Discovery of Loose Group Companion From Trajectory Data Streams

THI THI SHEIN¹, (Graduate Student Member, IEEE),
SUTHEERA PUNTHEERANURAK¹, (Associate Member, IEEE),
AND MAKOTO IMAMURA², (Member, IEEE)

¹Department of Computer Engineering, Faculty of Engineering, King Mongkut’s Institute of Technology Ladkrabang, Bangkok 10520, Thailand
²Department of Embedded Technology, Tokai University, Tokyo 151-8677, Japan

Corresponding author: Sutheera Puntheeranurak (sutheera.pu@kmitl.ac.th)

This work was supported in part by the Japan International Cooperation Agency (JICA), and in part by the ASEAN University Network/Southeast Asia Engineering Education Development Network (AUN/SEED-Net).

ABSTRACT The general usability of location tracking devices has been generated a high volume of spatial-temporal data in the form of trajectory. Exploring useful knowledge from these trajectory data can contribute to understanding many real-world applications, such as traffic monitoring and weather forecasting. The main task of trajectory data analysis is the tracking of an object group movement pattern. Existing algorithms, studying the evolving structure of moving object trajectories, have high computational complexity, particularly when tracking loose group companions. To address this problem, we describe a loose group companion tracking framework over trajectory data streams in an incremental manner, which reduces computational time. Loose group companion is the moving objects group that travels together. However, some members are allowed to leave at some timestamp. A crucial part of our framework is the micro-group based loose group companion discovery. It follows a moving object group and then incrementally detects the loose group companions. We validated our techniques using two real vehicle data sets and one synthetic data set. Our approach was, on average, 45% faster than previous algorithms.

INDEX TERMS Group pattern, loose group companion, trajectory data stream, moving object clustering, spatial-temporal pattern.

I. INTRODUCTION

The increasing availability of location tracking technologies, including GPS, WLAN networks, Radio Frequency Identification, and mobile phones, has enabled tracking of a variety of moving objects, for example, vehicles, animals, and people. It results in large volumes of spatial and temporal data as the form of trajectories. Therefore, moving object tracking applications generally need to handle trajectories, when they arrive at the server, for immediate analysis and return of the result.

Past work has discussed various kinds of object group movement patterns, e.g., a group of $m$ moving objects that travel together at $k$ time intervals. Examples of this pattern are flocks [1], [2], convoys [3], [4], swarms [5], platoons [6], and gatherings [7], [8]. To identify various patterns, we need to find the moving object cluster efficiently.

The associate editor coordinating the review of this manuscript and approving it for publication was Hocine Cherifi.

Some existing clustering algorithms find the cluster from the whole trajectory database, but this leads to high computational complexity in pattern discovery. To address this problem, Tang et al. described a buddy-based clustering algorithm to accelerate the traveling companion discovery. However, in this clustering algorithm, the detection of splits and merges of buddies leads to high computational complexity, when the trajectory of each moving object changes in every snapshot(timestamp), and every buddy center needs to update in every snapshot [9], [10]. Therefore, we developed a micro-group based clustering algorithm for discovering traveling companion to reduce running time. We have published details of this algorithm [11].

There is a wide range of applications in trajectory data mining, such as path discovery, destination prediction, and movement behavior analysis for an individual or a group of moving objects. For example, in traffic management systems, the early discovery of the moving object group in vehicle trajectories can assist in avoiding traffic congestion and can...
identify regular routes. A commuter can discover a traveler with the same route to share a carpool. In biology, moving object group discovery supports the scientist, who wants to study animal migration. In weather forecasting, the scientist can forecast the weather in advance, by analyzing bird migration.

Although there are many existing applications in trajectory mining, our focus is to discover moving object group behavior in traffic management systems. In the real world, moving object trajectories always change rapidly, influenced by external influences. For example, in a university, groups of students move together to their destination. At some random times, some students leave the group to buy snacks. In traffic monitoring, a group of vehicles travels together along the trip. Some vehicles temporarily leave the group to refuel at a petrol station or a car park. Following these natural behaviors, in traveling companion discovery, an inflexible requirement for the same members along the trip. Some vehicles temporarily leave the group to refuel at a petrol station or a car park. Following these natural behaviors, in traveling companion discovery, an inflexible requirement for the same members along the trip can miss some unusual pattern.

Tang et al. [10] introduced a loose companion as a member of a group of objects that move together for a specific period, but where the member can temporarily leave the group. It calls the buddy-based loose companion discovery that still has high computational time complexity in the buddy-based clustering, and the candidate extension. Thus, we focus on finding the loose group companions efficiently from trajectory data streams. To address these problems, we adopt our micro-group based clustering and introduce pruning strategies in candidate extension for loose group companion discovery. This paper is an extension of an efficient traveling companion discovery framework [11] by using micro-group based clustering in the discovery of loose group companions. Our main contributions are:

- Introducing the loose group companion discovery over evolving trajectory data stream.
- Formulating the corresponding loose group companion discovery algorithm based on the micro-group structure and
- Evaluating effectiveness and efficiency on both real and synthetic datasets.

Fig.1 illustrates a difference between traveling companion and loose companion discovery over the data stream. Each snapshot lasts for 1 minute. If we set the candidate size threshold, \( \delta_l = 3 \), and the candidate duration threshold, \( \delta_t = 4 \) minutes, then we have only three members: \( \{o_2, o_3, o_4\} \) as traveling companions. It means we find at least three members of the moving objects group that travel together for at least 4 minutes. We miss members, \( o_1 \) and \( o_5 \) because they leave temporarily. So, we set the leave time threshold, \( \delta_l = 2 \), and \( \delta_t \) be the period that \( o_i \) leaves the group movement. Then, \( o_1 \) leaves for 1 minute (i.e., \( o_1.\tau \leq \delta_l \)) and \( o_5 \) leaves for 2 minutes (i.e., \( o_5.\tau \leq \delta_l \)) based on their distance calculation in some snapshots. After this, companion discovery returns five members: \( \{o_1, o_2, o_3, o_4, o_5\} \) as a loose group companion.

In the rest of this paper, we describe the related work in section II. Problem formulation is described in section III, and the incremental loose group companion discovery process is discussed in section IV. The experiments are described in section V and discussed in section VI. Finally, we conclude and describe future work in section VII.

II. RELATED WORK

We review previous works on trajectory data clustering and various kinds of moving object group pattern discovery from trajectory data.

A. TRAJECTORY CLUSTERING

Lee’s TRACLUS used a sub-trajectory clustering approach to detect missing sub-trajectories. In essence, it firstly partitioned a trajectory into a set of sub-trajectory line segments. It found the group of density-connected sub-trajectories [12]. Birant et al.’s ST-DBSCAN [13] used a new density-based clustering, based on DBSCAN [14], which discovered the cluster with non-spatial, spatial, temporal properties of moving objects. Due to the changes in the distribution of moving object data points with time, Amini et al. developed LeaDenStream [15]. To reduce time complexity, they chose the proper micro-cluster leader in the online phase. They sent it to an offline phase to form the final cluster.

Silva et al. developed ClUstering Trajectory Stream (CUTiS), which investigated a group of moving object trajectory data as an incremental structure [16]. To handle missing clusters in density peaks clustering (DPC), Du et al. [17] proposed a density peaks clustering based on k nearest neighbors (DPC-KNN), which adopts the idea of k nearest neighbors (KNN) into the DPC for local density computation. Moreover, Du et al. [18] also proposed a new density-adaptive metric into the DPC to tackle the assumptions of local consistency, which means nearby points are likely to have a similar local density, and global consistency, which means points on the same high-density area are likely to have the same label.

B. GROUP PATTERN MINING

Many researchers described various kinds of group patterns – flock, convoy, swarm, platoon, and gathering – to extract information from the trajectory data stream. The earliest group pattern is the flock pattern [1], which captured the
moving object group pattern within a circular region for \(k\) consecutive timestamps. Tanaka et al.'s PSI [2] sped up the discovery of flock patterns using geometric techniques, e.g., plane sweeping, binary signatures, and inverted indices. Furthermore, the circular shape of a flock cannot capture the actual moving object group. So, Jeung et al. used a convoy pattern that has an arbitrary shaped density-based cluster. These moving object groups are moved together for at least \(k\) consecutive timestamps [3]. To avoid missing convoys and invalid ones, Yoon and Shahabi developed a Valid convoy Discovery Algorithm (VcoDA) to detect a valid convoy. They firstly discovered partially connected convoys and then validated their density-connected of group members to retrieve accurate convoys [4].

However, both flocks and convoys have a rigid constraint for the continuous timestamp. Li et al. developed a time-relaxed movement pattern for moving object groups, called a swarm pattern, in which they mined closed swarm by developing the ObjectGrowth method [5]. On the other hand, a very relaxed timestamp constraint in the swarm pattern and a strict requirement on the consecutive time in the convoy discovery, which resulted in an undesired pattern in real applications. So, Li et al. designed the platoon pattern, which considered locally consecutive time constraints [6]. Wang and Lim [19] defined a group pattern, which is a group of users that are within a distance threshold from one another for at least a minimum time. To mine such a group pattern, they developed AGP and VG-growth, derived from the Apriori algorithm and FP-growth algorithm. Wang et al. [20] extended this to the uncertain group pattern - a group pattern that is discovered from the uncertain trajectory. Since the search space for the group pattern is extremely large, they designed an efficient pattern mining algorithm based on pruning. Fan et al. designed a parallel framework to discover the co-movement pattern in the form of a combination of the flock, convoy, swarm, group, and platoon [21].

Naserian et al. formed a loose traveling companion pattern (LTCP) from individual trajectory data to avoid a strict requirement on retaining the same members during a group lifetime in the area of Guangzhou Baiyun International Airport [22]. They also extended this work with more optimistic algorithms to detect the LTCP more efficiently [23]. The main difference between LTCP and our approach is that the members in LTCP leave as sub-groups. To detect such sub-groups, they used a hierarchical clustering method. Hence, there was an obstacle to discovering LTCP by applying the density-based clustering algorithm. The advantage of our loose group companion, over LTCP, is that our approach allows members to leave as individuals or as a sub-group.

Zheng et al. introduced a gathering model, which is the congregation of moving objects that last for a certain period in the form of traffic congestion [7], [8]. Zhang et al. retrieved the gathering pattern by developing a gathering and retrieving algorithm (GR) based on the Bron-Kerbosch maximum clique discovery algorithm [24]. Xian designed the gathering pattern discovery over parallel distribution fashion as batch and streaming models [25]. To avoid the rigid requirement on the number of participants in the gathering pattern, Lan et al. designed an evolving group pattern as the dense group of moving objects. They proposed a discovery framework that efficiently detects the evolving groups using sliding window technique [26]. Also, Chen et al. defined a congregate group pattern to capture various congregations from trajectory data, in the form of a dense group but allows the member to join or leave the group at any time if some members remain [27]. Zhou et al. [28] modeled multi-behavior periodic mining and single-behavior periodic mining by developing the periodic pattern detection algorithms based on spatial and temporal multi-granularity considerations. Feng and Zhu discussed the trajectory data mining techniques related to group pattern behavior analysis [29].

Table 1. summarizes some characteristics of the related pattern discovery and our approach.

| Pattern                  | Shape of Pattern | Incremental Output | Released Time constraints | Stream processing | References |
|--------------------------|------------------|--------------------|--------------------------|------------------|------------|
| Flock                    | circle           | No                 | No                       | No               | [1], [2]   |
| Convoy                   | arbitrary        | No                 | No                       | No               | [3], [4]   |
| Swarm                    | arbitrary        | No                 | Yes                      | No               | [5]        |
| Platoon                  | arbitrary        | No                 | Yes                      | No               | [6]        |
| Group                    | circle           | No                 | No                       | No               | [19]       |
| Traveling Companion      | arbitrary        | Yes                | No                       | Yes              | [9]--[11]  |
| Loose Group Companion    | arbitrary        | Yes                | Yes                      | Yes              | [10]       |
| LTCP                     | arbitrary        | Yes                | Yes                      | Yes              | [22], [23] |
| Gathering                | arbitrary        | Yes                | Yes                      | Yes              | [7], [8]   |
| Evolving Group           | arbitrary        | Yes                | Yes                      | Yes              | [26]       |
| Congregate Group         | arbitrary        | Yes                | Yes                      | Yes              | [27]       |
Table 2. Notations.

| Notation | Explanation | Notation | Explanation |
|----------|-------------|----------|-------------|
| $S$      | Trajectory data stream | $s_1, s_2, s_i$ | Snapshots in streams |
| $O_{s_i}$ | Object set at $s_i$ | $o_1, o_2, o_n$ | Objects |
| $C_{s_i}$ | Cluster set at $s_i$ | $c_1^{s_i}, c_2^{s_i}, c_m^{s_i}$ | Clusters |
| $R$      | Loose group companion candidate set | $r_1, r_2, r_v$ | Loose group companion candidates |
| $Q$      | Loose group companion set | $q_1, q_2, q_w$ | Loose group companions |
| $G_{s_i}$ | Micro-group set at $s_i$ | $g_1, g_2, g_l$ | Micro-groups |
| $\delta_s$ | Candidate size threshold | $\varepsilon$ | Micro-group distance threshold |
| $\delta_l$ | Candidate leave time threshold | $R[g_i]$ | Representative object of micro-group $g_i$ |
| $U$      | The remaining objects list (i.e., not the members of micro-group) | $\text{radius}(g_i)$ | The radius of micro-group $g_i$ |

III. PROBLEM DEFINITION

In this section, we formulate the definition of the problem used in our system. Table 2 lists the terms used in this paper.

Definition 1 (Trajectory Data Stream): A trajectory data stream $S$ consists of a sequence of snapshots $S = \{s_1, s_2, \ldots, s_l\}$ where $s_j$ is the $j^{th}$ snapshot (timestamp).

Definition 2 (Snapshot): Each snapshot $s_i$ contains a set of moving objects:

$$O_{s_i} = \{(o_1, x_1, y_1), (o_2, x_2, y_2), \ldots, (o_n, x_n, y_n)\}$$

where each object $o_i$ is associated with the location $(x_i, y_i)$.

Definition 3 (Micro-Group): Given a snapshot moving object set $O_{s_i}$, a micro-group distance threshold $\varepsilon$, and a micro-group size threshold $\gamma$, then a micro-group $g_i$ is defined as a small group of moving objects $g_i = \{o_1, o_2, \ldots, o_n\}$ with one representative object $o_1$ (denoted as $R[g_i]$) and member objects $\{o_2, \ldots, o_n\}$, where:

- $\forall o_j \in \{o_2, \ldots, o_n\}: \text{dist}(R[g_i], o_j) \leq \varepsilon$
- $\text{size}(g_i) \geq \gamma$

Definition 4 (Snapshot Cluster): Given a snapshot moving object set $O_{s_i}$, then we define $C_{s_i} = \{c_1^{s_i}, c_2^{s_i}, \ldots, c_m^{s_i}\}$ be the snapshot cluster set, where $c_j^{s_i}$ is the $j^{th}$ cluster at snapshot $s_i$. Each $c_j^{s_i}$ includes a set of micro-groups and objects.

Definition 5 (Loose Group Companion Candidate): Given a candidate size threshold $\delta_s$, a candidate duration threshold $\delta_l$, and a candidate leaves time threshold $\delta_l$, then $R = \{r_1, r_2, \ldots, r_l\}$ be the loose group companion candidate set, where $r_i$ is a loose group companion candidate if:

- $r_i$ contains more than $\delta_s$ density connected members, i.e., $\text{size}(r_i) \geq \delta_s$
- the members of $r_i$ are density connected by themselves for a certain time, where $\text{duration}(r_i) < \delta_l$
- $\forall o_j \in r_i$, let $o_j.\tau$ be the period that leaves the group, then $o_j.\tau \leq \delta_l$. In this case, the period of an object is the total number of snapshots that leaves the group. For example, in Fig. 1, the leaving period of object $o_2$ is 2 snapshots (minutes) (i.e., $o_2.\tau = 2$).

Definition 6 (Loose Group Companion): Given a candidate size threshold $\delta_s$, a candidate duration threshold $\delta_l$, and the loose group companion candidate set $R = \{r_1, r_2, \ldots, r_v\}$, then $Q = \{q_1, q_2, \ldots, q_i, \ldots, q_w\}$ be the loose group companion set, where $q_i$ is a loose group companion if:

- the duration of the loose group companion candidate $r_i$ is more than a candidate duration threshold $\delta_l$, then $r_i$ is denoted as a loose group companion $q_i$ i.e., $\text{duration}(q_i) \geq \delta_l$.

IV. LOOSE GROUP COMPANION DISCOVERY FRAMEWORK

In this section, we describe the incremental discovery of the loose group companion from the trajectory data stream. The pseudo-code is shown in Algorithm 1. Before describing the detailed steps of the algorithm, according to the variation of sampling rate in the trajectory data, we apply the linear interpolation as a preprocessing step to adjust the missing point in the trajectory data stream. There are three primary phases in the loose group companion discovery: 1) clustering phase to represent moving object group, 2) candidate extension to detect loose group companion, and 3) candidate creation.

To reduce computational time complexity in the clustering phase, we apply the micro-group based clustering algorithm that we proposed previously [11]. Moreover, to avoid the overhead intersection between candidate and cluster in candidate extension, we define the following lemmas and a definition.

Lemma 1: If the remaining number of objects in $r_i$, after intersecting with one of the current snapshot clusters is less than $\delta_s$, then further intersections with remaining clusters will not generate any required member size.
These micro-groups are set as the members of the cluster. If the distance between the representative objects of two micro-groups is no more than the sum of their radius values, then the distance between the representative objects of two micro-groups is less than \( \epsilon \).

**Algorithm 4.** In this case, there are two considerations: (1) if the cluster based on the micro-groups and the list appears, disappear, split, and merge.

**Definition 7 (Closed Cluster):** For each snapshot cluster \( c_{s_i} \), if the cluster \( c_{s_i} \) satisfies the candidate size threshold, i.e., \( \text{size}(c_{s_i}) \geq \delta_s \), then the \( c_{s_i} \) is said to be the closed cluster.

The explanations of Algorithm 1 are as follows. For the first snapshot, when the new moving objects arrive, the system performs the micro-group creation in Algorithm 2. In this algorithm, in order to create a micro-group \( g_i \), the system randomly selects the representative object among all objects that have enough neighbors of objects. It means the number of objects in \( g_i \) must be greater than or equal to \( \gamma \). This process is performed until all moving objects are set as visited. The list \( U \) keeps the remaining objects that are not included in micro-group creation.

For continuous snapshots, the system uses the maintenance structure of the micro-group in Algorithm 3. Firstly, the system gets the micro-group set from the previous snapshot. It updates the micro-groups based on the changes in their location by detecting their evolution events such as survive, appear, disappear, split, and merge.

After getting a micro-group set \( G_{s_i} \), the system creates the cluster based on the micro-groups and the list \( U \) in Algorithm 4. In this case, there are two considerations: (1) if the distance between a member object \( o_j \) of the micro-group and an object \( o_k \) in the list \( U \) is \( \leq \epsilon \), then...
Algorithm 2 Micro-Group Creation

**Input:** $O_{s_j}, \varepsilon, \gamma$

**Output:** $G_{s_j}$

1. while the object $o_j \in O_{s_j}$ is unvisited do
   2. set $o_j$ as visited and get the neighbours of $o_j$ w.r.t. $\varepsilon$;
   3. if the number of neighbours for $o_j$ is more than $\gamma$ then
      4. create the micro-group $g_i$ with all neighbours of $o_j$ and set $o_j$ as the representative object $R[g_i]$; // definition 3
      5. set all neighbours of $o_j$ as visited;
      6. micro-group appear event occur and add $g_i$ to $G_{s_j}$;
   7. if there has remaining objects that are not included in micro-groups then
      8. add them to remaining objects list $U$
   9. return $G_{s_j}$;

Algorithm 3 Micro-Group Maintenance

**Input:** $O_{s_j}, \varepsilon, \gamma$

**Output:** $G_{s_j}$

1. get the micro-group set $G_{s_{j-1}}$ from previous snapshot $s_{j-1}$;
2. for each micro-group $g_i \in G_{s_{j-1}}$ do
   3. if the representative object $R[g_i]$ exists in $O_{s_j}$ then
      4. get the neighbours of $R[g_i]$ in current snapshot $s_j$;
      5. if all neighbours of $R[g_i]$ do not change then
         6. the micro-group $g_i$ survive event occurs;
      7. if the neighbours of $R[g_i]$ contain representative of another micro-group $g_k$ then
         8. merge the micro-groups $g_i$ and $g_k$, and merge event occurs;
      9. if the size of neighbours of $R[g_i]$ is less than $\gamma$ then
         10. the micro-group $g_i$ split event occurs and add them to $O_{s_j}$;
   11. else
      12. the micro-group $g_i$ disappear event occur;
13. if there has unvisited objects in $O_{s_j}$ then
14. Initialize micro-group; // Algorithm 2
15. return $G_{s_j}$;

The system sets $o_k$ as the member of the cluster. It is shown in Fig.2(b). By using these ways, the system discovers the clusters in the form of a density-connected structure.

The candidate extension phase is done for consecutive snapshots. For the current snapshot $s_j$, the system firstly gets the loose group companion candidate set $R$ from $R_{new}$ in the previous snapshot $s_{j-1}$ (line 1). Furthermore, it performs candidate extension by intersecting between candidates and the current snapshot clusters from $C_{s_j}$ (lines 9-25). We proposed to reduce time complexity. Therefore, the algorithm intersects only the candidates and clusters that satisfy with $\delta_s$ (lines 10-15). The intersected members are assigned to the a temporary list, $v$. If there has enough common members, i.e., $size(v) \geq \delta_s$, then a new candidate $r'_{i}$ is derived, by updating member objects’ leaving period if some objects leave group, and their duration (lines 16-19). If the new candidate $r'_{i}$ is satisfied with $\delta_t$, then it is labeled as a loose group companion $q_i$ and added to the loose group companion set, $Q$ (lines 20-22). Otherwise, $r'_{i}$ added to the temporary candidate list, $R_{new}$, for the candidate extension in the next
group companion candidate. The cluster $c$ lasts for 1 minute. In the first snapshot $s$, based on Algorithm 1 is explained in Fig.3. Each snapshot procedure of incremental loose group companion discovery candidates.

not the extension of candidates) as the loose group companion candidates. Continuous snapshots, the system creates the closed clusters (i.e., tin $s_r$, finished, the intersected members from $r_1$ and $c_{s_1}$ are removed to avoid overhead intersection (line 25).

The final phase is the candidate creation (lines 26-29). For the first snapshot, the system initializes all closed clusters (definition 7) as loose group companion candidates. For continuous snapshots, the system creates the closed clusters (i.e., not the extension of candidates) as the loose group companion candidates.

Example 1: Let $\delta_s = 5$, $\delta_t = 3$, $\delta_l = 1$, $v = 2$. The running procedure of incremental loose group companion discovery based on Algorithm 1 is explained in Fig.3. Each snapshot lasts for 1 minute. In the first snapshot $s_1$, the algorithm finds the cluster $c_{s_1}$ using Algorithm 2. It initializes as a new loose group companion candidate $r_1$ (candidate creation).

At the snapshot $s_2$, the system maintains the cluster using Algorithm 3, gets two snapshot clusters $c_{s_2}^1$, $c_{s_2}^2$, performs the candidate extension with them. Firstly, the candidate $r_1$ intersected with $c_{s_2}^1$. Since the members of $r_1$ and $c_{s_2}^1$ are more than $\delta_s$ (i.e., size($r_1$) = $8 \geq \delta_s$ and size($c_{s_2}^1$) = $5 \geq \delta_s$), the intersection is performed between them and resulted as, $v = \{g_2, g_3, o_7\}$. Its members size is more than $\delta_t$ (i.e., size($v$) = $5 \geq \delta_t$). Then, the system updates $r_1$ as a new candidate $r_1'$ by updating its duration as 2 minutes and the leave time of the group members (i.e., $g_1.\tau = 1$ and $o_8.\tau = 1$) and retains them in $r_1'$ since their leave time is less than $\delta_l$.

For the intersection of $r_1$ and $c_{s_2}^2$, in our system, clusters do not overlap. In essence, each object only appears once in a single snapshot and only belongs to one cluster. The system removes the members of intersected results $v$ from $r_1$, (i.e., $r_1 - v = \{g_1, o_8\}$), since the members in the intersection can not appear again in others. To avoid unnecessary intersection between $r_1$ and $c_{s_2}^2$, the remaining members’ size of $r_1 = \{g_1, o_8\}$ is validated. Since size($r_1$) < $\delta_s$, the results of the intersection with $r_1$ cannot be generated any required candidate members (Lemma 1). Hence, the system skips the intersection between $r_1$ and $c_{s_2}^2$. Then, the cluster $c_{s_2}^2$ is initialized as a new candidate $r_2$. Both of new candidates $r_1'$ and $r_2$ are kept in $R_{new}$ for the next snapshot.

In the next snapshot $s_3$, the system firstly gets the snapshot cluster set, $C_{s_3} = \{c_{s_3}^1, c_{s_3}^2\}$, and the candidate set, $R = \{r_1', r_2\}$. The micro-group $g_1$ and the member $o_8$ are joined again to the cluster $c_{s_3}^1$. Firstly, $r_1'$ is intersected with $c_{s_3}^1$ and gets as $v = \{g_1, g_2, g_3, o_7, o_8\}$. Then, it is updated as a new candidate $r_1''$. At this time, according to the duration threshold ($\delta_t$), the candidate $r_1''$ is outputted as a loose group companion $q_1$. With $c_{s_3}^2$, the system skips the intersection as the previous snapshot.

In the case of $r_2$, it typically has to do intersection with $c_{s_3}^1$, $c_{s_3}^2$. Nevertheless, most of all members in $c_{s_3}^1$ are appeared in intersected results: $v$. After removing the intersected members from $c_{s_3}^1$ (i.e., $c_{s_3}^1 - v = 0$), the members of $c_{s_3}^1$ is less than $\delta_t$ (Lemma 2). So, the system skips this intersection to
reduce the running time. Next, the system does the intersection between $r_2$ and $c^2$, and then resulted as $r_2'$ by modifying its duration.

V. EXPERIMENT
In this section, we evaluated our system efficiency and effectiveness compared with the baseline pattern discovery methods, explained in the following.

Dataset and Parameter Settings: We used two real trajectory datasets - Trucks (D1) [30], T-Drive project (D3) [31], and one synthetic dataset (D2) [32], with parameters listed in Table.3. The ‘Obj#’ indicates the number of moving objects, and the ‘Record#’ refers to the number of GPS points (i.e., locations). For D1, it contains 63,794 GPS points of 50 trucks. The default values are selected based on high effectiveness and acceptable efficiency. The micro-group distance threshold, $\varepsilon$, is set based on the gathering pattern discovery [7] that uses the T-Drive dataset. The dataset is also the taxicab trajectory data, that are collected around the Beijing metropolitan area.

Baselines: Our Micro-group based loose group companion discovery (MU) algorithm was compared with other pattern discovery methods: 1) Buddy-based loose companion discovery (BU) [10], 2) Loose companion discovery with Smart-and-Closed method (SC) [10], 3) Loose companion discovery with Clustering- and-Intersection method (CI) [10], 4) TraClu Clustering Algorithm (TC) [12], and 5) Swarm pattern discovery(SW) [5].

Environment: Experiments used an Intel Core i7-6500 CPU 3.59 GHz and 8.00 GB of RAM and Windows 10 operating system. All the algorithms were implemented in Java using Eclipse Kepler with JDK 1.7.

A. EFFECTIVENESS COMPARISON
In this section, we determined our algorithm’s effectiveness to evaluate the discovered loose companion group. In this evaluation, we used precision and recall comparison. The retrieved results of SC are set as ground truth since SC is a commonly used baseline DBSCAN clustering algorithm. Here, precision and recall are defined as follows:

Precision: the proportion of true group companions over all the retrieved results of the algorithm.

Recall: the proportion of true group companions over the ground truth.

The evaluation was conducted on the candidate size threshold ($\delta_s$) and candidate duration threshold ($\delta_t$) over the real dataset D1. Since TC is a sub-trajectory clustering method, it was not affected by values of $\delta_s$ and $\delta_t$. So, we did not consider it in the effectiveness comparison.

| Dataset | Obj# | Record# | Duration | Sampling interval |
|---------|------|---------|----------|------------------|
| Trucks  | 50   | 63,794  | 24 hours | 1 minute         |
| Synthetic | 1586 | 363,172 | 24 hours | 1 minute         |
| T-Drive | 1164 | 412,936 | 36 hours | 5 minutes        |

Range for candidate size threshold ($\delta_s$): 10-30; default: 15
Range for candidate duration threshold ($\delta_t$): 5-30; default: 15
Range for candidate leave time threshold ($\delta_l$): 2-10; default: 6
Micro-group distance threshold ($\varepsilon$): 200 meters for D1,D3; 100 meters to D2
Micro-group size threshold ($\gamma$): 3
FIGURE 8. Efficiency: Time and Space based on $\delta_s$.

Fig. 4 shows effectiveness (i.e., precision and recall) comparison based on $\delta_s$. The precision and recall of MU and BU are very close since the companion groups that they output are nearly identical. The improvements of them were: the accuracy is $\sim$0.4 precision and $\sim$0.5 recall compared to SW, and $\sim$0.3 precision and $\sim$0.2 recall compared to CI. More than 40% of CI’s loose group companions were useless because it includes many redundant groups and cannot count group companions accurately since it did not use a candidate pruning strategy. SW generated a closed swarm pattern, i.e., a group of objects moving together that frequently meets at a specific time. Moreover, about half of the discovered closed swarm did not output as the loose group companion. Thus it reports $<0.5$ in precision and recall.

At $\delta_s = 10$, the number of loose group companion of MU is 205, and the number of loose group companion of SC as ground truth is 154. Then, the true companion is the common group between them. The number of a true companion is 144. The precision of MU is 0.702 (i.e., $\frac{144}{205}$), and the recall of MU is 0.935 (i.e., $\frac{144}{154}$).

Fig. 5 shows precision and recall of the variation of $\delta_t$. When $\delta_t$ increased, precision also increased in all algorithms. The reason is that all loose group companions travel together for a long time. Overall, for precision, MU and BU were better than CI and SW. For recall, CI was nearly identical to MU and BU, because CI also retrieved the same true companions. Thus the variation of $\delta_t$ had little effect on the resulting companion. For SW, recall increased gradually with $\delta_t$. SW could output the true loose group companions when we set tremendous $\delta_t$ values. Nevertheless, its actual recall was still low compared with others.

B. EFFICIENCY IN CLUSTERING

The efficiency of our Micro-group based clustering (MC) method was compared with the Buddy-based clustering algorithm (BC) [9], [10], TraClu (TC) [12] and DBSCAN (DC) [14]. In BC, the buddy center is updated separately, with every object entering or leaving. However, splitting and merging of the buddies in evolving trajectories made BC computationally costly. In DC, the reconstruction of the index for each object in each snapshot causes the computational complexity. In this evaluation, the two parameter values (i.e., maximum radius of the neighborhood, $eps$, and minimum number of points in an $eps$-neighborhood, minPts) for DC are set as the same values of $\varepsilon$ and $\tau$. Since TC cannot
incrementally handle the trajectories, it discovered the cluster group from the whole database.

Fig.6 compares the running time for all clustering methods on the different datasets. Overall, the running time of MC is the lowest in all the different datasets.

C. EFFICIENCY IN PATTERN DISCOVERY

To compare efficiency in pattern discovery, we evaluated running times and space costs for all datasets. The running times for SW and TC were the cost for the pattern discovery process from the whole database because it cannot output results incrementally. The space cost is measured based on the number of objects (#) that are included in the candidate sets. So, the results for TC are not included in the space comparison, because it can only group the sub-trajectories and cannot store any candidates for the pattern discovery.

Fig.7 depicts the efficiency (time and space) for all datasets based on the default settings. Overall, the SW is the highest because the ObjectGrowth algorithm discovers the pattern from the whole database, i.e., not incremental manner. The CI’s running time is higher than SC because the intersection is computed for every pair of candidates and clusters in the candidate extension.

However, the DBSCAN clustering algorithm used in SC is the slowest component. Thus, Tang et al. developed a buddy-based clustering algorithm (BU) and applied it in the discovery of the loose companion [10]. The improvement of BU over SC is shown in Fig.7. Due to the evolving nature of the trajectories, BU still has a long-running time. Therefore, in MU, we tracked the loose group companion, using our micro-group based clustering approach and the pruning strategies in candidate extension. Compared with BU, MU reduced running times by an average of 45% overall datasets.

D. EFFECT OF CANDIDATE SIZE THRESHOLD

Fig.8 compares running time and space usage on the variation of the candidate size threshold, $\delta_t$, on datasets: D1 and D3. As $\delta_t$ increased, the running time of all algorithms turned to decline because the fewer number of clusters was set as the qualified candidates. Except, the time is not affected by $\delta_t$ for TC. As $\delta_t$ increased, the number of intersections between clusters and candidates was reduced. So, the cluster and candidate matching mechanism became more effective. In SW, even though the number of clusters was reduced, the running time of SW decreased only slightly, because fully tracking of the ObjectGrowth pattern was slow.
On the other hand, the increasing $\delta_l$ directly affected to the candidate size. Thus, the space to store the candidates was also significantly reduced as $\delta_l$ increased. In summary, our MU used less time and space compared to the baseline pattern discovery.

### E. EFFECT OF CANDIDATE DURATION THRESHOLD

We also investigated the influence of candidate duration threshold, $\delta_t$, on the running time and space. Fig.9 shows the effects of $\delta_t$ on the datasets D1 and D3. The running time increased gradually with $\delta_t$. The candidate members were maintained for a longer time, i.e., until they reached $\delta_t$. Although $\delta_t$ was varied, the time and space of MU were the lowest.

### F. EFFECT OF CANDIDATE LEAVE TIME THRESHOLD

Fig.10 depicts the effect of candidate leave time threshold, $\delta_l$, on time, and space for D1 and D3. SW and TC were not included in this evaluation since SW did not allow members to leave the group, and TC could only discover the cluster. The $\delta_l$ assumed the member could leave temporarily. At that time, objects in the candidate set were not removed immediately from the candidate set and maintained in the buffer until they violated $\delta_l$. So, Fig.10 shows that as $\delta_l$ increased, time and space also increased. With increased $\delta_l$, our MU approach showed better performance than the baselines.

### VI. DISCUSSION

#### A. PARAMETERS SETTING

In the previous section, we described extensive experiments to demonstrate the efficiency of our approach. Parameter value selection played an essential role in the evaluation (both efficiency and effectiveness). So, in this section, we will explain how to set the value for each threshold parameter. There are three fundamental threshold values: 1) candidate size threshold ($\delta_s$), 2) candidate duration threshold ($\delta_t$), and 3) candidate leave time threshold ($\delta_l$).

1) **CANDIDATE SIZE THRESHOLD ($\delta_s$)**

We set the $\delta_s$ range based on the two primary considerations. The first was the size of the groups that we want to handle. The second was based on the quality of the group (i.e., effectiveness) that we want to achieve. In Fig. 4, $\delta_s = 15$ gives high precision and recall. So, we selected this value as a default for $\delta_s$. It should be noted that the performance of
around the Athens metropolitan area in Greece. In the T-Drive of 50 concrete delivery trucks heading to construction sites every minute. So, we set one minute per snapshot. It consists of reports from their positions every 30 seconds, and some reported differently for each dataset. In the truck dataset, some trucks reported its position to the server within every 10-second interval, then fixed time interval and generate a snapshot for each time step—obtaining a moving object dataset with different timestamps. The different timestamps of each object can cause difficulty in tracking objects meaningfully in the loose group companions. Most of the group members moved together for a long time and left temporarily at a particular time. Based on this, we chose the default $\delta_t = 15$ that achieved acceptable performance in both precision and recall.

2) CANDIDATE DURATION THRESHOLD ($\delta_t$)
We chose this parameter to set the maximum ‘coherence’ of a group, i.e., the time a group would stay together. Generally speaking, $\delta_t$ did not significantly affect time or space for any algorithm. However, the setting $\delta_t$ too short led to low precision as shown in Fig. 5. It was challenging to detect loose group companions. Most of the group members moved together for a long time and temporarily at a particular time. Based on this, we chose the default $\delta_t = 15$ that achieved acceptable performance in both precision and recall.

3) CANDIDATE LEAVE TIME THRESHOLD ($\delta_l$)
If we set the object to leave time, $\delta_l = 0$, then we do not allow the member to leave the group at any time and will get as a traveling companion. If the leave time threshold was too large, it did not track the loose companion accurately. In essence, a member temporarily joins another group and returns later. Thus, increasing $\delta_l$ degraded the performance of the algorithm’s result.

B. DATA SETTING
In real applications, moving objects report their positions with different timestamps. The different timestamps of each object can cause difficulty in tracking objects meaningfully and accurately in the loose group companions. So, we set a fixed time interval and generate a snapshot for each time step—with equal length snapshots. For example, if an object reports its position to the server within every 10-second interval, then we set each snapshot for 10 seconds.

In our experiment evaluation, the sampling rate was different for each dataset. In the truck dataset, some trucks reported their positions every 30 seconds, and some reported every minute. So, we set one minute per snapshot. It consists of 50 concrete delivery trucks heading to construction sites around the Athens metropolitan area in Greece. In the T-Drive dataset, the sampling time varied from 3 to 10 minutes: it was collected from GPS devices on taxis in Beijing, China. On average, we set 5 minutes for each snapshot. For the synthetic data, the data was generated with the GSTD data generator [32] at one-minute intervals.

In setting this period for each snapshot, point redundancy and missing point problems occurred in some snapshots. Point redundancy occurred when a single object reported multiple positions in a snapshot. To solve the redundancy problem, we found a common point by averaging the points. Objects sometimes missed reporting their positions due to lost connections or device faults. To handle the missing points, we used the linear interpolation to fill the gaps. Fig. 11 shows an example of the redundancy and the missing points that occur in the trajectories of three objects for three snapshots.

VII. CONCLUSION AND FUTURE WORK
We described an improved strategy for tracking loose group companions to avoid the strict requirement for members in a single group. Since the whole discovery process from the database may cause high computational time complexity, we proposed the algorithms in the form of an incremental manner. Besides, due to the evolving nature of the trajectory data stream, the cost of discovering a loose companion with the baseline discovery algorithms is still high. Thus, we firstly developed a micro-group based clustering method to reduce the time complexity in the clustering phase. Our clustering method outperformed when moving objects change their locations.

Consequently, we defined a new micro-group based loose companion discovery algorithm with candidate pruning strategies. Our proposed algorithm also improved the efficiency of the loose group companion discovery. As a result, our micro-group based loose group companion discovery was about 45% faster than Tang et al.’s buddy-based loose companion discovery.

In the traffic monitoring system, this research benefits the authorities in understanding the routes of a group of vehicles and helps to avoid traffic congestion by detecting the loose group companions in advance. In future work, we will discover some challenging moving object group patterns as real-world traffic scenarios to be used in the traffic monitoring system.

REFERENCES
[1] M. R. Vieira, P. Bakalov, and V. J. Tsotras, “On-line discovery of flock patterns in spatio-temporal data,” in Proc. 17th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inf. Syst. (GIS), Sydney, NSW, Australia, 2009, pp. 286–295.
[2] P. S. Tanaka, M. R. Vieira, and D. S. Kaster, “An improved base algorithm for online discovery of flock patterns in trajectories,” J. Inf. Data Manag., vol. 7, no. 1, pp. 52–67, Oct. 2016.
[3] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, “Discovery of convoys in trajectory databases,” Proc. VLDB Endowment, vol. 1, no. 1, pp. 1068–1080, Aug. 2008.
[4] H. Yoon and C. Shahabi, “Accurate discovery of valid convoys from moving object trajectories,” in Proc. IEEE Int. Conf. Data Mining Workshops, Dec. 2009, pp. 636–643.
[10] T. Chen, Q. Liu, and C. Xu, “Online discovery of gathering patterns over trajectories,” IEEE Trans. Knowl. Data Eng., vol. 28, no. 5, pp. 1147–1159, May 2016.

[11] A. Amini and T. Y. Wah, “LeaDen-stream: A leader density-based clustering approach for mining co-movement patterns over large-scale trajectories,” Proc. VLDB Endowment, vol. 10, no. 4, pp. 313–324, Nov. 2016.

[12] E. Naserian, X. Wang, X. Xu, and Y. Dong, “Discovery of loose travelling companion patterns from human trajectories,” in Proc. IEEE 18th Int. Conf. High Perform. Comput. Commun., IEEE 14th Int. Conf. Smart City, IEEE 2nd Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS), Dec. 2016, pp. 1238–1245.

[13] T. T. Shein (Graduate Student Member, IEEE) received the M.Eng. degree in electrical engineering from the King Mongkut’s Institute of Technology Ladkrabang, Bangkok, Thailand. Her research interests include data analytic, data mining, and recommendation systems.