner and ObjectGrowth will significantly increase for larger numbers of objects.

Effect of min_o and min_t thresholds

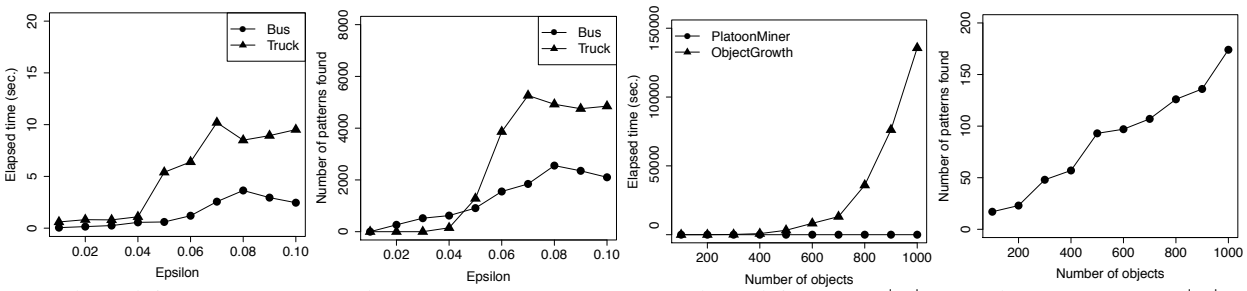
The effect of min_o and min_t on running time and number of patterns found by PlatoonMiner and ObjectGrowth is reported in Figure 8. We vary min_o (min_t) with fixed $min_t = 20$ and $min_c = 1$ ($min_o = 10$ and $min_c = 1$). The number of patterns returned decreases dramatically with increasing threshold value. For min_o, the running time of PlatoonMiner declines with larger min_o due to the shrinking search space by the Object pruning rule. In contrast, the effect of min_o on ObjectGrowth is negligible since the min_o threshold does not narrow down the search space of ObjectGrowth. For min_t, there is a significant decrease in running time of both PlatoonMiner and ObjectGrowth due to the timestamp pruning rules. The number of patterns found in the truck dataset drops more rapidly than those found in the bus dataset, as the threshold value increases.

Effect of LoC value

In addition, we performed platoon queries on these two datasets by varying the consecutiveness threshold min_c from 1 to 20. We define the level of consecutiveness (LoC) as:

$$LoC(min_c, min_t) = \begin{cases} 0 & \text{if } min_c = 1 \\ \frac{min_c}{min_t} & \text{otherwise} \end{cases} \quad (1)$$

A platoon query with $LoC = 0$ retrieves the complete set of closed swarms, while a value of $LoC = 100\%$ retrieves all convoy patterns with fixed duration of min_t. In contrast, swarm queries require a postprocessing step to obtain



(a) Elapsed time w.r.t. ε (b) #Patterns w.r.t. ε (c) Elapsed time w.r.t. $|O|$ (d) #Patterns w.r.t. $|O|$

Figure 9: (a) & (b) Effect of the setting of DBSCAN on elapsed time and number of patterns. (c) & (d) Tests on the datasets generated by Brinkhoff data generator.

platoon patterns with $LoC > 0$ due to the removal of the patterns with nonconsecutive timestamps, and convoy queries cannot mine the platoon patterns of $LoC < 100\%$. Thus, the platoon query is flexible and effective.

Figure 7 (c) shows for PlatoonMiner that elapsed time is negatively correlated with *LoC* value. Mining time for the bus dataset and truck dataset decreases from 12.8s to 0.6s and from 67.5s to only 5.4s, when *LoC* increases from 0 to 100%. In Figure 7 (d), PlatoonMiner retrieved 103247 (37147) closed swarms from the bus (truck) dataset when $LoC = 0$, but only 911 (1282) of them are convoys ($LoC = 100\%$).

Effect of the setting of DBSCAN

We also tested the effect of the DBSCAN clustering algorithm on elapsed time and number of platoon patterns by choosing different ε (the radius of a cluster). The results are shown in Figure 9 (a) and (b). We observe that there no patterns are found for a small radius, since the cluster is too small to form a moving object cluster that fulfills min_o constraint. Generally more patterns are found on clusters with bigger radii. However, since we are finding closed platoon patterns, when the cluster becomes bigger, smaller patterns will be merged into closed patterns. That is the reason why the number of patterns start to decline when $\varepsilon > 0.7$ for truck dataset and $\varepsilon > 0.8$ for bus dataset. In fact, for $\varepsilon = 0.1$, most of the timestamps only have one big cluster. Recall that DBSCAN is considered as a preprocessing step and other spatial clustering algorithms are also applicable to our methods.

5.2. Evaluation On Synthetic Datasets

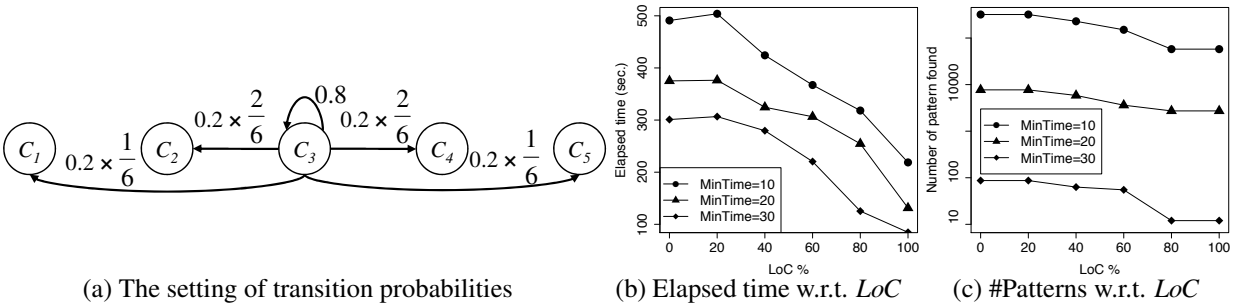
In this section, we will test PlatoonMiner and ObjectGrowth on two categories of synthetic datasets that are generated by two different data generators. (1) The Brinkoff data generator[2] , which simulates the behavior of moving objects by using a group of factors including maximum speed of objects and maximum capacity of connections. (2) Our own data generator in which we use transition probabilities to define at which cluster a moving object stays at each timestamp.

5.2.1. Benchmark: Brinkhoff Data Generator

We first ran PlatoonMiner and ObjectGrowth on the datasets generated by the Brinkhoff data generator. We used the map of Oldenburg as the input map data. In order to control the exact size of the objectset we tested, we vary the number of objects from 100 to 1000 and set the number of newly generated objects at each timestamp as zero. The maximum number of timestamps is set to 10000. In order to make moving objects last longer (thus the data has more timestamps), we set the speed divided by 250 which is the default value for slow. Other parameters were set as default. The parameters of PlatoonMiner and ObjectGrowth were set as $min_o = 5$, $min_t = 200$ and $min_c = 1$ (for PlatoonMiner). The results are shown in Figure 9 (c) and (d).

As we can see, the elapsed time of ObjectGrowth jumps as dramatically as the number of objects increases. On the other hand, the elapsed time of PlatoonMiner grows approximately linearly from 0.5 second to just 5.4 seconds (which cannot be visualized in Figure 9 (c)). The difference of elapsed time becomes more obvious when the number

[2]http://www.fh-oow.de/institute/iapg/personen/brinkhoff/



(a) The setting of transition probabilities (b) Elapsed time w.r.t. *LoC* (c) #Patterns w.r.t. *LoC*

Figure 10: (a) Transition probabilities of objects in C_3 moving from two adjacent timestamps calculated by equations 2 where $P(stay) = 0.8$ and $P(move) = 0.2$. (b) & (c) Effect of min_t and *LoC* on platoon queries.

of objects reaches 800. In addition, Figure 9 (d) indicates that the number of closed swarm patterns found increases from 17 to 174.

One limitation of using the Brinkhoff data generator to test the performance of PlatoonMiner is that the number of timestamps at which objects appear is relatively small. For example, even though we set a slow speed for moving objects, most of the moving objects will disappear after about 3000 timestamps and are only active for a short time. Therefore, to increase the volume of timestamps the datasets contain, we developed our own data generator as shown in the following section.

5.2.2. High Volume of Moving Object Data

To test the performance of PlatoonMiner on a larger scale, we developed a simulator which generates temporal object clusters with various distributions. It takes four input parameters: the number of moving objects $|O_S|$, the number of timestamps $|T_S|$, the number of clusters at initial timestamps ξ, the probability distribution $\delta = \{P(stay), P(move)\}$ of moving objects to stay in a cluster or move to another cluster at next timestamp. At t_1, the objects are assigned to the given ξ clusters equally. From current timestamp t_i to next timestamp t_{i+1}, a moving object can either stay in its current cluster C_j or move to another cluster C_k by given probability distribution δ, where $1 \leq i \leq |T_S|$ and $1 \leq j,k \leq \xi$. The transition probability is calculated by:

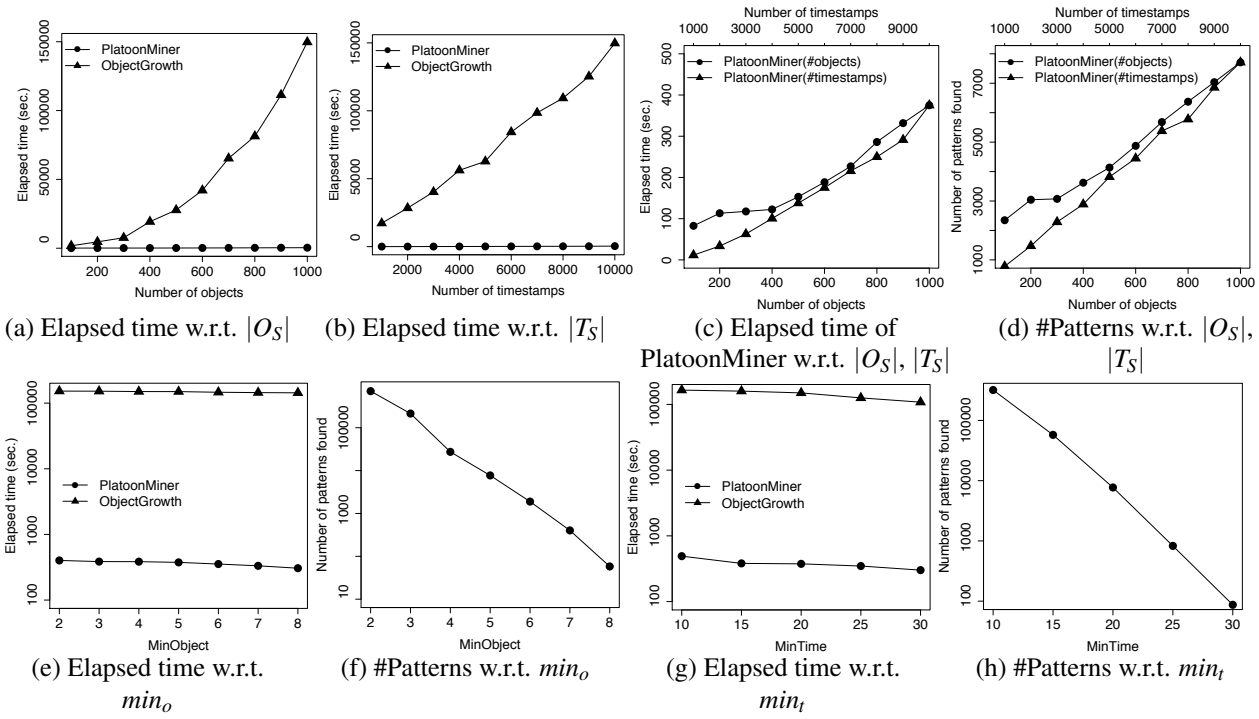
$$\begin{cases} P(C_j|C_j) = P(stay) \\ \sum_{k \neq j}^{\xi} P(C_k|C_j) = P(move) = 1 - P(stay) \\ P(C_{j\pm1}|C_j) : P(C_{j\pm2}|C_j) = 2 : 1 \end{cases} \tag{2}$$

where $P(C_k|C_j)$ denotes the event that an object moves from cluster C_j to C_k. In addition, an object is more likely to move to closer clusters rather than those further away. Figure 10 (a) gives an example of transition probabilities of objects in C_3 from timestamp t_i to timestamp t_{i+1}, where $P(stay) = 0.8$ and $P(move) = 0.2$.

In the basic setting of our experiment, we set $|O_S| = 1000$, $|T_S| = 10000$, $\xi = |O_S|/10$ (i.e. 10 objects per cluster at t_1) and $P(stay) : P(move) = 0.8 : 0.2$. Since objects exist all the time during our simulation, the total number of data points reaches 10^7 in the basic setting. For the parameters of PlatoonMiner and ObjectGrowth, we set $min_o = 5$, $min_t = 20$ and $min_c = 1$ (for PlatoonMiner). We compared the performance of PlatoonMiner against ObjectGrowth by changing one of $|O_S|$, $|T_S|$, min_o and min_t with other parameters fixed. The results are shown in Figure 11.

Effect of number of objects and timestamps

As shown in Figure 11 (a) and (b), PlatoonMiner outperforms ObjectGrowth more significantly as the size of dataset increases. The performance of ObjectGrowth is very sensitive to the number of objects, and the elapsed time rises approximately linearly with increasing number of timestamps. Figure 11 (c) shows running time of PlatoonMiner for different data sizes of data. as its behavior cannot be visualized in Figure 11 (a). The elapsed time of PlatoonMiner for increasing number of objects has a similar trend with that for an increasing number of timestamps. Both grow approximately linearly. The elapsed time of PlatoonMiner is almost proportional to the number of patterns found as the size of data increases (Figure 11 (d)). In Figure 11 (a), we observe that the PlatoonMiner is 21 times faster than



(a) Elapsed time w.r.t. $|O_S|$ (b) Elapsed time w.r.t. $|T_S|$ (c) Elapsed time of PlatoonMiner w.r.t. $|O_S|$, $|T_S|$ (d) #Patterns w.r.t. $|O_S|$, $|T_S|$

(e) Elapsed time w.r.t. min_o (f) #Patterns w.r.t. min_o (g) Elapsed time w.r.t. min_t (h) #Patterns w.r.t. min_t

Figure 11: Synthetic data using transition probabilities. Effect of $|O_S|$, $|T_S|$, min_o and min_t on elapsed time and number of closed platoons/swarms. $|O_S| = 1000$, $|T_S| = 10000$, $|min_o| = 5$ and $|min_t| = 20$. min_c is set to 1 for PlatoonMiner. Y-axes of (e), (f), (g) and (h) are logarithmic.

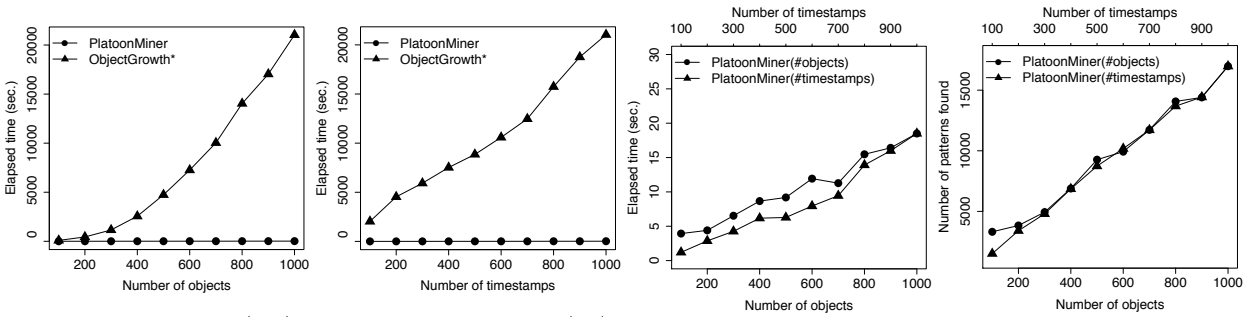
ObjectGrowth when $|O| = 100$. However, the gap elapsed time between these two algorithm increases significantly and PlatoonMiner outperforms ObjectGrowth by almost 400 times when $|O|$ grows to 1000.

Effect of min_o and min_t thresholds

Similar to the experiments on real datasets, increasing the min_o or min_t threshold requires less elapsed time, except that a larger min_o threshold does not decrease the cost of ObjectGrowth due to the lack of a pruning rule in min_o threshold, as shown in Figure 11 (e) and (g). The elapsed time of PlatoonMiner decreases from 401 seconds to 306 seconds as min_o increases. Similarly, elapsed time of PlatoonMiner decreases from 491 seconds to 301 seconds as min_t increases. Compared to the experiments on real datasets, in Figure 11 (f) and (g), the number of patterns found decreases more dramatically with an increasing threshold value. In our synthetic datasets, the probability of n objects in a cluster to stay together in next timestamp is $P(stay)^n$, while the probability of one object stay in the same clusters for m timestamp is $P(stay)^m$. As the threshold value increases, the probability drops exponentially. Note that Figure 11 (f) and (g) use a logarithmic scale.

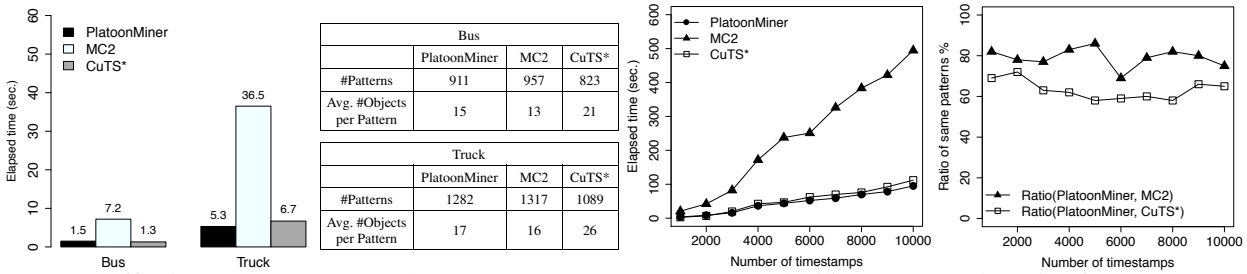
Effect of LoC value

We study the effect of *LoC* on platoon queries on the synthetic dataset composed of 10^7 data points. Figure 10 (b) and (c) reports the elapsed time of PlatoonMiner and the number of closed platoon found for combinations of min_t and *LoC*. In general, a larger min_t incurs less elapsed time and has less patterns. Running time rises slightly when *LoC* increases from 0 to 20%, since PlatoonMiner requires extra time to perform the consecutiveness check on timestamps. In contrast, the number of closed platoons remains the same in this interval, implying that all the moving objects stay in the same cluster for at least 20% of the minimum duration. In such a case, the consecutiveness check of Frequent-Consecutive pruning rule does not affect the search space but costs more elapsed time. However, the running time decreases significantly in the higher level of consecutiveness. Another observation from Figure 10 (b) is that the significant drop in elapsed time for $min_t = 30$ happens earlier than that of $min_t = 10$ and $min_t = 20$. Since a larger



(a) Elapsed time w.r.t. $|O_S|$ (b) Elapsed time w.r.t. $|T_S|$ (c) Elapsed time of PlatoonMiner w.r.t. $|O_S|$, $|T_S|$ (d) #Patterns found w.r.t. $|O_S|$, $|T_S|$

Figure 12: Datasets with overlapping clusters. Settings: $|O_S| = 1000$, $|T_S| = 1000$, $|min_o| = 5$, $|min_t| = 20$ and $min_c = 1$.



Bus			
	PlatoonMiner	MC2	CuTS*
#Patterns	911	957	823
Avg. #Objects per Pattern	15	13	21

Truck			
	PlatoonMiner	MC2	CuTS*
#Patterns	1282	1317	1089
Avg. #Objects per Pattern	17	16	26

(a) Traffic datasets (b) #Patterns (c) Scalability (d) Ratio of same patterns

Figure 13: Mining platoon patterns with global consecutive timestamps ($min_t = min_c$).

min_t has a larger number of consecutive timestamps for a fixed LoC, the elapsed time of $min_t = 30$ drops rapidly since $LoC = 40\%$ while the elapsed time of $min_t = 10$ and $min_t = 20$ decreases insignificantly until $LoC = 60\%$.

5.3. Datasets With Overlapping Clusters

We further compare the efficiency of PlatoonMiner against ObjectGrowth* as ObjectGrowth does not support overlapping clusters. The datasets were created with our data generator using the same settings as Section 5.2, except that when an object o chooses to move from cluster C_j to cluster C_k ($j \neq k$) at timestamp t_i we instead keep a copy of o in both C_j and C_k for one more timestamp. Object o is then removed from C_j at timestamp t_{i+1}. Therefore, o exists in both C_j and C_k at timestamp t_i. The experiment setting: $|O_S| = 1000$, $|T_S| = 1000$, $|min_o| = 5$ and $|min_t| = 20$. min_c is set to 1 for PlatoonMiner. Note that the number of data points in this setting is more than 10^6 since there are overlapping clusters. Results are presented in Figure 12 and we see that datasets with overlapping clusters are more complicated. As shown in Figure 12 (d), the number of patterns found in the dataset of $|O_S| \times |T_S| = 1000 \times 1000$ reaches 16978 compared to only 781 for the same size of non-overlapping dataset (left-most point in Figure 11 (d)). ObjectGrowth* takes up to 21036.3 seconds for the basic setting while PlatoonMiner only takes 18.5 seconds.

5.4. Mining Platoon Patterns with Global Consecutive Timestamps

Lastly, we compared our approach against MC2 [11] and CuTS* [8] for mining platoon patterns with global consecutive timestamps (i.e. $LoC = 100\%$). MC2 and CuTS* are originally designed to mine *moving clusters* and *convoy patterns* respectively (c.f. Section 2 for details). In order to achieve a direct comparison, the experiments were set up as follows. For PlatoonMiner, we set $min_c = min_t (LoC = 100\%)$ which means all timestamps of a pattern are required to be consecutive. For MC2, we set the percentage of common objects in clusters between two consecutive timestamps as $\theta = 1$.

The spatial clustering process is embedded in the original implementation of MC2 and CuTS*. In our experiments, we generated the clusters at each timestamp in advance for MC2 and CuTS* and the running time of pre-clustering was not counted. The trajectory simplification [8] technique used in the filter step of CuTS* was still applied.

The comparison was first carried out on the traffic datasets used in Section 5.1.2 with the same configuration except that we set $min_t = min_c = 20$ for PlatoonMiner. As shown in Figure 13 (a), our approach exhibits significantly faster elapsed time than MC2 for retrieving interesting patterns and has comparable performance with CuTS*. This is mainly due to the fact that PlatoonMiner avoids the time consuming computation of intersection and union of clusters required by MC2. Although all three algorithms used DBSCAN with the same distance threshold in the clustering process, the patterns discovered vary to an extent. The use of cluster combination technique and trajectory simplification in MC2 and CuTS* respectively are responsible for the difference. As shown in Figure 13 (b), by using trajectory simplification, CuTS* tend to have bigger objectsets in the patterns. Consequently, a smaller number of patterns are returned as some of them are combined.

We then further use the large synthetic datasets in Section 5.2.2 for comparing the scalability of three approaches. In order to apply the line simplification technique of CuTS*, the clusters generated by our simulator are plotted with data points. This allows CuTS* to use its filter step (c.f. [8]) to process the raw data. The results are demonstrated in Figure 13 (c). Again, both PlatoonMiner and CuTS* outperform significantly MC2 (about five times faster). As mentioned before, there are difference among the returned patterns. We measure the difference by computing the ratio of same patterns returned by two groups (Figure 13 (d)): (1) PlatoonMiner and MC2; (2) PlatoonMiner and CuTS*. Overall, PlatoonMiner and MC2 share about 80% of their patterns whilst PlatoonMiner and CuTS* return about 65% same patterns.

From this experiment, it is evident that PlatoonMiner can be used for simulating MC2 and CuTS* and finding *moving clusters* and *convoy patterns* with promising efficiency. We can observe that the number of patterns found are significantly less compared to the previous section due to the strict constraint on the consecutiveness of timestamps. In fact, we found that many disqualified patterns have only small gaps (one timestamp in some cases) between time segments. MC2 and CuTS* are designed to find such patterns with strict constraints. This may be a desirable feature for finding convoy patterns [8]. However, employing such strict constraint means that one can miss interesting patterns.

6. Conclusions

In this paper we have formalized the concept of platoon patterns. Unlike previously proposed patterns, the platoon query is more flexible and retrieves temporal object clusters according to different levels of temporal consecutiveness. To efficiently discover platoon patterns in a large-scale datasets, we introduced the PlatoonMiner algorithm, which employs four types of pruning rules to discover the set of closed platoons. Our experiment using eye movement data qualitatively demonstrated the utility of platoon pattern. Our experiments using other datasets showed the scalability of PlatoonMiner: it is approximately 20 and 400 times faster than ObjectGrowth, for real and synthetic datasets, respectively. In future work, we aim to investigate the use of platoon patterns for understanding co-location behaviour, in contexts where location privacy is important.

Appendix A. Proofs of Lemmas

Appendix A.1. Lemma 1

Proof 1. *If a temporal object cluster $C = (O : T)$ is not frequent, then the number of timestamps in T_C is less than min_t. Adding any object into O to form a superset O' cannot make the objects occur in more timestamps, i.e. $T'_C \subseteq T_C$. Therefore, a $C' = (O' : T')$ is not frequent either.*

Appendix A.2. Lemma 2

Proof 2. *A temporal object cluster $C = (O : T)$ is not locally consecutive implies $\exists T' \in S_{l-con}(T) \wedge l < min_c$. $C' = (O' : T')$, $O \subseteq O'$, cannot increase the number of consecutive timestamps in T'. Therefore, for $S_{l'-con}(T')$, $l' \leq l$. Also $C' = (O' : T')$ is not locally consecutive either.*

Appendix A.3. Lemma 3

Proof 3. *If O is not significant then the number of objects in O is less than min_o. Since o_i is the first object in O, in the depth-first search order, any object o_j that can be added into O must have an index $j < i$, where $1 \leq i, j \leq |O_S|$. Thus, the maximum possible number of objects can be added into O to form a descendant C' is $i-1$. If $(i-1)+|O| < min_o$, then C' is not significant.*

Appendix A.4. Lemma 4

Proof 4. *C is not a closed platoon since $O \subset O'$ and $T = T'$. For any descendant C'' of C', since $T = T'$ and $N = N'$, it implies that O always occurs together with $O' - O$. If $O'' - O$ occurs with O, then $O'' - O$ must also occur with O'. Therefore, if C'' is a platoon, then $(O'' \cup (O' - O) : T : N)$ is also a platoon which has been found before (DFS order). Thus C'' is not closed.*

Appendix A.5. Lemma 5

Proof 5. *(1) Since O' and O occur together in every timestamp in T and there is no $O'' \supset O'$ that does so, thus there exists no C''' such that $(O' \cup O) \subset O'''$ and $T = T'''$. i.e. $(O' \cup O : T : N)$ is object-maximal. Since search uses the object space, at each node we always calculate the T_{max} for current candidate, thus $T = T_{max}$ and $(O' \cup O : T : N)$ is time-maximal. Also, the infrequent or non locally consecutive candidate is removed by Frequent-Consecutive pruning rule. Therefore, $(O' \cup O : T : N)$ is a closed platoon if $|O' \cup O| > min_o$. (2) Let $C' = (X \cup O : T' : N')$ be a descendant of C that containing O', and $C'' = (X \cup O' \cup O : T'' : N'')$ be a descendant of C containing O'. Since O' and O are always in the same cluster, that means $T' = T''$. So C' is not object-maximal. Thus C' is not a closed platoon.*

Appendix A.6. Theorem 1

Proof 6. *(1) Correctness: Frequent-Consecutive pruning rule ensures that $\forall C \in R$ is frequent and locally consecutive, where R is the set of results patterns. The Object pruning rule together with the test on min_o for every candidate pattern ensures that $\forall C \in R$ is significant. Therefore, $\forall C \in R$ is a platoon pattern according to the Definition 1. Subset checking and common prefix checking ensure every pattern in R is object-maximal. Since search is based on O_S and we compute T_{max} for every current candidate, every pattern in R is time-maximal. Therefore, $\forall C \in R$ is a closed platoon. (2) Completeness: the search space of the PlatoonMiner algorithm covers all cases for candidate patterns. Lemma 1 to 5 prove that the four pruning rules only remove the redundant patterns from the search space. Thus every closed platoon is retrieved by the PlatoonMiner algorithm.*

Appendix B. The running example

Step 1: We first build the prefix table PT for the suffix ϕ (PT_ϕ) by calling Algorithm 2. After checking each object in PT_ϕ (line 8 - 15, Alg.1), object o_5 fails to satisfy the locally consecutive threshold min_c and hence the whole branch of enumeration tree can be pruned by Frequent-Consecutive pruning rule. Then we build the prefix table for the remaining objects in PT_ϕ recursively in the reversed order by calling Algorithm 3. o_4 is the first object to be extended (line 10, Alg.3), which leads to step 2.

Step 2: o_4 is the next node we visit and PT_{o_4} takes the PL_{o_4} of PT_ϕ as the input C_{DB}. After PT_{o_4} is built, we found $\{o_3\}$ is the common prefix of o_4 ($N_{con}(o_3) = N = 5$), that means $\{o_3\}$ and $\{o_4\}$ are always travel together in C_{DB}. According to Lemma 3, $(o_3, o_4 : t_1, t_2, t_3, t_4, t_5 : 5)$ is a closed platoon, and $\{o_3\}$ can be extracted from PT_{o_4}. That is, we turn PT_{o_4} into PT_{o_3,o_4} (line 11 and 22, Alg.1), and other subtrees of $\{o_4\}$ that does not contain $\{o_3\}$ ($\{o_2, o_4\}$ and $\{o_1, o_4\}$) can be pruned (subtree substitution, refer to Example 5). o_2 is the only object in PT_{o_3,o_4}, the subset checking result for $(o_2, o_3, o_4 : t_2, t_3, t_4, t_5 : 4)$ is 2, which is part of the input of step 3.

Step 3: There is no prefix object in PT_{o_2,o_3,o_4} and the closed platoon $(o_2, o_3, o_4 : t_2, t_3, t_4, t_5 : 4)$ is returned (line 18, Alg.1).

Step 4: PT_{o_3} takes PL_{o_3} of PT_ϕ as input C_{DB}. o_2 is extended first. As $C' = (o_2, o_3, o_4 : t_2, t_3, t_4, t_5 : 4)$ has been returned, the subset checking result for $C = (o_2, o_3 : t_2, t_3, t_4, t_5 : 5)$ is 1 ($O \subset O'$, $T = T'$ and $N > N'$), thus C is not a closed platoon. Node $\{o_2, o_3\}$ is extended in step 5.

Step 5: As mentioned in step 4, the subset checking result for $C = (o_2, o_3 : t_2, t_3, t_4, t_5 : 5)$ is 1 thus C is not a closed platoon (line 18, Alg.1). o_1 is the object to be extended and has a subset checking result of 2, which goes to step 6.

Step 6: $C = (o_1, o_2, o_3 : t_4, t_5 : 2)$ is returned as a closed platoon. After that, o_1 of PT_{o_3} is the next object to be extended in step 4. However, the subset checking result for platoon $C' = (o_1, o_3 : t_4, t_5 : 2)$ is 0 ($O \subset O'$ and $T = T'$ and $N = N'$) thus it (and its descendants if any) is pruned by subset checking rule (line 7, Alg.3).

Step 7: Object o_1 is the only object to be extended in PT_{o_2} and the subset checking result for $(o_1, o_2 : t_1, t_2, t_4, t_5 : 4)$ is 2 as the part of the input of step 8.

Step 8: $(o_1, o_2 : t_1, t_2, t_4, t_5 : 4)$ is returned as a closed platoon. After step 8, we go back to o_1 in PT_ϕ. Since $1 + |\phi| < min_o$, we can stop extending o_1 in PT_ϕ according to Object pruning rule.

Appendix C. Pseudo-code

Algorithm 1 PlatoonMiner

Input: Clustered trajectory database C_{DB}, suffix X, timestamp sequence T, occurrences N, minimum objects min_o, minimum timestamps min_t, minimum locally consecutive timestamps min_c, subset-checking result s

Output: R: the complete set of platoon patterns

1: $PT \leftarrow \phi$ //Construct prefix table PT for C_{DB}
2: **for** each temporal object cluster C in C_{DB} **do**
3: $o_{1st} \leftarrow$ the first object of the objectset of C
4: Call Insert-Table(o_{1st}, C, PT)
5: **end for**
6: $CP \leftarrow \phi$ //Common prefix
7: $RO \leftarrow \phi$ //Remove objects
8: **for** each object o in PT **do**
9: Call Extract-LC-Timestamps(T_{max}, min_c)
10: **if** $N_{con} = N$ **then**
11: $CP \leftarrow CP \cup \{o\}$ //Common prefix pruning
12: **else if** $|S_{min_c - con}| < min_t$ **then**
13: $RO \leftarrow RO \cup \{o\}$ //Frequent-Consecutive pruning
14: **end if**
15: **end for**
16: $RO \leftarrow RO \cup CP$
17: **if** $|CP| = 0$ **then**
18: $R \leftarrow R \cup (X, T, N)$, if $|X| > min_o$ and $s = 2$
19: **else**
20: $R \leftarrow R \cup (CP \cup X, T, N)$, if $|CP \cup X| > min_o$
//Lemma 5
21: **end if**
22: Remove the objects in RO from PT.
23: Call Suffix-Merge(PT, $CP \cup X$, min_o, min_t, min_c, R)

Algorithm 2 Insert-Table

Input: Current object o, temporal object cluster C,
prefix table PT
Output: Updated prefix table

$p \leftarrow$ prefix of o in O //where $C = (O : T : N)$
if $o \in PT$ **then**
  $T_{max} \leftarrow T_{max} \oplus T$
  if $p \in PL_o$ **then**
    $T_p \leftarrow T_p \cup T$ and $N_p \leftarrow N_p + N$
  else
    $PL_o \leftarrow PL_o \cup \{(p : T : N)\}$
  end if
else
  $PT \leftarrow PT \cup \{(o : T : N)\}$
  $PL_o \leftarrow PL_o \cup \{(p : T : N)\}$
end if
$o \leftarrow$ next object in O, if o is not the last object of O
Call Insert-Table(o, C, PT)

Algorithm 4 Suffix-Merge

Input: Prefix table PT, suffix X, min_o, min_t, min_c, R
Output: Prefix tables of children nodes

for each object o in PT in reversed order **do**
  if IndexOf$(o) + |X| < min_o$ **then**
    break //Object pruning
  end if
  $X' \leftarrow \{o\} \cup X$ //X' is the objectset of the child of X
  $C \leftarrow (X' : S_{min_c - con} : N_{con})$
  $s \leftarrow$ Subset-Checking(C, R) //Subset pruning
  if $s \neq 0$ **then**
    $C'_{DB} \leftarrow PL_o$
    Call PlatoonMiner(C'_{DB}, X', $S_{min_c - con}$, N_{con}, min_o, min_t, min_c, s)
  end if
end for

Appendix D. Discussion of Parameter Configuration

Overall, the number of patterns found increases in inverse proportion to the value of parameters min_o, min_t and min_c. The value of min_o controls the size of object clusters which can be seen as the level of granularity of the data studied. Generally speaking, a large min_o should be used for analyzing large groups such as in animal seasonal migration [2] whereas a relatively small min_o should be used for analyzing small group behaviors such as students in a class. The default value of min_o was set to 10 for PlatoonMiner in our experiments. The value of min_t indicates the extent of a pattern exists, whilst the value of min_c shows how coherent of objects in a pattern stay together over time. Obviously, the number of patterns increase in inverse proportion to the value of min_o, min_t and min_c. This inverse relationship can be used for eliminating noisy patterns. The combination of the setting of min_t and min_c controls the level of consecutiveness of timestamps (c.f. Section 5.1.2). According to our experimental results in Figure 7 (d), the gradient descent for the number of patterns found occurs at around $LoC = 10\%$. Overall, the configuration of these parameters is largely driven by the end-user application.

Algorithm 3 Extract-LC-Timestamps

Input: T_{max}, min_c
Output: Locally consecutive timestamps S_{min_c-con}, number of occurrence N_{con}

$S_{min_c-con} \leftarrow \phi$ and $N_{con} \leftarrow 0$
$T_{con} \leftarrow \phi$ and $c \leftarrow 0$ //T_{con}: consecutive timestamp
Let t_{1st} be the first timestamp in T_{max}
$j \leftarrow$ IndexOf(t_{1st})
for each timestamp t in T_{max} **do**
  $i \leftarrow$ IndexOf(t)
  if $i - j \leq 1$ **then**
    $c \leftarrow c + 1$
    $T_{con} \leftarrow T_{con} \cup \{t\}$, if $i - j = 1$
  else
    $S_{min_c-con} \leftarrow S_{min_c-con} \cup T_{con}$ and $N_{con} \leftarrow N_{con} + c$, if $|T_{con}| \geq min_c$
    $T_{con} \leftarrow \{t\}$ and $c \leftarrow 1$
  end if
  $j \leftarrow i$
end for
$S_{min_c-con} \leftarrow S_{min_c-con} \cup T_{con}$ and $N_{con} \leftarrow N_{con} + c$, if $|T_{con}| \geq min_c$

Algorithm 5 Subset-Checking

Input: Candidate object cluster C, patterns found so-far R
Output: Subset checking result s

$s \leftarrow 0$, if $\exists C' \in R$ such that $O \subset O' \wedge T = T' \wedge N = N'$;
$s \leftarrow 1$, if $\exists C' \in R$ such that $O \subset O' \wedge T = T' \wedge N > N'$;
$s \leftarrow 2$, otherwise.

Reference

[1] Y. Zheng, L. Zhang, X. Xie, W. Ma, Mining interesting locations and travel sequences from gps trajectories, in: International Conference on World Wide Web, 2009, pp. 791–800.
[2] http://www.movebank.org.
[3] T. Judd, K. Ehinger, F. Durand, A. Torralba, Learning to predict where humans look, in: International Conference on Computer Vision, pp. 2106–2113.
[4] P. Laube, S. Imfeld, Analyzing relative motion within groups of trackable moving point objects, in: International Conference on Advances in Geographic Information Systems, ACM, 2002, pp. 132–144.
[5] J. Gudmundsson, M. van Kreveld, Computing longest duration flocks in trajectory data, in: International Conference on Advances in Geographic Information Systems, ACM, 2006, pp. 35–42.
[6] M. Vieira, P. Bakalov, V. Tsotras, On-line discovery of flock patterns in spatio-temporal data, in: International Symposium on Spatial and Temporal Databases, ACM, 2009, pp. 286–295.
[7] H. Jeung, H. Shen, X. Zhou, Convoy queries in spatio-temporal databases, in: IEEE Transactions on Knowledge and Data Engineering, IEEE, 2008, pp. 1457–1459.
[8] H. Jeung, M. Yiu, X. Zhou, C. Jensen, H. Shen, Discovery of convoys in trajectory databases, in: International Conference on Very Large Data Bases, 2008, pp. 1068–1080.
[9] Z. Li, B. Ding, J. Han, R. Kays, Swarm: Mining relaxed temporal moving object clusters, in: International Conference on Very Large Data Bases, 2010, pp. 723–734.
[10] J. Gudmundsson, M. van Kreveld, B. Speckmann, Efficient detection of motion patterns in spatio-temporal data sets, in: International Conference on Advances in Geographic Information Systems, ACM, 2004, pp. 250–257.
[11] P. Kalnis, N. Mamoulis, S. Bakiras, On discovering moving clusters in spatio-temporal data, in: International Symposium on Spatial and Temporal Databases, 2005, pp. 364–381.
[12] Z. Li, B. Ding, F. Wu, T. K. H. Lei, R. Kays, M. Crofoot, Attraction and avoidance detection from movements, Proceedings of the VLDB Endowment 5 (3).

[13] J. Lee, J. Han, K. Whang, Trajectory clustering: a partition-and-group framework, in: SIGMOD Record, ACM, 2007, pp. 593–604.
[14] Y. Li, J. Han, J. Yang, Clustering moving objects, in: SIGKDD Conference on Knowledge Discovery and Data Mining, ACM, 2004, pp. 617–622.
[15] H. Kriegel, M. Pfeifle, Density-based clustering of uncertain data, in: SIGKDD Conference on Knowledge Discovery and Data Mining, ACM, 2005, pp. 672–677.
[16] C. Jensen, D. Lin, B. Ooi, Continuous clustering of moving objects, in: IEEE Transactions on Knowledge and Data Engineering, IEEE, 2007, pp. 1161–1174.
[17] M. Ester, H. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: SIGKDD Conference on Knowledge Discovery and Data Mining, ACM, 1996, pp. 226–231.
[18] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: SIGMOD Record, ACM, 2000, pp. 1–12.
[19] J. Pei, J. Han, R. Mao, et al., Closet: An efficient algorithm for mining frequent closed itemsets, in: Data Mining and Knowledge Discovery, Springer, 2000.
[20] J. Wang, J. Han, J. Pei, Closet+: Searching for the best strategies for mining frequent closed itemsets, in: SIGKDD Conference on Knowledge Discovery and Data Mining, ACM, 2003, pp. 236–245.
[21] J. Han, J. Pei, Y. Yin, R. Mao, Mining frequent patterns without candidate generation: A frequent-pattern tree approach, in: Data Mining and Knowledge Discovery, Springer, 2004, pp. 53–87.
[22] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proceedings of International Conference on Data Engineering, IEEE, Taipei, Taiwan, 1995, pp. 3–14.
[23] M. J. Zaki, Sequence mining in categorical domains: incorporating constraints, in: Proceedings of International Conference on Information and Knowledge Management, ACM, 2000, pp. 422–429.
[24] M. Zaki, Spade: An efficient algorithm for mining frequent sequences, Machine Learning 42 (1) (2001) 31–60.
[25] J. Wang, J. Han, C. Li, Frequent closed sequence mining without candidate maintenance, IEEE Transactions on Knowledge and Data Engineering 19 (8) (2007) 1042–1056.
[26] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M. Hsu, Mining sequential patterns by pattern-growth: The prefixspan approach, IEEE Transactions on Knowledge and Data Engineering 16 (11) (2004) 1424–1440.
[27] J. Han, J. Pei, Mining frequent patterns by pattern-growth: methodology and implications, ACM SIGKDD explorations newsletter 2 (2) (2000) 14–20.
[28] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, ACM SIGMOD Record 29 (2) (2000) 1–12.
[29] P. Kasprowski, J. Ober, Eye movements in biometrics, Biometric Authentication (2004) 248–258.