A Framework of Traveling Companion Discovery on Trajectory Data Streams

LU-AN TANG, University of Illinois at Urbana-Champaign and Microsoft Research Asia
YU ZHENG and JING YUAN, Microsoft Research Asia
JIAWEI HAN, University of Illinois at Urbana-Champaign
ALICE LEUNG, BBN Technologies
WEN-CHIH PENG, National Chiao Tung University
THOMAS LA PORTA, Pennsylvania State University

The advance of mobile technologies leads to huge volumes of spatio-temporal data collected in the form of trajectory data streams. In this study, we investigate the problem of discovering object groups that travel together (i.e., *traveling companions*) from trajectory data streams. Such technique has broad applications in the areas of scientific study, transportation management, and military surveillance. To discover traveling companions, the monitoring system should cluster the objects of each snapshot and intersect the clustering results to retrieve moving-together objects. Since both clustering and intersection steps involve high computational overhead, the key issue of companion discovery is to improve the efficiency of algorithms. We propose the models of closed companion candidates and smart intersection to accelerate data processing. A data structure termed *traveling buddy* is designed to facilitate scalable and flexible companion discovery from trajectory streams. The traveling buddies are microgroups of objects that are tightly bound together. By only storing the object relationships rather than their spatial coordinates, the buddies can be dynamically maintained along the trajectory stream with low cost. Based on traveling buddies, the system can discover companions without accessing the object details. In addition, we extend the proposed framework to discover companions on more complicated scenarios with spatial and temporal constraints, such as on the road network and battlefield. The proposed methods are evaluated with extensive experiments on both real and synthetic datasets. Experimental results show that our proposed buddy-based approach is an order of magnitude faster than the baselines and achieves higher accuracy in companion discovery.

Categories and Subject Descriptors: H.2.8 [**Database Applications**]: Data Mining

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Trajectory, data stream, clustering

L.-A. Tang is currently affiliated with NEC Labs America.

The work was supported in part by U.S. NSF grants IIS-0905215, CNS-0931975, CCF-0905014, IIS-1017362, the U.S. Army Research Laboratory under Cooperative Agreement no. W911NF-09-2-0053 (NS-CTA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government.

Authors' addresses: L.-A. Tang (corresponding author), University of Illinois at Urbana-Champaign, 601 E John St, Champaign, IL 61820; email: leon.tang82@gmail.com; Y. Zheng and J. Yuan, Microsoft Research Asia; J. Han, University of Illinois at Urbana-Champaign, 601 E John St, Champaign, IL 61820; A. Leung, BBN Technologies; W.-C. Peng, National Chiao Tung University, 300, Taiwan, Hsinchu City, Dong District, Taiwan, ROC; T. La Porta, Pennsylvania State University, 201 Old Main, University Park PA 16802.

DOI: http://dx.doi.org/10.1145/2542182.2542185

ACM Reference Format:
Tang, L.-A., Zheng, Y., Yuan, J., Han, J., Leung, A., Peng, W.-C., and Porta, T. L. 2013. A framwork of traveling comanion discovrey on trajectory data streams. ACM Trans. Intell. Syst. Technol. 5, 1, Article 3 (December 2013), 34 pages.
DOI: http://dx.doi.org/10.1145/2542182.2542185

1. INTRODUCTION

The technical advances in mobile devices and tracking technologies have generated huge amount of trajectory data recording the movement of people, vehicles, animals, and natural phenomena in a variety of applications, such as social network, transportation management, scientific studies, and military surveillance [Zheng and Zhou 2011]: (1) In Foursquare[1], the users check in the sequence of visited restaurants and shopping malls as trajectories. In many GPS-trajectory-sharing Web sites like Geolife [Zheng et al. 2010], people upload their travel or sports routes to share with friends. (2) Many taxis in major cities have been embedded with GPS sensors. Their locations are reported to the transportation system in the format of streaming trajectories [Yuan et al. 2010; Tang et al. 2011]. (3) Biologists solicit the moving trajectories of animals like migratory birds for their research[2]. (4) The battlefield sensor network watches the designated area and collects the movement of possible intruders [Tang et al. 2010]. Their trajectories are watched by military satellites all the time.

In the aforementioned applications, people usually expect to discover the object groups that move together, that is, *traveling companions*. For example, commuters want to discover people with the same route to share car pools. Scientists would like to study the pathways of species migration. Information about traveling companions can also be used for resource allocation, security management, infectious disease control, and so on.

Despite wide applications, the discovery of traveling companions is not efficiently supported in existing systems, partly due to the following challenges.

—*Colocation*. Companions are objects that travel together. Here “travel together” means the objects are spatially close at the same time. Many state-of-the-art trajectory clustering methods, retrieving the object’s major moving direction from their trajectories, ignore the temporal information of objects [Lee et al. 2007; Har-Peled 2003; Li et al. 2004; Yang et al. 2009; Zhang and Lin 2004; Jensen et al. 2007]. Hence they cannot be directly used for companion discovery.

—*Incremental discovery*. In several applications like military surveillance, the system needs to monitor objects for a long time and discover companions as soon as possible. Hence the algorithm should report the companions in an incremental manner, that is, output the results simultaneously while receiving and processing the trajectory data stream.

—*Efficiency*. Most trajectories are generated in a format of data stream. Huge amounts of data arrive rapidly in a short period of time. The monitoring system has to cluster the data and intersect the clusters for companions. These steps involve high computational overhead. The algorithm should develop efficient data structures to process large-scale data.

—*Effectiveness*. The number of companions is usually large. The system should report the large and long-lasting companions rather than small and short-time ones.

[1] http://foursquare.com.

[2] http://www.movebank.org.

The companion discovery algorithm should be effective to select the most important results.

—*Spatio-temporal constraints*. In real applications, objects move with several spatial and temporal constraints, such as where a vehicles travel along the road network, the military objects need to follow certain orders to leave the team for a short time. The algorithm should be adapted for such constraints to improve the system feasibility and applicability.

We are aware that several studies have retrieved object groups similar to the traveling companions, such as flock [Gudmundsson and Kreveld 2006], convoy [Jeung et al. 2008], and swarm [Li et al. 2010]. However, most of them are designed to work on static datasets on 2D Euclidean space, and some methods need multiple scans of the data, or cannot output results in an incremental manner. Hence it is still desirable to provide high-quality but less costly techniques for companion discovery on trajectory streams with spatio-temporal constraints.

In this study, we investigate the models, principles, and methodologies to discover traveling companions from trajectory streams. Since the objects keep on moving in the trajectory streams, it is hard to maintain an index for their spatial positions. However, the relationships among most objects are gradual evolutions rather than fierce mutations. The *traveling buddy* is proposed to store the relationship. Such a model can be easily maintained along the data streams. Thus, in this article, we explore the traveling-buddy-based companion discovery, which is able to discover companions without accessing the object details and significantly improve the system's efficiency. The main contributions of this study include: (1) introducing the companion models to define the problem; (2) proposing the concepts of smart intersection and closed companions to accelerate data processing; (3) analyzing the bottleneck of the problem and proposing a traveling-buddy-based approach; (4) extending the proposed methods to complicated scenarios with spatio-temporal constraints, developing the methods to discover the road companions and loose companions; and (5) demonstrating the scalability and feasibility of the proposed methods by experiments on both real and synthetic datasets.

This article substantially extends the version on ICDE 2012 conference [Tang et al. 2012], in the following ways: (1) introducing the concepts of *road companion* and *loose companion* to model the companion discovery problems on more complicated scenarios; (2) analyzing the main bottleneck of road companion discovery and proposing a filtering-and-refinement-based framework; (3) designing new algorithms with the *road buddy* for efficient companion discovery on road networks; (4) proposing the leaving time threshold and introducing the concept of *loose companion* to release the time constraints for more effective companion discovery; (5) carrying out the time complexity analysis for proposed algorithms; (6) providing complete formal proofs for lemmas and propositions; (7) covering the related work in more details and including recent ones; and (8) expanding our performance studies on datasets on road networks and battlefield. The experimental results show that the new proposed methods are an order of magnitude faster than the old ones in Tang et al. [2012].

The rest of the article is organized as follows. Section 2 defines the problem; Section 3 introduces the general framework of companion discovery; Section 4 proposes the traveling-buddy-based method; Section 5 extends the proposed methods to discover companions on road networks; Section 6 discusses the techniques to discover companions with released temporal constraints; Section 7 evaluates the algorithms' performances; Section 8 gives a survey of the related work and finally in Section 9 we conclude the work.

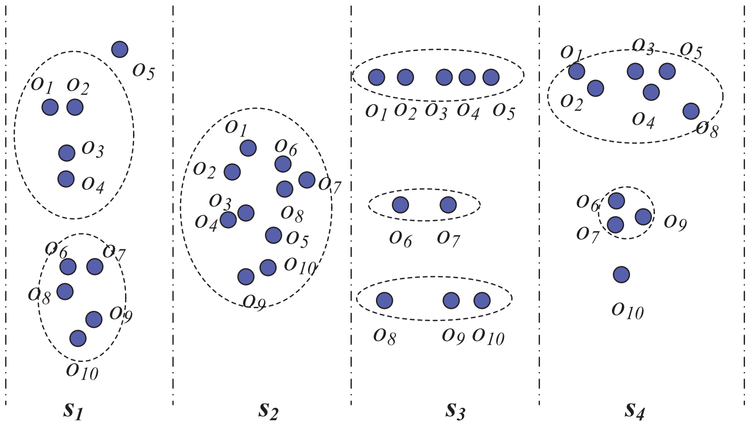


Fig. 1. Example: discover traveling companions.

2. PROBLEM DEFINITION

In the various applications of traveling companion, there are some common principles shared in different scenarios. We illustrate the characteristics of companion discovery by the following example.

Example 1. Ten objects are tracked by a monitoring system. Figure 1 shows their positions in four snapshots. There are three key issues to discover the companions.

—*Cluster*. The companions are the objects that travel together, that is, in the same cluster. Since the people, vehicles, and animals often move and organize in arbitrary ways, the companion shape is not fixed. In Figure 1, the objects are grouped in round shape in snapshots s_1 and s_2, while in s_3, they are moving in a queue and the companions are formed as thin and long ellipses.

—*Consistency*. The companions should be consistent enough to last for a few snapshots. This feature makes it possible to find the companions by intersecting the clusters of different snapshots.

—*Size*. Most users are only interested in object groups that are big enough. They may have requirements on the companion's size. For example, if the user sets the size threshold as four and requires the companion to last for at least four snapshots, then $\{o_1, o_2, o_3, o_4\}$ is the result companion.

To discover the traveling companions with various shapes, we employ the concepts of density-based clustering [Ester et al. 1996] in this study.

Definition 1 (*Direct Density Connection*). Let O be the object set in a snapshot, ε be the distance threshold, μ be the density threshold, and $N_\varepsilon(o_i) = \{o_j \in O \mid dist(o_i, o_j) \le \varepsilon\}$. Object o_j is directly density connected from object o_i if $o_j \subset N_\varepsilon(o_i)$ and $|N_\varepsilon(o_i)| \ge \mu$.

Definition 2 (*Density Connection*). Let O be the object set in a snapshot, object o_i is density connected to object o_j, if there is a chain of objects $\{o_1, \ldots, o_n\} \in O$ where $o_1 = o_j$, $o_n = o_i$ such that o_{i+1} is directly density connected from o_i.

With the concepts of density connection, we formally define the traveling companion as follows.

Definition 3 (*Traveling Companion*). Let δ_s be the size threshold and δ_t be the duration threshold, a group of objects q is called *traveling companion* if:

Notation	Explanation	Notation	Explanation
S	the trajectory stream	s, s_i, s_j	the snapshots in stream
C	the cluster set	c_i, c_j	the clusters
Q	the companion set	q	the traveling companion
R	the candidate set	r_i, r_j	the companion candidates
B	the buddy set	b_i, b_j	the traveling buddies
O	the object set	o_1, o_2, o_i	the objects
ε	the distance threshold	μ	the density threshold
δ_s	the size threshold	δ_t	the duration threshold
δ_γ	the buddy radius threshold	γ_i, γ_j	the buddy radius
M	the road network	δ_l	the leaving time threshold

Fig. 2. List of notations.

(1) the members of q are density connected by themselves for a period t where $t \geqslant \delta_t$;
(2) q's size $size(q) \geqslant \delta_s$.

Problem Definition. Let trajectory data stream S be denoted by a sequence of snapshots $\{s_1, s_2, \ldots, s_i, \ldots\}$. Each snapshot $s_i = \{(o_1, x_{1,i}, y_{1,i}), (o_2, x_{2,i}, y_{2,i}), \ldots, (o_n, x_{n,i}, y_{n,i})\}$, where $x_{j,i}$, $y_{j,i}$ are the spatial coordinates of object o_j at snapshot s_i. When the data of snapshot s_i arrives, the task is to discover companion set Q that contains all the traveling companions so far.

We will introduce the framework and techniques for companion discovery in the next few sections. Figure 2 lists the notations used throughout this article.

3. COMPANION DISCOVERY FRAMEWORK

3.1. The Clustering-and-Intersection Method

A general framework of *clustering and intersection* is proposed in Gudmundsson and Kreveld [2006] and Jeung et al. [2008] to retrieve the convoy patterns. This framework can also be adapted to discover companions on trajectory streams: The idea is to retrieve *companion candidates* by counting common objects in the clusters from different snapshots. The system keeps clustering the objects in coming snapshots and intersecting them with the stored candidates. In this way the candidates are gradually refined to become resulting companions.

Definition 4 (*Companion Candidate*). Let δ_s be the size threshold and δ_t be the duration threshold, a group of objects r is a companion candidate if:

(1) the members of r are density connected by themselves for a period t where $t < \delta_t$;
(2) $size(r) \geqslant \delta_s$.

Intuitively, the companion candidates are the object groups with enough size but shorter duration. The candidate's size reduces when intersecting with the clusters from other snapshots, but its lasting time increases. Once a candidate's time grows longer than the threshold, it will be reported as a traveling companion. Meanwhile, as soon as the candidate is not large enough, it is no longer qualified and should be removed from memory. Figure 3 lists the steps of the clustering-and-intersection algorithm.

Algorithm 1 first performs density-based clustering for all the objects in coming snapshots (lines 1–3). Then the system refines companion candidates by intersecting them with new clusters (lines 4–7). The intersection results with enough size are

ALGORITHM 1: Clustering-and-Intersection
Input: size threshold δ_s, duration threshold δ_t, distance threshold ε, density threshold μ, candidate set R and the trajectory data stream S
Output: every qualified companion q

```
1.  for each coming snapshot $s$ of $S$
2.      initialize new candidate set $R'$;
3.      cluster the objects in $s$ w.r.t to $\varepsilon$ and $\mu$;
4.      for each candidate $r_i \in R$, do
5.          for each cluster $c_j \in s$, do
6.              new candidate $r_i' \leftarrow r_i \cap c_j$;
7.              $duration(r_i') = duration(r_i) + duration(s)$;
8.              if size($r_i'$) $\geq \delta_s$ then
9.                  add $r_i'$ to $R'$;
10.                 if $duration(r_i') \geq \delta_t$ then
11.                     output $r_i'$ as a qualified companion $q$;
12.     add all the new clusters to $R'$;
13.     $R \leftarrow R'$;
```

Fig. 3. Algorithm: the clustering-and-intersection method.

stored as new candidates (lines 8–9). The ones with enough duration are reported as traveling companions (lines 10–11). The new clusters are added to the candidate set (line 12). At last the candidate set R is updated to process following snapshots (line 13).

Proposition 1. *Let n_1 be the size of objects and n_2 be the total size of candidate set R. The time complexity of Algorithm 1 is $O(n_1^2 + n_1 * n_2)$.*

Proof. In the clustering step, the algorithm needs $O(n_1^2)$ time to generate density-based clusters[3]. In the intersection step, suppose there are average m_1 clusters and m_2 candidates, the system carries out $m_1 * m_2$ intersections, and the intersection takes $l_1 * l_2$ time, where l_1 is the average cluster size and l_2 is the average candidate size. Since $m_1 * l_1 = n_1$, $m_2 * l_2 = n_2$, thus the time complexity of the intersection step is $O(m_1 * m_2 * l_1 * l_2) = O(n_1 * n_2)$ and the total time complexity is $O(n_1^2 + n_1 * n_2)$. □

Example 2. Figure 4 shows the running process of the clustering-and-intersection algorithm. Suppose each snapshot lasts for 10 minutes, the size threshold is 3 and the time threshold is 40 minutes. The objects are first clustered in each snapshot. Two clusters in s_1 are taken as the candidates, namely r_1 and r_2. Then they are intersected with the clusters in s_2, meanwhile, the cluster of s_2 is also added as a new candidate r_3. The clustering and intersection steps are carried out in each snapshot. Finally, the algorithm reports $\{o_1, o_2, o_3, o_4\}$ as a traveling companion in s_4. The total intersection times are 29, and the largest candidate set R appears in s_3 with 23 objects involved.

3.2. The Smart-and-Closed Algorithm

The computational overhead of the clustering-and-intersection method is high in both time and space. In each snapshot, the intersection is carried out in every pair of candidate and cluster. However, most intersections cannot generate qualified results with enough size. In this section we introduce the methods to improve the efficiency: (1) the

[3] The clustering process can be improved to $O(n_1 * logn_1)$ with a spatial index, however, it is costly to maintain such a spatial index in each time snapshot [Lee et al. 2003].

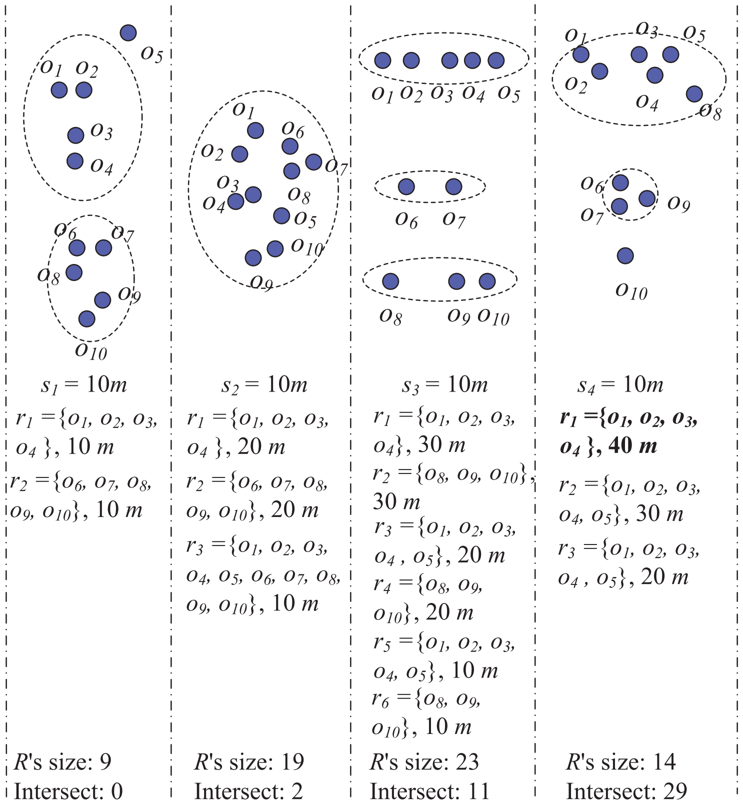

o_1 o_2 o_3 o_4 o_5 o_6 o_7 o_8 o_9 o_{10}
$s_1 = 10m$
$r_1 = \{o_1, o_2, o_3, o_4\}$, 10 m
$r_2 = \{o_6, o_7, o_8, o_9, o_{10}\}$, 10 m
R's size: 9
Intersect: 0
$s_2 = 10m$
$r_1 = \{o_1, o_2, o_3, o_4\}$, 20 m
$r_2 = \{o_6, o_7, o_8, o_9, o_{10}\}$, 20 m
$r_3 = \{o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8, o_9, o_{10}\}$, 10 m
R's size: 19
Intersect: 2
$s_3 = 10m$
$r_1 = \{o_1, o_2, o_3, o_4\}$, 30 m
$r_2 = \{o_8, o_9, o_{10}\}$, 30 m
$r_3 = \{o_1, o_2, o_3, o_4, o_5\}$, 20 m
$r_4 = \{o_8, o_9, o_{10}\}$, 20 m
$r_5 = \{o_1, o_2, o_3, o_4, o_5\}$, 10 m
$r_6 = \{o_8, o_9, o_{10}\}$, 10 m
R's size: 23
Intersect: 11
$s_4 = 10m$
$r_1 = \{o_1, o_2, o_3, o_4\}$, 40 m
$r_2 = \{o_1, o_2, o_3, o_4, o_5\}$, 30 m
$r_3 = \{o_1, o_2, o_3, o_4, o_5\}$, 20 m
R's size: 14
Intersect: 29

Fig. 4. Example: the clustering-and-intersection method.

smart algorithm stops the intersection step early once it is impossible to generate qualified candidates, and (2) the closed candidates are used to help reduce the memory cost.

LEMMA 1. *Let r be a companion candidate and δ_s be the size threshold, if there are more than $size(r) - \delta_s$ objects of r already appearing in intersected clusters, continuously intersecting r with remaining clusters will not generate any meaningful results with size larger than δ_s.*

PROOF. Since each object only appears once in a single snapshot and only belongs to one cluster[4], if there are more than $size(r) - \delta_s$ objects appearing in already intersected clusters, even in the best case (all the remaining objects are in a single cluster), the intersection result will still be smaller than $size(r) - (size(r) - \delta_s) = \delta_s$. □

Lemma 1 can be used to improve the candidate refining process with smart intersection. Once an object is found in the cluster, the algorithm removes it from the candidate. The intersection process will stop earlier if there are less than δ_s objects remaining in the candidate.

Another problem of the clustering-and-intersection method is the space efficiency; if all new clusters are added as candidates, the size of the candidate set will increase rapidly as the trajectory stream passes-by. Such a huge candidate set is a

[4] The clustering methods used in this study are all "hard-clustering", that is, an object can only belong to one cluster.

ALGORITHM 2: Smart-and-Closed Algorithm
Input: size threshold δ_s, duration threshold δ_t, distance threshold ε, density threshold μ, candidate set R and the trajectory data stream S
Output: every qualified companion q

```
1.  for each coming snapshot $s$ of $S$
2.      initialize new candidate set $R'$;
3.      cluster the objects in $s$ w.r.t to $\varepsilon$ and $\mu$;
4.      for each candidate $r_i \in R$, do
5.          for each cluster $c_j \in s$, do
6.              if $r_i$'s size is less than $\delta_s$ then break;
7.              new candidate $r_i' \leftarrow r_i \cap c_j$;
8.              $duration(r_i') = duration(r_i) + duration(s)$;
9.              remove intersected objects from $r_i$;
10.             if size($r_i'$) $\geq \delta_s$ then
11.                 add $r_i'$ to $R'$;
12.                 if $duration(r_i') \geq \delta_t$ then
13.                     output $r_i'$ as a qualified companion $q$;
14.         for each cluster $c_j$ do
15.             if $c_j$ is closed then add to $R'$;
16.     $R \leftarrow R'$;
```

Fig. 5. Algorithm: the smart-and-closed discovery.

burden for system memory. In the worst case, all the clusters stay constant in the series of snapshots, the intersection process cannot prune any existing candidates, and all the new clusters are added to the candidate set. After m snapshots, the system needs to maintain an $m * n$ size candidate set, where n is the number of objects.

In Figure 4, candidates r_3 and r_5 in s_3 contain the same objects with different lasting time. In such cases, the system only needs to store the one with longer time (e.g., r_3). Such candidates like r_3 are called *closed candidates*.

Definition 5 (*Closed Candidate*). For a companion candidate r_i, if there does not exist another candidate r_j such that $r_i \subseteq r_j$, and r_i's duration is less than r_j's duration, then r_i is a *closed candidate*.

Armed with Lemma 1 and Definition 5, we propose the smart-and-closed algorithm. The modifications are underlined in Figure 5: the algorithm removes intersected objects from the candidate set and checks its remaining size before the next intersection (lines 5 and 9); when adding the new clusters to the candidate set, the algorithm always checks if there is already a candidate containing the same objects but with longer duration, and only the ones passing the closeness check are added as new candidates (lines 14–15).

In the worst case, Algorithm 2 cannot prune any candidates and the time complexity is the same as Algorithm 1. However, we find out that the smart-and-closed algorithm can save about 50% time and space in the experiments.

Example 3. Figure 6 shows the running process of the smart-and-closed algorithm. In snapshot s_3, when making intersections for candidate r_1 with three clusters, the process ends early after the first round. Since the system only stores closed candidates, the largest candidate set size is only 19 in s_2, and the total intersection time is 12, less than half of the cost in the clustering-and-intersection algorithm.

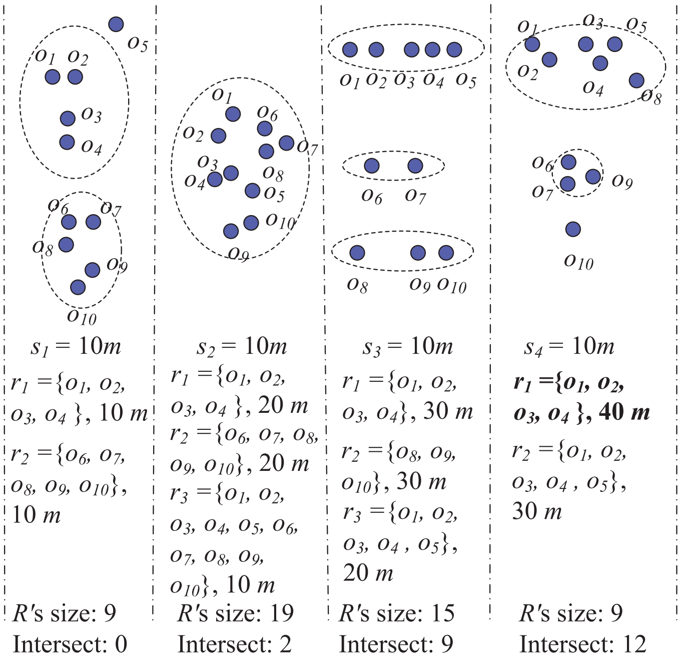

o_1 o_2 o_3 o_4 o_5 o_6 o_7 o_8 o_9 o_{10}
$s_1 = 10m$
$r_1 = \{o_1, o_2, o_3, o_4\}$, 10 m
$r_2 = \{o_6, o_7, o_8, o_9, o_{10}\}$, 10 m
R's size: 9
Intersect: 0
$s_2 = 10m$
$r_1 = \{o_1, o_2, o_3, o_4\}$, 20 m
$r_2 = \{o_6, o_7, o_8, o_9, o_{10}\}$, 20 m
$r_3 = \{o_1, o_2, o_3, o_4, o_5, o_6, o_7, o_8, o_9, o_{10}\}$, 10 m
R's size: 19
Intersect: 2
$s_3 = 10m$
$r_1 = \{o_1, o_2, o_3, o_4\}$, 30 m
$r_2 = \{o_8, o_9, o_{10}\}$, 30 m
$r_3 = \{o_1, o_2, o_3, o_4, o_5\}$, 20 m
R's size: 15
Intersect: 9
$s_4 = 10m$
$\boldsymbol{r_1 = \{o_1, o_2, o_3, o_4\}}$, **40 *m***
$r_2 = \{o_1, o_2, o_3, o_4, o_5\}$, 30 m
R's size: 9
Intersect: 12

Fig. 6. Example: smart-and-closed algorithm.

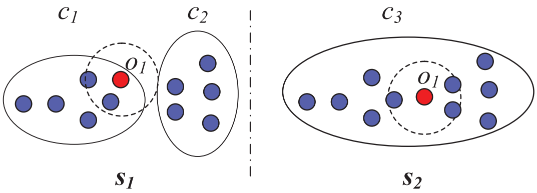


Fig. 7. Example: individual sensitivity problem.

4. TRAVELING-BUDDY-BASED DISCOVERY

The smart-and-closed algorithm improves the efficiency of the intersection step to generate companions, but the system still has to cluster the objects in each snapshot. The density-based clustering costs $O(n^2)$ time without a spatial index, where n is the number of objects [Han and Kamber 2006]. Due to the dynamic nature of streaming trajectories (i.e., the objects' positions are always changing), maintaining traditional spatial indexes (such as R-tree or quad-tree) at each time snapshot incurs high cost [Lee et al. 2003]. In this section, we introduce a new structure, called *traveling buddy*, to maintain the relationship among objects and help discover companions.

4.1. The Traveling Buddy

In streaming trajectories, the objects keep on moving and updating their positions, however, the changes of object relationships are gradual evolutions rather than fierce mutations. The object relationships are possible to be retained in a few snapshots, that is, the objects are likely to stay together with several members of the current cluster. It is attractive to reuse such information to speed up the clustering tasks. However, the system cannot reuse it directly. The major issue is about the intrinsic feature of density-based clustering. Unlike other types of clusters, the results of density-based clustering may be quite different due to minor position changes of an individual object. This phenomenon is called *individual sensitivity* as illustrated in Example 4.

Example 4. Figure 7 shows two consecutive snapshots of the trajectory stream. Suppose the density threshold μ is set to three. In snapshot s_1, two clusters c_1 and c_2 are independent. However in s_2, object o_1 moves a little to the south, and this movement makes the two clusters density connected and merged as one cluster c_3. Such cases may impose important meanings in real applications, for instance, in the scenario of infected disease monitoring, the people in the two clusters should then be watched together since the disease may spread among them.

The time cost of checking individual sensitivity is quadratic to the cluster size, and in many cases the system has to generate large clusters to produce meaningful companions. Hence high computational overhead is still involved in the clustering stage.

Then is it possible to explore a smaller and more flexible structure? In the real world, there are some kinds of microgroups in a trajectory stream. For example, couples would like to stay together on trips, military units operate in teams, families of birds, deer, and other animals often move together in species migration. Such objects stay closer to each other than outside members. Even though they might not be as big as the companion, their information can be used to help clustering. Since they are way smaller than the cluster, their maintenance cost is much lower.

Definition 6 (*Traveling Buddy*). Let O be the object set and δ_γ be the buddy radius threshold, traveling buddy b is defined as a set of objects satisfying: (1) $b \subseteq O$; (2) for $\forall o_i \in b, dist(o_i, cen(b)) \leqslant \delta_\gamma$, where $cen(b)$ is the geometry center of b. The buddy's radius γ is defined as the distance from $cen(b)$ to b's farthest member.

The traveling buddies can be initialized by incrementally merging the objects in two steps: (1) treating all objects as individual buddies; and (2) merging them with their nearest neighbors. This process stops if the buddy's radius is larger than γ. The initialization step costs $O(n^2)$ time for n objects. However, this step only needs to be carried out once and the traveling buddies are dynamically maintained along the stream.

There are two kinds of operations to maintain buddies on the data stream, namely, *split* and *merge*, as shown in the following example.

Example 5. Figure 8 shows the traveling buddies in two snapshots. Traveling buddy b_1 is split into three parts in snapshot s_2. At the same time, b_2, b_3, and a part of b_1 are merged as a new buddy in s_2.

When the data of a new snapshot s_{t+1} arrives, the maintenance algorithm first updates the center of each buddy b. For object $o_i \in b$, the system calculates the shift (Δx_i, Δy_i) between s_{t+1} and s_t. And the new center is updated as

$$cen_{t+1}(b) = cen_t(b) + \sum_{o_i \in b} (\Delta x_i, \Delta y_i).$$

Then every object $o_i \in b$ checks its distance to the buddy center; if the distance is larger than δ_γ, o_i will be split out as a new buddy. The cen(b) is also updated by subtracting the shift of o_i.

The second operation is to merge the buddies that are close to each other. If two buddies b_i and b_j satisfy the following equation, they should be merged as a

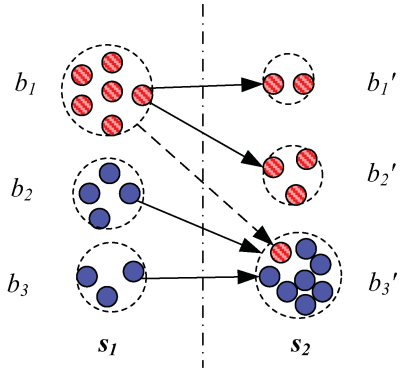


Fig. 8. Example: merge and split buddies.

ALGORITHM 3: Traveling Buddy Maintenance
Input: the radius threshold δ_γ, the traveling buddy set B and the coming snapshot s
Output: updated buddy set B'

```
1.  for each $b_i$ in $B$ do
2.      update cen($b_i$);
3.      for $o_j$ in $b_i$, do
4.          if $dist(o_j, cen(b_i)) > \delta_\gamma$ then // Split Operation
5.              split $o_j$ out as a new buddy $b_j$;
6.              add $b_j$ to $B'$;
7.              update cen($b_i$);
8.      add $b_i$ to $B'$;
9.  //Merge Operation
10. for each $b_i, b_j$ in $B'$, $b_i \neq b_j$ do
11.     if $dist(cen(b_i), cen(b_j)) + \gamma_i + \gamma_j \leq 2\delta_\gamma$ then
12.         merge $b_i$, $b_j$ as $b_k$;
13.         remove $b_i$, $b_j$ and add $b_k$ to $B'$;
14. return $B'$;
```

Fig. 9. Algorithm: buddy maintenance.

new buddy.

$$dist(cen(b_i), cen(b_j)) + \gamma_i + \gamma_j \leqslant 2\delta_\gamma$$

Suppose b_i has m_i objects and b_j has m_j objects, the new buddy b_k's center is computed as $cen(b_k) = (m_i * cen(b_i) + m_j * cen(b_j))/(m_i + m_j)$. Therefore, the system does not need to access the detailed coordinates of each object to merge buddies; the computation can be done with the information from the old buddy's center and size.

The detailed steps of buddy maintenance are shown in Figure 9. When the data of a new snapshot arrives, the algorithm first updates the center of each buddy (line 2). Then each buddy member is checked to see whether a split operation is needed (lines 3–7). At last, the system scans the buddy set and merges the buddies that are close to each other (lines 10–13).

PROPOSITION 2. *Let m be the average number of traveling buddies and n be the number of objects. The time cost of Algorithm 3 is $O(n + m^2)$.*

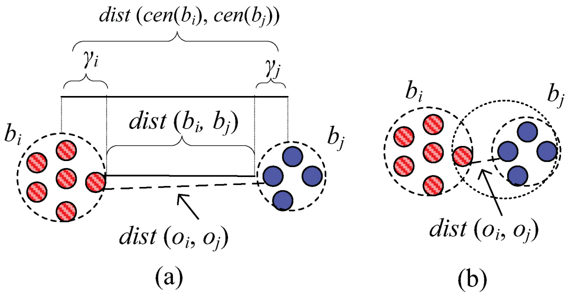


Fig. 10. Proof of Lemmas 3 and 4.

Proof. The split operation needs to check each object and the time cost is $O(n)$. The merge operation has to check the buddies pairs with time complexity $O(m^2)$. Therefore the total maintenance cost is $O(n + m^2)$. □

In the worst case, if the objects are sparse and each of them is an individual buddy, where $m = n$, the maintenance cost is still $O(n^2)$. However, the number of m is usually much smaller than n and the algorithm is likely to strike a relatively high efficiency.

4.2. Buddy-Based Clustering

In the clustering step, the system has to check the density connectivity for each object. The traveling buddies can help the clustering process avoid accessing those object details. To bring down computational overhead, we introduce following lemmas.

Lemma 2. *Let b be a traveling buddy, ε be the distance threshold, and μ be the density threshold. If b's size is larger than $\mu + 1$ and the buddy radius $\gamma \leqslant \varepsilon/2$, then all the objects in b are directly density reachable to each other. Such a traveling buddy is called a* density-connected buddy.

Proof. Note that $\gamma \leqslant \varepsilon/2$, thus for $\forall o_i, o_j \in b$, $dist(o_i, o_j) \leqslant 2\gamma \leqslant \varepsilon$. Then all the members of b are included in $N_\varepsilon(o_i)$. If b's size is larger than $\mu + 1$, then $|N_\varepsilon(o_i)| \geqslant \mu$. By Definition 1, o_i and o_j are directly density reachable. □

Lemma 2 shows that, if a traveling buddy is tight and large by itself, then all its members can be considered as density connected. Lemma 2 also gives the directions that the radius threshold δ_γ should not be set larger than $\varepsilon/2$.

Lemma 3. *Let b_i and b_j be two traveling buddies with radius γ_i and γ_j, and ε be the distance threshold. If $dist(cen(b_i), cen(b_j)) - \gamma_i - \gamma_j > \varepsilon$, then the objects in b_i and b_j are not directly density reachable.*

Proof. As shown in Figure 10(a): if $dist(cen(b_i), cen(b_j)) - \gamma_i - \gamma_j > \varepsilon$, then for $\forall o_i \in b_i, o_j \in b_j$, $dist(o_i, o_j) > \varepsilon$. Therefore, o_j does not belong to $N_\varepsilon(o_i)$ and they are not directly density reachable. □

Lemma 3 tells us that, when searching for the directly density reachable objects for a traveling buddy, if another buddy is too far away, then the system can prune all its members without further computation. This lemma is very helpful. In the experiments it helps prune more than 80% of the objects.

For the traveling buddies that are close to each other, the detailed distance computation still needs to be carried out. But with the following lemmas, the system does not

ALGORITHM 4: Buddy-based Clustering
Input: the distance threshold ε, the density threshold μ, the coming snapshot s and the buddy set B.
Output: the cluster set C.

```
1.  update buddy set $B$; //Algorithm 3
2.  randomly pick a buddy $b$;
3.  initialize cluster $c \leftarrow b$, add $c$ to $C$;
4.  remove $b$ from $B$;
5.  for each unvisited buddy $b_i$ in $c$
6.    mark $b_i$ as visited;
7.    for each buddy $b_j$ in $B$, do
8.      if $dist(cen(b_i), cen(b_j))$ - $\gamma_i$ - $\gamma_j > \varepsilon$, then
9.        continue; // Lemma 3
10.     for each $o_i$ in $b_i$, $o_j$ in $b_j$, do
11.         if $dist(o_i, o_j) \leq \varepsilon$, then
12.           if $b_i$ , $b_j$ are density connected then
13.             add $b_j$ to $c$; //Lemma 4
14.             remove $b_j$ from $B$;
15.             break;
16.           else if $o_j$ is density connected from $o_i$ then
17.             split $b_j$ to objects;
18.             add $o_j$ to $c$;
19. repeat steps 2 - 18 until all buddies are processed;
20. return the cluster set $C$;
```

Fig. 11. Algorithm: buddy-based clustering.

need to compute distances between all the pairs. Lemma 4 provides heuristics to speed up the computation.

LEMMA 4. *Let b_i, b_j be two density-connected buddies and ε be the distance threshold. If $\exists o_i \in b_i, o_j \in b_j$ such that $dist(o_i, o_j) \leqslant \varepsilon$, then all the objects of b_i and b_j are density connected.*

PROOF. As Figure 10(b) shows, since b_i is a density-connected traveling buddy and $|N_\varepsilon(o_i)| \geqslant \mu$, if $dist(o_i, o_j) \leqslant \varepsilon$, then o_i and o_j are directly density reachable. Since all the objects in b_i and b_j are directly density reachable from o_i and o_j, respectively, therefore, all the objects in the two traveling buddies are density connected. □

Based on Lemma 4, once the system finds a pair of objects close to each other, it ends the computation and considers the corresponding buddies density connected. The detailed algorithm is listed in Figure 11. The algorithm first updates the buddy set in a new snapshot (line 1). Then it randomly picks a buddy and initializes it as a new cluster (lines 2–4). For each buddy in the cluster, the algorithm checks its density connectivity to others (lines 5–18). The far-away buddies are filtered out (Lines 8–9). With the help of Lemma 4, the algorithm searches density reachable buddies and objects and adds them to the cluster (lines 10–18). Finally, the algorithm outputs clustering results when all the buddies are processed (line 20).

In the worst case, Algorithm 4 is still with $O(n^2)$ time complexity, where n is the number of objects. But in most cases, Lemmas 3 and 4 can prune the majority of buddies and save time for distance computation. The experiment results show that buddy-based clustering is an order of magnitude faster than the original clustering algorithm.

ALGORITHM 5: Buddy-based Companion Discovery
Input: Size threshold δ_s, duration threshold δ_t, candidate set R, buddy index BI and the trajectory data stream S
Output: every qualified companion q

```
1.  for each coming snapshot $s$ of $S$;
2.      initialize new candidate set $R'$;
3.      buddy based clustering; // Algorithm 4
4.      update $BI$ and corresponding candidates;
5.      for each candidate $r_i$ in $R$, do
6.          if size($r_i$) < $\delta_s$ then break;
7.          for each cluster $c_j$ in $s$, do
8.              $r_i'$ ← buddy-based-intersection($r_i$, $c_j$);
9.              $duration$ ($r_i'$) = $duration$ ($r_i$)+$duration$ ($s$);
10.             remove intersected objects and buddies from $r_i$;
11.             if size($r_i'$) ≥ $\delta_s$ then
12.                 add $r_i'$ to $R'$;
13.                 if $duration$ ($r_i'$) ≥ $\delta_t$ then
14.                     output $r_i'$ as a qualified companion $q$;
15.     for each cluster $c_j$ do
16.         if $c_j$ is closed then add to $R'$;
17.     $R$ ← $R'$;
```

Fig. 12. Algorithm: buddy-based companion discovery.

4.3. Companion Discovery with Buddies

The buddies are not only useful in the clustering step, they are also helpful for the intersection process to generate companions. When intersecting a candidate with a cluster, the system needs to check whether each candidate's objects appear in the cluster or not. The information of traveling buddies can provide a shortcut to this process: If a buddy stays unchanged during the period, and it appears both in the candidate and the cluster, then the system can put all its members into the intersection result without accessing the detailed objects.

To efficiently utilize the buddy information, a buddy index is designed to keep the candidates dynamically updated with the buddies.

Definition 7 (*Buddy Index*). The buddy index is a triple {*BID*, *ObjSet*, *CanIDs*}, where *BID* is the buddy's ID, *ObjSet* is comprised of the object members of the buddy, and *CanIDs* records the IDs of candidates containing the buddy.

As long as the buddy stays unchanged, the candidates only store the *BID* instead of detailed objects. While making intersections, the buddy is treated as a single object. When the buddy changes, the system updates all the candidates in *CanIDs* and replaces *BID* with the corresponding objects in *ObjSet*. The buddy-based companion discovery algorithm is listed in Figure 12.

When a new snapshot arrives, the algorithm performs buddy-based clustering and updates the buddy index (lines 2–4), then selects out the candidates with enough size (lines 5–6). The candidates are intersected with the generated clusters with the help of the buddy index (lines 7–10). The candidate's duration and size are checked again after the intersection, and the qualified ones are output as the companions (lines 11–14). Finally, the closed candidates are added to the memory for further processing (lines 15–17).

Example 6. Figure 13 shows the running process for buddy-based companion discovery. There are four buddies initialized in snapshot s_1. In the candidates, the buddy

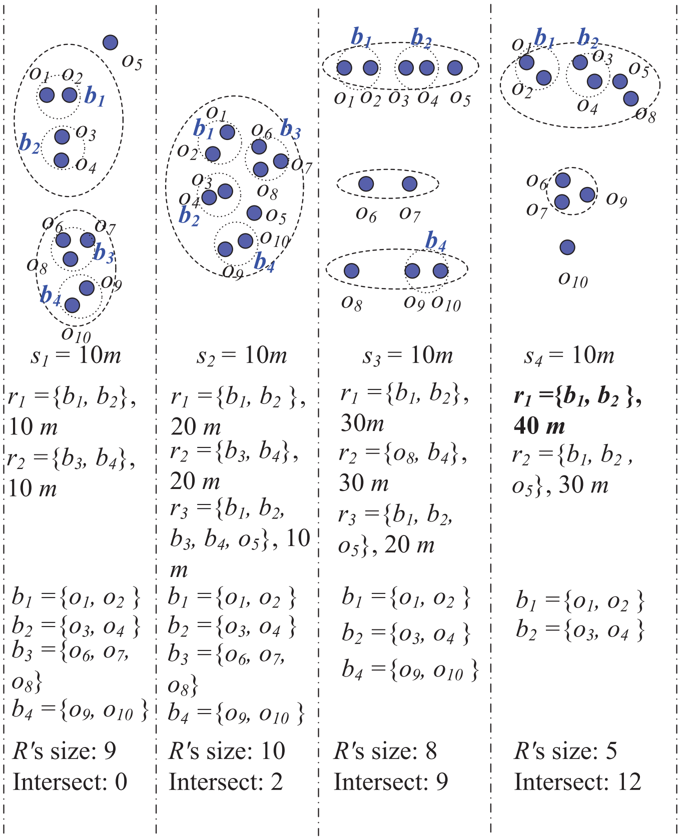

o_5
o_1 o_2 b_1
b_2 o_3 o_4
o_6 o_7 o_8 b_3 o_9 b_4 o_{10}
$s_1 = 10m$
$r_1 = \{b_1, b_2\}$, 10 m
$r_2 = \{b_3, b_4\}$, 10 m
$b_1 = \{o_1, o_2\}$
$b_2 = \{o_3, o_4\}$
$b_3 = \{o_6, o_7, o_8\}$
$b_4 = \{o_9, o_{10}\}$
R's size: 9
Intersect: 0
o_1 b_1 o_2 o_6 b_3 o_7 o_3 o_4 b_2 o_8 o_5 o_{10} o_9 b_4
$s_2 = 10m$
$r_1 = \{b_1, b_2\}$, 20 m
$r_2 = \{b_3, b_4\}$, 20 m
$r_3 = \{b_1, b_2, b_3, b_4, o_5\}$, 10 m
$b_1 = \{o_1, o_2\}$
$b_2 = \{o_3, o_4\}$
$b_3 = \{o_6, o_7, o_8\}$
$b_4 = \{o_9, o_{10}\}$
R's size: 10
Intersect: 2
b_1 b_2
o_1 o_2 o_3 o_4 o_5
o_6 o_7
b_4
o_8 o_9 o_{10}
$s_3 = 10m$
$r_1 = \{b_1, b_2\}$, 30m
$r_2 = \{o_8, b_4\}$, 30 m
$r_3 = \{b_1, b_2, o_5\}$, 20 m
$b_1 = \{o_1, o_2\}$
$b_2 = \{o_3, o_4\}$
$b_4 = \{o_9, o_{10}\}$
R's size: 8
Intersect: 9
o_1 b_1 b_2 o_3 o_5 o_2 o_4 o_8
o_6 o_9 o_7
o_{10}
$s_4 = 10m$
$\boldsymbol{r_1 = \{b_1, b_2\}}$, **40 m**
$r_2 = \{b_1, b_2, o_5\}$, 30 m
$b_1 = \{o_1, o_2\}$
$b_2 = \{o_3, o_4\}$
R's size: 5
Intersect: 12

Fig. 13. Example: buddy-based discovery.

ID is stored instead of detailed objects. In snapshot s_2, the four buddies stay the same and the algorithm makes intersections by only checking their *BID*s. Although the total intersection time is not reduced, the time cost for each intersection operation has been brought down. It is common that different candidates contain the same objects, such as r_1 and r_3 in s_2. The buddy index helps to keep only one copy of the objects and add only pointers (the *BID*s) to candidates. Therefore, the space cost is further reduced. In s_3, the buddy b_3 is no longer valid, then the system updates candidate r_2, using the objects to replace the buddy's ID. In s_4, traveling companion r_1 is discovered as $\{b_1, b_2\}$. With the help of the buddy index, the system can easily look up detailed objects and output the companion as $\{o_1, o_2, o_3, o_4\}$.

5. ROAD COMPANION DISCOVERY

In the previous sections, we have investigated the problem of companion discovery on 2D Euclidean space. However, many objects move on road networks in real applications. There are several unique difficulties for companion discovery on the road network. In this section we explore the problem of discovering road companions.

5.1. Problem Formulation

Example 7. Figure 14 shows the example of moving vehicles on the road network. There are several issues different from the companion discovery in 2D Euclidean space.

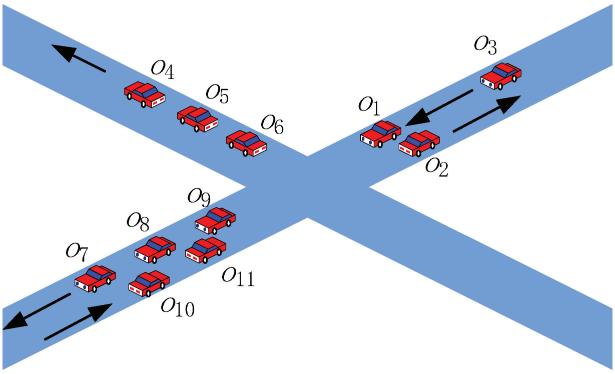


Fig. 14. Example: traveling companions on road network.

—*Distance computation.* In the road network, the distance between two objects should be the length of the shortest path connecting them, rather than a straight line between them. As shown in the figure, o_1 and o_2 are close to each other in the Euclidean space, but they are on different directions. The road network distance between them is actually much larger.

—*Moving direction.* In most cases, the road companion moves in the shape of a line. The moving direction of the object is an important factor in determining the companion. For example, o_7, o_8, and o_9 in Figure 14 have neighboring vehicles o_{10} and o_{11} moving in opposite direction. Such vehicles should not be counted as the companion members. Therefore, traditional density-based clustering should be modified to model the vehicle's moving directions.

Since the road companion discovery is a new type of problem, it is necessary to modify some basic concepts of the traveling companion and redefine the task with new constraints.

Definition 8 (*Direct Road Connection*). Let O be the object set in a snapshot, M be the road network, and ε be the distance threshold. Object o_j is *directly road connected* from object o_i on M if $netd(o_i, o_j) \leq \varepsilon$, where $netd(o_i, o_j)$ is the road network distance between o_i and o_j on M.

Note that we remove the requirements about density and replace the Euclidean distance with the road network distance in Definition 8.

Definition 9 (*Road Connection*). Let O be the object set in a snapshot, M be the road network, object o_i is *road connected* to object o_j on M, if there is a chain of objects $\{o_1, \ldots, o_n\} \in O$ where $o_1 = o_j$, $o_n = o_i$ such that o_{i+1} is directly road connected from o_i on M.

Based on the previous definitions, we can formally define the task of road companion discovery as follows.

Definition 10 (*Road Companion*). Let M be the road network, δ_s be the size threshold, and δ_t be the duration threshold, a group of objects q is called *road companion* if:

(1) the members of q are road connected on M for a period t where $t \geq \delta_t$;
(2) q's size $size(q) \geq \delta_s$.

Problem Definition. Let trajectory data stream S be denoted by a sequence of snapshots $\{s_1, s_2, \ldots, s_i, \ldots\}$. Each snapshot $s_i = \{(o_1, x_{1,i}, y_{1,i}), (o_2, x_{2,i}, y_{2,i}), \ldots, (o_n, x_{n,i}, y_{n,i})\}$,

ALGORITHM 6: Road-connection-based Clustering
Input: the distance threshold ε, the size threshold δ_s, the object set O in a snapshot
Output: the road-connected cluster set C

```
1.  for each unvisited object $o$ of $O$, do
2.      mark $o$ as visited;
3.      initialize a new cluster $c$;
4.      add $o$ to $c$;
5.      for each unexpanded object $o_i$ in $c$, do
6.          mark $o_i$ as expanded;
7.          for each unvisited object $o_j$ of $O$, do
8.              if netd($o_i$, $o_j$) $< \varepsilon$, then
9.                  mark $o_j$ as visited;
10.                 add $o_j$ to $c$;
11.     if size($c$) $\geq \delta_s$ then
12.         add $c$ to $C$;
13. return $C$;
```

Fig. 15. Algorithm: road-connection-based clustering.

where $x_{j,i}$, $y_{j,i}$ are the spatial coordinates of object o_j at snapshot s_i, and all the objects move on a road network M. When the data of snapshot s_i arrives, the task is to discover the road companion set Q that contains all the road companions so far.

Note that we assume the system can match the spatial coordinates of the moving objects to the road network efficiently. There are many state-of-the-art studies on this map-matching problem. In our previous studies, we have developed several methods for map matching; the details can be found in Yuan et al. [2010] and Zheng et al. [2012].

5.2. The Discovery Framework

The general framework of clustering and intersection can be adapted to discover road companions. In each snapshot, the system first generates the road connected clusters and intersects them with the *road companion candidates*. The candidates are gradually refined to be the road companions.

Definition 11 (*Road Companion Candidate*). Let M be the road network, δ_s be the size threshold, and δ_t be the duration threshold, a group of objects q is called *road companion candidate* if:

(1) the members of q are road connected on M for a period t where $t < \delta_t$;
(2) $size(q) \geq \delta_s$.

Similarly, the ideas of the smart-and-closed algorithm also work for this framework. To apply those algorithms on road networks, the only difference is to replace the process of density-based clustering with the following algorithm of road-connection-based clustering.

Algorithm 6 first picks a random object as the seed to initialize a cluster (lines 1–4), then expands the cluster (lines 5–10). In the expansion process, the algorithm starts from the seed, and adds in any objects that are directly road connected to the cluster member (lines 7–10). Once a cluster is generated, the system compares its size with the threshold, and only the ones with enough size are included in the final clustering results (lines 11–13).

PROPOSITION 3. *Let n be the size of object set O and N be the total node number of road network M. The time complexity of Algorithm 6 is $O(n^2 * N)$.*

PROOF. There are three loops in Algorithm 6 (lines 1, 5, and 7). In the worst case, no objects are road connected. Hence the algorithm has to run n times for the loops in lines 1 and 7, and 1 time for the loop in line 5 (each cluster only contains one object in such a case). The total running number is $O(n^2)$. In each run, the system has to find the shortest path between objects o_i and o_j to compute their road network distance. The time cost of the shortest path searching step is determined to the detailed algorithm and heuristics [Pearl 1984]. In the worst case, the algorithm has to visit all the nodes of M to find out the shortest path, hence the time complexity of Algorithm 6 is $O(n^2 * N)$. □

In many applications, the road network M contains millions of nodes, that is, N is a quite large number. To make things worse, the system may not have enough memory to load in M in one time. Therefore the shortest path computation involves huge I/O overhead. The time cost of Algorithm 6 is much larger than the density-based clustering, and it is not feasible for efficient road companion discovery on trajectory streams.

The bottleneck in Algorithm 6 is searching for the directly road connected objects (lines 7–10). For a particular object o_i, the system has to find the shortest paths between o_i and all unvisited objects. This computation process is the most costly step of the algorithm. However, it is actually not necessary to compute all those shortest paths, and the algorithm's time cost can be reduced significantly with the following lemma.

LEMMA 5. *In the road network M, if the Euclidean distance between two objects o_i and o_j is larger than the distance threshold ε, o_i and o_j are not directly road connected.*

PROOF. In the Euclidean space, the shortest path between o_i and o_j is a straight line connection. Since the road network M is also in the same Euclidean space, the Euclidean distance must be less than or equal to the road network distance: $dist(o_i, o_j) \leq netd(o_i, o_j)$. If $dist(o_i, o_j) > \varepsilon$, then $netd(o_i, o_j) > \varepsilon$. According to Definition 8, o_i and o_j are not directly road connected. □

Lemma 5 can help accelerate the road connection clustering process. We develop a new clustering algorithm with the *filtering-and-refinement* strategy, as listed in Figure 16.

The main step of Algorithm 7 is at line 8. Since the main workload of the road-connection-based clustering is on the shortest path computation, Algorithm 7 is designed to reduce such computation and avoid the huge I/O cost of accessing the road network data. When searching for the directly road connected objects for object o_i, the system first computes the Euclidean distance $dist(o_i, o_j)$, the measure whose computation only needs the coordinates of o_i and o_j and involves no I/O cost. If $dist(o_i, o_j)$ is already larger than the threshold ε, according to Lemma 5, o_j is not possible to be road connected with o_j, and the system can filter it without any further computation. In such way, about 80% of the objects are pruned and the algorithm is nearly an order of magnitude faster in our experiments.

PROPOSITION 4. *Let n be the size of object set O, N be the total node number of road network M, and m be the number of objects that pass the filtering process. The time complexity of Algorithm 7 is $O(n^2 + mN)$.*

PROOF. With the filtering-and-refinement strategy, the algorithm only needs to compute road network distances for the m objects which pass the filtering process. Therefore the total time complexity is $O(n^2 + mN)$. □

ALGORITHM 7: Clustering with Filtering-and-refinement
Input: the distance threshold ε, the size threshold δ_s, the object set O in a snapshot
Output: the road-connected cluster set C

```
for each unvisited object $o$ of $O$, do
    mark $o$ as visited;
    initialize a new cluster $c$;
    add $o$ to $c$;
    for each unexpanded object $o_i$ in $c$, do
        mark $o_i$ as expanded;
        for each unvisited object $o_j$ of $O$, do
            if dist($o_i$, $o_j$) > $\varepsilon$, then continue;
            if netd($o_i$, $o_j$) < $\varepsilon$, then
                mark $o_j$ as visited;
                add $o_j$ to $c$;
    if size($c$) ≥ $\delta_s$ then
        add $c$ to $C$;
return $C$;
```

Fig. 16. Algorithm: clustering with filtering-and-refinement.

Note that m is much smaller than n with a reasonable distance threshold ε. And the Euclidean distance computation does not need to access the road network M. The computation time and I/O overhead are reduced dramatically.

5.3. The Road-Buddy-Based Approach

The road-connection-based clustering algorithm also has the problem of individual sensitivity. The similar idea of traveling buddy can be applied to improve the algorithm's efficiency. The *road buddy* is thus proposed to maintain small groups of objects moving together along the roads.

Definition 12 (*Road Buddy*). Let M be the road network, O be the object set, and δ_γ be the buddy radius threshold, the road buddy b is defined as a set of objects satisfying: (1) $b \subseteq O$; (2) for $\forall o_i \in b$, $netd(o_i, netcen(b)) \leq \delta_\gamma$, where $netcen(b)$ is the projection of the geometry center of b on the road network M. The buddy's radius γ is defined as the road network distance from $netcen(b)$ to b's farthest member.

To obtain $netcen(b)$, the system needs to first compute the geometry center of b, then employ a map-matching algorithm to project the geometry center to the nearest road. In this study, we use the map matching algorithm developed in our previous works [Yuan et al. 2010].

The road buddy has the same operations of *split* and *merge* as the traveling buddy. Their initializations are also similar. Their major difference is at the maintenance process. Because it is costly to compute the road network distance from $netcen(b)$ to each member, the maintenance algorithm employs the filtering-and-refinement strategy to reduce time cost, as listed in Figure 17.

When the data of a new snapshot arrives, Algorithm 8 first computes the network center of each buddy (lines 2–3), then checks each road buddy to see whether a split operation is needed (lines 4–11), finally scans the buddy set and merges the ones that are close to each other (lines 12–17). The key steps of filtering-and-refinement are at lines 5, 6, 14, and 15. Before computing the road network distance between two points,

ALGORITHM 8: Road Buddy Maintenance
Input: the road network M, the radius threshold δ_γ, the road buddy set B and the coming snapshot s
Output: updated buddy set B'

```
for each $b_i$ in $B$ do
    update $cen(b_i)$;
    match $cen(b_i)$ to $M$ and compute $netcen(b_i)$;
    for $o_j$ in $b_i$, do
        if $dist(o_j, cen(b_i)) > \delta_\gamma$ then isSplit ← true;
        else if $netd(o_j, cen(b_i)) > \delta_\gamma$ then isSplit ← true;
        if isSplit = true, then  //Split Operation
            split $o_j$ out as a new buddy $b_j$;
            add $b_j$ to $B'$;
            update $netcen(b_i)$;
    add $b_i$ to $B'$;
//Merge Operation
for each $b_i$, $b_j$ in $B'$, $b_i \neq b_j$ do
    if $dist(netcen(b_i), netcen(b_j)) + \gamma_i + \gamma_j \leq 2\delta_\gamma$ then
        if $netd(netcen(b_i), netcen(b_j)) + \gamma_i + \gamma_j \leq 2\delta_\gamma$ then
            merge $b_i$, $b_j$ as $b_k$;
            remove $b_i$, $b_j$ and add $b_k$ to $B'$;
return $B'$;
```

Fig. 17. Algorithm: road buddy maintenance.

the algorithm checks whether their Euclidean distance is passing the threshold and only carries out further computation on the qualified pairs.

The road buddy can be used to improve the efficiency of road-connection-based clustering and companion generation by avoiding accessing the object details. Similar to the traveling buddy, we propose several lemmas that are helpful for road companion discovery.

LEMMA 6. *Let b be a road buddy, ε be the distance threshold. If the buddy radius $\gamma \leq \varepsilon/2$, then all the objects in b are directly road connected to each other. Such a road buddy is called a* road connected buddy.

PROOF. Note that $\gamma \leq \varepsilon/2$, thus for $\forall o_i, o_j \in b$, $netd(o_i, netcen(b)) \leq \gamma$ and $netd(o_j, netcen(b)) \leq \gamma$. Hence there exists a path ζ bypassing $netcen(b)$ that connects o_i and o_j, and $length(\zeta) \leq 2\gamma \leq \varepsilon$. Therefore $netd(o_i, o_j) \leq length(\zeta) \leq \varepsilon$. According to Definition 8, o_i and o_j are directly road connected. □

LEMMA 7. *Let b_i, b_j be two road connected buddies and ε be the distance threshold. If $\exists o_i \in b_i, o_j \in b_j$ such that $netd(o_i, o_j) \leq \varepsilon$, then all the objects of b_i and b_j are network connected.*

PROOF. If $netd(o_i, o_j) \leq \varepsilon$, then o_i and o_j are directly road connected. Since all the objects in b_i and b_j are directly road connected from o_i and o_j, respectively, therefore, all the objects in the two traveling buddies are road connected. □

Lemma 6 and 7 can be used to speed up the road-connection-based clustering. The lemmas show that if two buddies are tight by themselves and close to each other, the system can consider all their members as road connected without further computation.

ALGORITHM 9: Road-Buddy-based Clustering
Input: the distance threshold ε, the coming snapshot s and the buddy set B.
Output: the cluster set C

```
1.  update buddy set $B$; //Algorithm 3
2.  randomly pick a buddy $b$;
3.  initialize cluster $c \leftarrow b$, add $c$ to $C$;
4.  remove $b$ from $B$;
5.  for each unvisited buddy $b_i$ in $c$
6.    mark $b_i$ as visited;
7.    for each buddy $b_j$ in $B$, do
8.      if $dist(netcen(b_i), netcen(b_j))$ - $\gamma_i$ - $\gamma_j > \varepsilon$, then
9.        continue; // Lemma 3
10.     for each $o_i$ in $b_i$, $o_j$ in $b_j$, do
11.       if $netd(o_i, o_j) \leq \varepsilon$, then
12.         if $b_i$ , $b_j$ are road connected then
13.           add $b_j$ to $c$; //Lemma 4
14.           remove $b_j$ from $B$;
15.           break;
16.         else if $o_j$ is road connected from $o_i$ then
17.           split $b_j$ to objects;
18.           add $o_j$ to $c$;
19. repeat steps 2 - 18 until all buddies are processed;
20. return the cluster set $C$;
```

Fig. 18. Algorithm: road-buddy-based clustering.

Lemma 8. *Let b_i and b_j be two road buddies with radius γ_i and γ_j, and ε be the distance threshold. If $dist(netcen(b_i), netcen(b_j)) \geq \gamma_i + \gamma_j + \varepsilon$, then the objects in b_i and b_j are not directly road connected.*

Proof. As Lemma 5 shows, the Euclidean distance is the lower bound of road network distance, $netd(netcen(b_i), netcen(b_j)) \geq dist(netcen(b_i), netcen(b_j)) \geq \gamma_i + \gamma_j + \varepsilon$, then for $\forall o_i \in b_i, o_j \in b_j$, $netdist(o_i, o_j) \geq \varepsilon$. Therefore, o_i and o_j are not directly network connected. □

Lemma 8 is helpful to prune most of the unconnected buddies in road-connection-based clustering. Especially the lemma does not require the system to compute any road network distance on M. The system only needs the network center of buddies and their radius as input (which are already computed), and the huge I/O cost could be saved.

The detailed algorithm is listed in Figure 18. Algorithm 9 first calls Algorithm 8 to update the road buddies with new data (line 1), then randomly picks a road buddy as the seed to form a cluster (lines 2–4). The algorithm searches for the buddies that are road connected and adds them to the cluster (lines 2–18). The buddies that are distant from the seed are filtered out directly without detailed distance computation (lines 8–9). The algorithm searches road connected buddies with Lemmas 6 and 7 (lines 10–18). Finally, the algorithm outputs the clustering results when all road buddies are processed (line 20).

The buddy index can be retrieved from road buddies and help companion generation. Because this technique is actually independent from the metrics and distance computation, Algorithm 5 can be applied directly on road buddies.

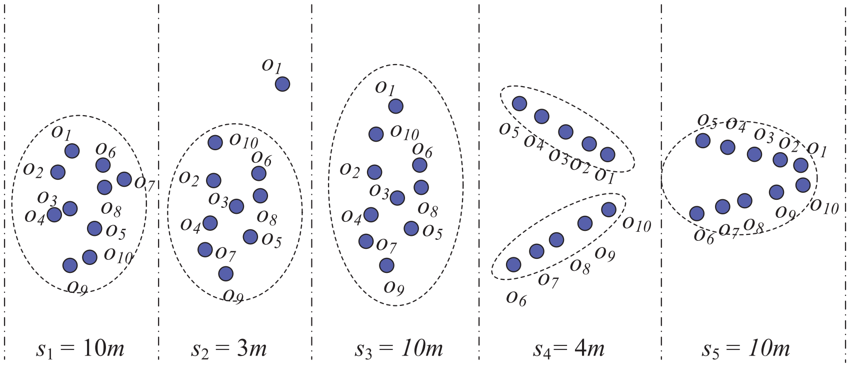

$s_1 = 10m$
$s_2 = 3m$
$s_3 = 10m$
$s_4 = 4m$
$s_5 = 10m$
o_1 o_2 o_3 o_4 o_5 o_6 o_7 o_8 o_9 o_{10}

Fig. 19. Example: movement of military troops.

6. LOOSE COMPANION DISCOVERY

In many applications such as military object monitoring, several members may temporarily leave the group and go back in short time. The companion discovery algorithm will miss such companions if strictly following the time constraints.

Example 8. Figure 19 shows the trajectory streams of a small team of military troops. At snapshot s_1, the team members move together. They send out a member o_1 to scout around at s_2 and o_1 returns to the team at s_3. The team then splits to two parts at s_4 to conduct a "pincer attack" against enemies. Finally they reunite at s_5. Suppose the size threshold is 6 and the duration threshold δ_t is set as 30 minutes. The system cannot discover any companion from the data if strictly following the constraints.

In most cases, the rigid time constraints may lead to no result or not the best results of discovered traveling companion. It is necessary to release the constraints for more effective discovery. To this end, we introduce the concept of *loose companion* as follows.

Definition 13 (*Loose Companion*). Let δ_s be the size threshold, δ_t be the duration threshold, and δ_l be the leaving time threshold, a group of objects q is called *loose companion* if:

(1) let T be the total time that the members of q are density connected, $T \geq \delta_t$;
(2) q's size $size(q) \geq \delta_s$;
(3) for each member o of q, let t be the maximum period that o is not density connected with other members of q, $t \leq \delta_l$.

The loose companion allows the member objects temporarily leaving the companion, as long as the leaving time is less than the threshold δ_l. In Figure 19, if we set δ_l as 5 minutes, the military team could be discovered as a companion.

Similarly we propose the definition of *loose buddy*.

Definition 14 (*Loose Buddy*). Let s be a snapshot of the trajectory stream, δ_γ be the buddy radius threshold, and δ_l be the leaving time threshold, *loose buddy* b is defined as a set of objects, for $\forall o_i \in b$:

(1) $dist(o_i, cen(b)) \leq \delta_\gamma$, where $cen(b)$ is the geometry center of b;
(2) $dist(o_i, cen(b)) > \delta_\gamma$, but the total time of $dist(o_i, cen(b)) > \delta_\gamma$ is less than δ_l.

To discover the loose companions and maintain the loose buddies, the system can follow the same frameworks proposed in previous sections. Only minor modifications need to be carried out in the intersection and split operations. When an object leaves

Dataset	Obj. #	Duration	Sample Freq.	Snapshot#	Record#
Taxi (D_1)	500	4.2 hours	5 minutes	50	25,000
Military (D_2)	780	3 hours	1 minute	180	140,400
Syn 1 (D_3)	1,000	24 hours	1 minute	1,440	1.44 M
Syn 2 (D_4)	10,000	24 hours	1 minute	1,440	14.4 M
The companion size threshold δ_s: 5 – 40, default 10					
The companion duration threshold δ_t: 3 – 15, default 10					
The clustering parameter ε and μ are set according to different datasets.					
The buddy radius threshold δ_γ: $\varepsilon/2$– $\varepsilon/10$, default $\varepsilon/2$.					
The leaving time threshold δ_l: 0 – 6, default 0					

Fig. 20. Experiment settings.

the companion candidate or buddy, the system does not remove that object or split the buddy immediately, instead puts the object/buddy in a buffer to be removed/split after a time period of δ_l. If the object returns in δ_l, the remove/split command will be canceled. Such modification does not influence the general frameworks of companion discovery. The other steps of the clustering-and-intersection algorithm, smart-and-closed method, and the buddy-based approach remain the same for loose companion discovery, hence we omit the details here due to space limitation.

7. PERFORMANCE EVALUATION

7.1. Experiment Setup

Datasets. We evaluate the proposed methods on both real and synthetic trajectory datasets. The taxi dataset (D_1) is retrieved from the Microsoft GeoLife and T-Drive projects [Yuan et al. 2010; Zheng et al. 2010] with the road network of Beijing. The trajectories are generated from GPS devices installed on 500 taxis in the city of Beijing. The dataset is available to the public[5]. The military trajectory dataset (D_2) is retrieved from the CBMANET project [Krout 2007], in which an infantry battalion of 780 units, divided as 30 teams, moves from Fort Dix to Lakehurst for a mission on two routes in 3 hours. Meanwhile, to test the algorithm's performance in large datasets, we also generate two synthetic datasets (D_3 and D_4), being comprised of 1,000 to 10,000 objects, with more than 10 million data records.

Baselines. The proposed Smart-and-Closed algorithm (SC) and Buddy-based discovery algorithm (BU) are compared with Clustering-and-Intersection method (CI), which is used as the framework to find convoy patterns [Jeung et al. 2008]; and two state-of-the-art algorithms: (1) The Swarm pattern (SW) [Li et al. 2010] that captures the objects moving within arbitrary shape of clusters for certain snapshots that are possibly nonconsecutive; (2) the TraClu algorithm (TC) [Lee et al. 2007] that discovers the common subtrajectories with a density-based line-segment clustering algorithm.

Environments. The experiments are conducted on a PC with Intel 6400 Dual CPU 2.13 GHz and 2.00GB RAM. The operating system is Windows 7 Enterprise. All the algorithms are implemented in Java on the Eclipse 3.3.1 platform with JDK 1.6.0. The parameter settings are listed in Figure 20.

7.2. Comparisons in Discovery Efficiency

In this section we conduct experiments to evaluate the efficiency of companion discovery algorithms in Euclidean space. Since both SW and TC cannot output the results incrementally, we take the running time of the entire dataset as the measure for time

[5] GeoLife GPS Trajectories Datasets. Released at: http://research.microsoft.com/en-us/downloads/b16d359d-d164-469e-9fd4-daa38f2b2e13/default.aspx.

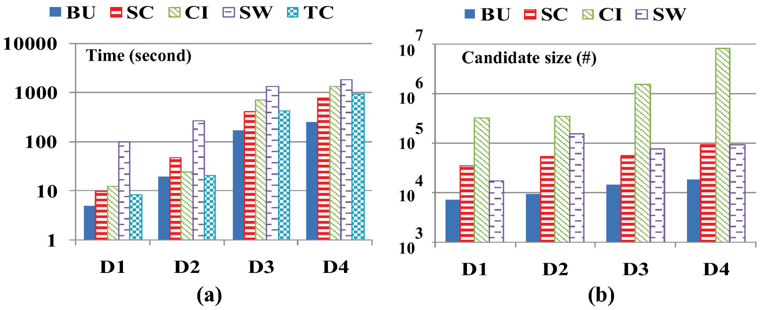


Fig. 21. Efficiency: (a) time, (b) space on different datasets.

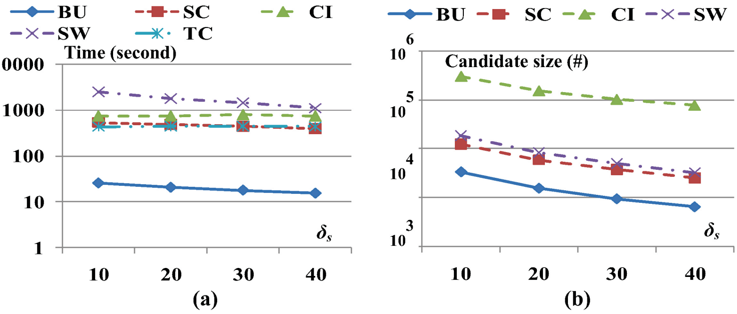


Fig. 22. Efficiency: (a) time, (b) space versus δ_s.

cost. The size of candidate set (number of objects) is used to measure the space cost of companion computation. The only exception is TC, where since the algorithm only carries out the subtrajectory clustering task and does not store any companion candidates, TC's space cost is not included in the experiment.

We first evaluate the algorithm's time and space costs on different datasets with default settings. Figure 21 shows the experiment results. Note that the y-axes are in logarithmic scale. BU achieves the best performances on all the datasets. In the largest dataset D_4, BU is an order of magnitude faster than CI and SW. BU's space cost is only 20% of SW and less than 5% of CI.

Figure 22 illustrates the influences of companion size threshold δ_s in the experiments. The experiment is carried on dataset D_3. Based on default settings, we evaluate the algorithms with different values of δ_s. Generally speaking, when the size threshold grows larger, the filtering mechanism is more effective to prune more companion candidates in each snapshot. The space costs reduce significantly, and the running times also decrease for fewer intersections.

We also study the influence of duration threshold δ_t. Based on default settings, the experiments are conducted on dataset D_3. The value of δ_t is changed from 3 to 15, and the algorithm's performances are shown in Figure 23. BU, SC, and CI are all faster when δ_t grows larger, because many companion candidates are not consistent enough to last for a long time. When setting δ_t as 15 snapshots, BU can process the dataset in less than 20 seconds (Figure 23(a)). It is almost an order of magnitude faster than SC and CI. TC is not influenced by δ_s and δ_t, since it is only a clustering algorithm and does not generate any companion candidates. Beside TC, SW also could not improve the performance when δ_t increases. The reason is SW utilizes the *object growth* strategy to prune candidates. Such heuristics could only work with the size threshold δ_s, but cannot benefit from larger δ_t.

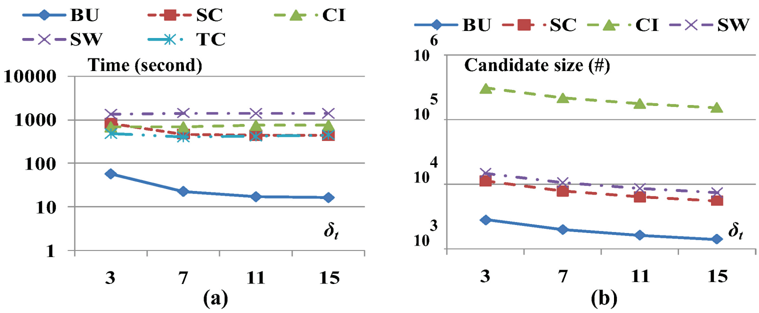


Fig. 23. Efficiency: (a) time, (b) space versus δ_t.

In summary, δ_s and δ_t are two important factors that influence the efficiency of companion discovery algorithms. When increasing the threshold, more companion candidates are pruned and the time and space costs are reduced. BU outperforms other methods in the efficiency evaluations, especially in the scenarios of long-lasting streams with a large number of objects.

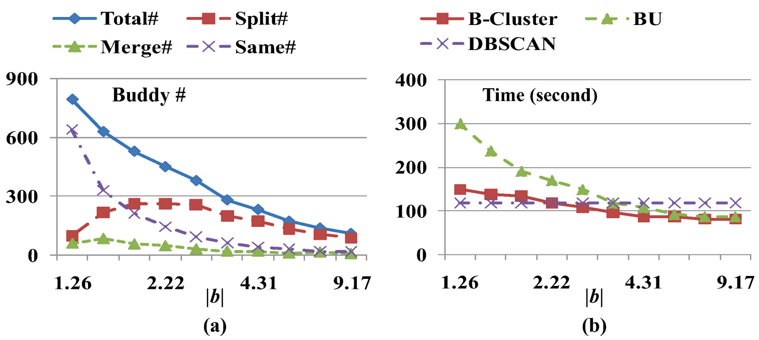
7.3. Efficiency Analysis for Buddy-Based Discovery

Why is the buddy-based discovery algorithm more efficient? In this section we carry out the experimental analysis to reveal the advantages of the buddy-based discovery method.

In the beginning, we tune the parameters of BU to study the factors that influence its efficiency. With δ_s and δ_t set as default values, we test BU with different buddy radius threshold δ_γ from $\varepsilon/10$ to $\varepsilon/2$, and record the average buddy size $|b|$, buddy number, and algorithm's running time. Their relationships are demonstrated in Figure 24. One can clearly learn from Figure 24(a) the total buddy number is inversely proportional to the average buddy size $|b|$. In addition, the number of unchanged buddies decreases rapidly as $|b|$ grows larger. However, as shown in Figure 24(b), the running time of both buddy-based clustering (B-Cluster) and BU decreases with larger $|b|$. This phenomenon can be explained by Proposition 2, where the cost of buddy's maintenance algorithm is $O(n+m^2)$, where n is the number of objects and m is the number of buddies. If n is fixed, then m is inversely proportional to $|b|$. Hence BU costs less time if $|b|$ is larger. Based on the efficiency analysis, we recommend setting the buddy radius as a relatively large value (such as $\varepsilon/2$). Figure 24(b) also records the time cost of the DBSCAN clustering algorithm as a reference. Even if less than 20% buddies stay unchanged (which is rare for real-world objects), as long as the average size of the buddies is larger than 3, the buddy-based clustering algorithm can still outperform DBSCAN. The experiment results show that BU is especially feasible for processing a trajectory stream with dense object clusters.

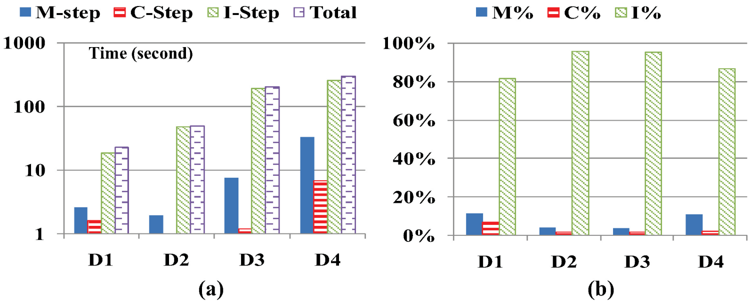
BU has three steps, namely the maintenance step (M-step, Algorithm 3), clustering step (C-step, Algorithm 4), and intersection step (I-step, Algorithm 5). To study the time cost of each step, the system carries out BU on the four datasets and records the time costs of each step, as well as their proportions in the total running time, as shown in Figure 25. The results denote that the clustering step is actually the most efficient in the three, costing less than 5% of the total running time, compared to the *DBSCAN* clustering which usually takes 40–50% of the total running time of SC. BU spends an extra 10%–15% time in maintaining the buddies to save more time from the clustering task.

From the aforesaid experiments, one can clearly see the two key advantages of BU: (1) utilizing the buddy information to filter out most objects without accessing their



(a) (b)

Fig. 24. Efficiency analysis: (a) buddy number; (b) time versus buddy size.



(a) (b)

Fig. 25. Efficiency analysis: (a) running time; (b) percentage of BU steps on diffeferent datasets.

details; (2) employing the buddy index to reduce the size of the candidate set, and so decrease the intersection times of companion discovery.

7.4. Evaluations on Algorithm's Effectiveness

The third part of the experiment is to evaluate the quality of the retrieved companions. In dataset D_2, an infantry battalion of 780 units moves from Fort Dix to Lakehurst for a mission on two routes in 3 hours. The objects are organized in 30 teams, with each team having 25 to 30 units. The information of team partitioning is retrieved as the ground truth. The algorithm's outputs are matched to the ground truth and the measures of precision and recall are calculated as follows.

Precision. The proportion of true companions over all the retrieved results of the algorithm is the precision. It represents the algorithm's selectivity in finding out meaningful companions.

Recall. The proportion of detected true companions over the ground truth is the precision. This criteria shows the algorithm's sensitivity for detecting traveling companions.

We conduct experiments with different values of the size threshold δ_s. The results of effectiveness evaluation are shown in Figure 26. BU and SC have the same precision and recall since they output identical companions. They have about 20% precision improvement over SW, and near 40% precision improvement over CI. SW generates the swarm patterns of frequently meeting objects, which is actually a superset of the companions. The swarm pattern is highly sensitive to helping find out all the companions (i.e., 100% recall), but SW also generates more false positives that bring down the algorithm's selectivity. CI has the same problem with even lower precision. Since there are many redundant and nonclosed companions in the results, more than half of CI's results are not useful.

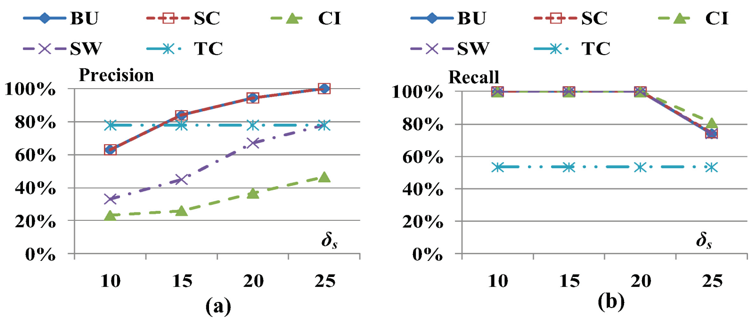


Fig. 26. Effectiveness: (a) precision, (b) recall versus δ_s.

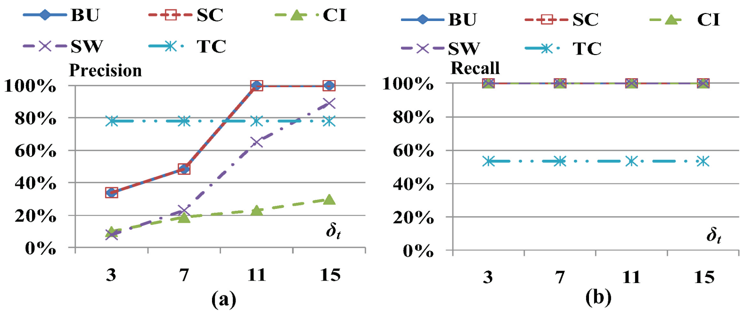


Fig. 27. Effectiveness: (a) precision, (b) recall versus δ_t.

Again, TC is not affected by the parameters of δ_s and δ_t. TC takes the movement direction as an important measure to compute subtrajectory clusters; its results reflect the major directions of the object movements. However, such clusters may not capture the information of companions, because the companion member's moving direction might be different. As an illustration, please go back to Figure 1. From snapshot s_2 to s_3, the moving directions of o_8 and o_9 are different, hence they may be put in different subtrajectory clusters.

Another interesting observation is that, in Figure 26, BU, SC, CI, and SW's precisions all increase when δ_s becomes larger, since fewer companions can pass a higher size threshold. However, if δ_s is set too high (more than 25), several true companions will also be filtered out and the algorithm cannot achieve 100% recall.

In the next experiment, we study the influence of time threshold δ_t. Figure 27 shows the precision and recall of the five algorithms with different δ_t on D_2. BU and SC achieve better performance than SW and CI. When increasing δ_t, the algorithm's precision increases, but they can still keep a high recall. Since all the true companions last for a long period in D_2, if we set δ_t greater than 11, both BU and SC can achieve 100% precision and recall. However, if δ_t is set too high, for example, 15, no companion can be discovered since there exist no object groups moving together for such a long time.

In general, BU and SC can guarantee 100% recall (i.e., not missing any real companion), we suggest that in real applications, the user should set a relatively high time threshold to filter out false positives, but a moderate size threshold to guarantee the algorithm's sensitivity.

7.5. Experiments on Road Companion Discovery

To test the efficiency of road companion discovery, we perform the evaluation on dataset D_1 with the road network of Beijing, which has 106,579 road nodes and 141,380

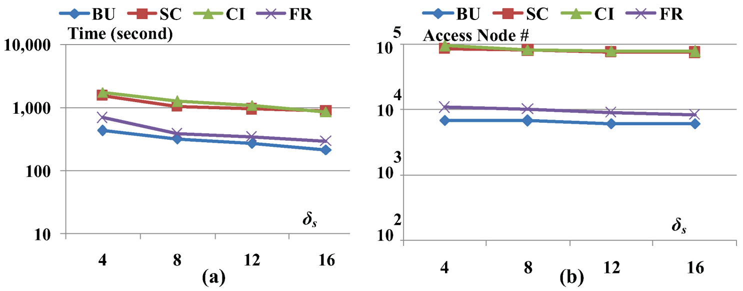


Fig. 28. Efficiency: (a) time; (b) I/O of road companion discovery versus δ_s.

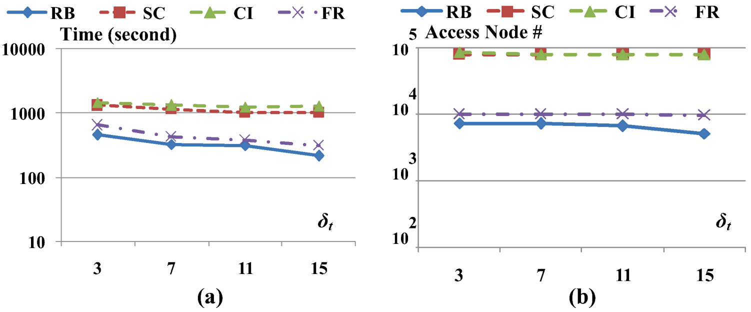


Fig. 29. Efficiency: (a) time; (b) I/O of road companion discovery versus δ_t.

road segments. The default size threshold δ_s is set as 8 and the time threshold δ_t is set as 11. In this experiment, we compare the performance of four methods: (1) the Clustering-and-Intersection framework with road network distance computation (CI); (2) the Smart-and-Closed algorithm with road network distance computation (SC); (3) the smart-and-closed algorithm with Filtering-and-Refinement strategy (FR); and (4) The Road-Buddy-based method (RB).

We first evaluate the time and space costs of road companion discovery. The number of accessed road nodes is used as the measure for I/O cost. Based on default settings, we evaluate the algorithms with different values of δ_s. Figure 28 shows the running time and accessed node number. Generally speaking, when the size threshold grows larger, both running time and I/O costs decrease. The computation cost of road companion discovery is much larger than the traveling companion discovery on Euclidean space. This is mainly caused by the high I/O overhead in road network distance computation. Since the road network distance computation becomes the major cost, SC cannot save much time comparing to CI. However, FR and RB are an order of magnitude faster than SC and CI, because they utilize the filtering-and-refinement strategy to avoid most unnecessary road network distance computations. The effects of RB are better, since RB groups the objects in small buddies and limits the distance computation in a small region with lower I/O overhead.

The influence of duration threshold δ_t is also studied in our experiment. Based on default settings, the value of δ_t is changed from 3 to 15; the algorithms' performances are shown in Figure 29. All the algorithms run faster when δ_t grows larger, because fewer road companion candidates can last for a long time. Again, RB and FR only cost 20%–50% time as CI and SC.

The experiment results show that the main bottleneck of road companion discovery is at the distance computation stage. The traditional companion discovery methods,

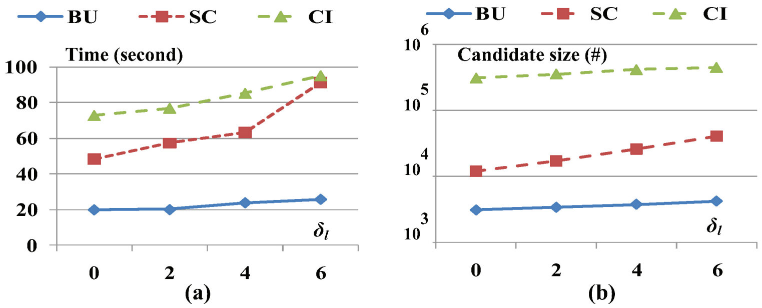


Fig. 30. Efficiency: (a) time; (b) space versus δ_l.

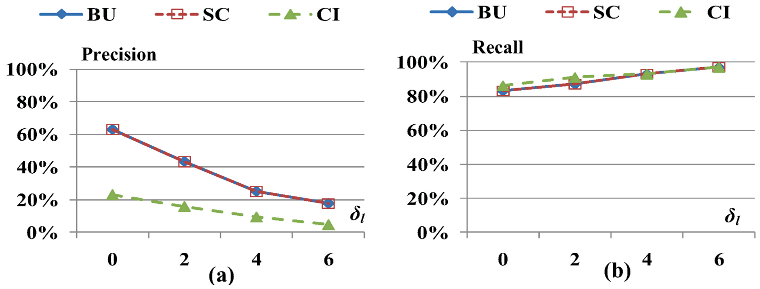


Fig. 31. Effectiveness: (a) precision; (b) recall versus δ_l.

BU and SC, do not work well on the road networks. The new frameworks of RB and FR reduce the time cost on unnecessary shortest path computation, therefore they can achieve higher efficiency peformances.

7.6. Evaluations on Loose Companion Discovery

In the previous experiments, we set the leaving threshold δ_l as 0. In this section, we conduct experiments on loose companion discovery. We run the algorithms of BU, SC, and CI on dataset D_3 by tuning δ_l from 0–6 snapshots.

Figure 30 shows the algorithms' time and space costs. With larger δ_l, all the algorithms' space costs increase rapidly since they cannot prune the candidates if several objects temporarily leave the companion, hence the system has to spend more time in making intersections with a larger candidate set. However, even with large δ_l, BU still can discover the loose companions in about 20 seconds.

Finally we carry out the effectiveness experiment on the military dataset D_2. δ_l is changed from 0 to 6 snapshots, and other parameters are set as the default values. As shown in Figure 31(a), the precision of companion discovery decreases with larger δ_l, since more companions are generated and inevitably the number of false positives increases. However, the good news is that the recall increases as δ_l grows (Figure 31(b)).

The experiment results show the necessity of loose companion discovery. With a released time constraint, BU and SC can discover more meaningful companions and achieve a higher recall. The system's feasibility is increased in real applications.

8. RELATED WORK

According to the methodologies, the related works of traveling companion discovery can be loosely classified into two categories: trajectory clustering and movement pattern discovery.

8.1. Trajectory Clustering

The works in this category focus on developing efficient algorithms to cluster moving objects. Gaffney and Smyth first proposed the fundamental principles of clustering moving objects based on the theories of probabilistic modeling [Gaffney and Smyth 1999; Cadez et al. 2000]. Many distance functions, such as DTW [Yi et al. 1998] and LCSS [Gunopoulos 2002] are proposed. Lee et al. proposed a novel partition-and-group framework to find the clusters based on subtrajectories [Lee et al. 2007].

In Har-Peled [2003], Har-Peled shows that the moving objects can be clustered when the resulting clusters are competitive at any time during the motion. Yang et al. proposed the idea of a neighbor-based pattern detection method for windows [Yang et al. 2009]. Ester et al. made the progress to generate incremental clusters [Ester et al. 1998]. Li et al. propose a microcluster [Li et al. 2004] based schema to cluster moving objects. Zhang and Lin use the k-center clustering algorithm [Gonzalez 1985] for histogram construction. A distance function combining velocity and position differences is proposed in their work [Zhang and Lin 2004]. More recently, Jensen et al. utilize the velocity features to cluster objects for the current and near future positions [Jensen et al. 2007].

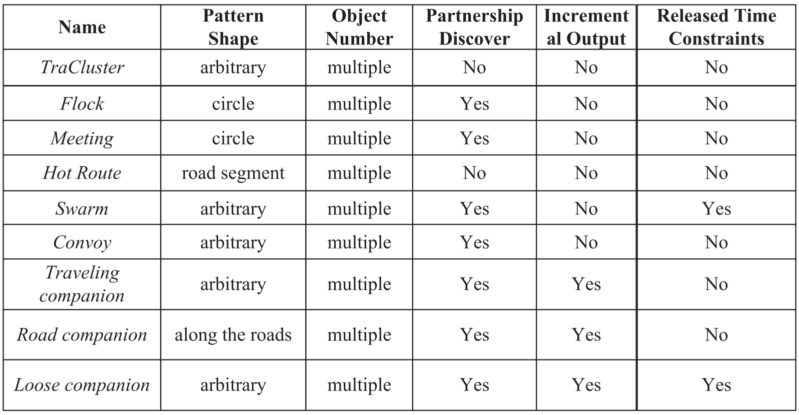
However, as pointed out in Jeung et al. [2008], most of the aforsaid methods cannot be used directly for traveling companion discovery. The major problem is that those algorithms tend to generate clusters for the entire trajectory dataset, instead of each snapshot. Hence the detailed object relationships and evolving companion patterns are all lost. In addition, some algorithms require the object's velocity in advance and need to scan the data for multiple times. Such requirements are not fit for trajectory streams.

8.2. Movement Pattern Discovery

Movement pattern discovery is a hot topic in recent years. The problem has been variously referred to as the search for *flocks* [Gudmundsson and Kreveld 2006], *moving clusters* [Kalnis et al. 2005], *spatial-tempo joins* [Bakalov et al. 2005], *spatial colocations* [Yoo and Shekhar 2004], *meetings* [Gudmundsson et al. 2004], *convoys* [Jeung et al. 2008], *moving groups* [Aung 2008], *swarms* [Li et al. 2010] and so on.

One of the earliest works is *flock* discovery [Gudmundsson et al. 2004]. A flock is defined as a group of objects moving together within a circular region [Gudmundsson and Kreveld 2006]. There are several variations of this model: *Variable flock* permits the members to change during the time span [Benkert et al. 2008],while *meeting* is a circle similar to flock but fixed in a single location all the time [Gudmundsson and Kreveld 2006]. However, such shapes are restricted to circles and the results are also sensitive to the parameter of radius.

Li et al. designed a flow scan algorithm for hot route mining [Li et al. 2007]. Liu et al. mined frequent trajectory patterns by using RF tag arrays. Their work successfully demonstrated the feasibility and the effectiveness of movement patterns in real life [Liu et al. 2007]. Tao et al. proposed the technique of spatio-temporal aggregation using a sketch index. This method can process the queries an order of magnitude faster than the previous works [Tao et al. 2004]. Giannotti et al. proposed the *interest-region*-based mining algorithm [Giannotti et al. 2007]. Horvitz et al. propose the models of using groups of mobile users to discover congestions in urban areas [Horvitz et al. 2005]. The shortest path problem has been studied on land surface [Xing and Shahabi 2010; Liu and Wong 2011] and this technique has been used to process the k-NN queries [Shahabi et al. 2008; Xing et al. 2009]. Tao et al. propose the techniques to find k-skip shortest paths [Tao et al. 2011]. Yuan et al. present a cloud-based system computing customized and practically fast driving routes for an end-user using traffic conditions and driver behavior, which is a milestone study in this field [Yuan et al. 2011].



Name	Pattern Shape	Object Number	Partnership Discover	Increment al Output	Released Time Constraints
TraCluster	arbitrary	multiple	No	No	No
Flock	circle	multiple	Yes	No	No
Meeting	circle	multiple	Yes	No	No
Hot Route	road segment	multiple	No	No	No
Swarm	arbitrary	multiple	Yes	No	Yes
Convoy	arbitrary	multiple	Yes	No	No
Traveling companion	arbitrary	multiple	Yes	Yes	No
Road companion	along the roads	multiple	Yes	Yes	No
Loose companion	arbitrary	multiple	Yes	Yes	Yes

Fig. 32. The comparison with related works.

Zhang et al. propose the techniques to produce intersections of streaming moving objects [Zhang et al. 2008, 2011]. This method is a big improvement from existing algorithms by the speedup of several orders of magnitude. Nutanong et al. use a safe region to report objects that do not change over time [Nutanong et al. 2008, 2010]. The proposed V*-Diagram has much smaller I/O and computation costs than previous methods. It outperforms the best existing technique by two orders of magnitude.

However, since the preceding methods focus more on discovering hot spots, regions, or routes rather than object groups, they cannot be used directly for companion discovery.

Kalnis et al. proposed the first study to automatic extraction of *moving clusters* from large spatial datasets [Kalnis et al. 2005]. In a recent work, Jeung et al. proposed the framework of *convoy* query [Jeung et al. 2008]. It is a significant step forward in the works of movement pattern mining, since it allows the objects to organize in arbitrary shapes. Li et al. further released the constraints of convoy and proposed the *swarm pattern* to discover object groups in a sporadic way [Li et al. 2010].

The concepts of convoy and swarm patterns are similar to traveling companion. However, the convoy mining algorithm needs to scan the entire trajectory into memory to make trajectory simplification, and the system also needs to load the whole dataset into memory to search for swarms. It is impractical to use such a method in a data stream environment. The swarm pattern is a frequent itemset-based concept. Since it is difficult to detect large size frequent itemsets [Zhu et al. 2007], the swarm pattern has limited applicability for datasets with large-scale objects. The major advantage of the companion discovery technique is about the discovery efficiency. The buddy-based method can discover the companions of arbitrary shapes an order of magnitude faster. Hence it is a feasible method to be applied in the data stream scenarios of huge amount of trajectories.

Figure 32 compares the features of some related methods with the proposed algorithms to discovery of traveling companions, road companions, and loose companions.

9. CONCLUSION AND FUTURE WORK

In this study we investigate the problem of traveling companion discovery on trajectory data streams. We propose the algorithms of smart-and-closed discovery to efficiently generate companions from trajectory data. The model of traveling buddy is proposed to help improve both the clustering and intersection processes for companion discovery. The proposed methods are extended to more complex scenarios for road companion and loose companion discovery. We evaluate the proposed algorithms in extensive experiments on both real and synthetic datasets. The buddy-based method is shown to be an order of magnitude faster than existing approaches on both Euclidean space and road networks. The effectiveness of the buddy-based algorithm also outperforms other competitors in terms of precision and recall.

In the future, we are going to integrate the companion discovery methods to real application services such as battlefield monitoring systems and traffic analysis services.

REFERENCES

AUNG, H.-H. 2008. Discovering moving groups of tagged objects. Tech. rep., National University of Singapore. http://www.nus.edu.sg/.

BAKALOV, P., HADJIELEFTHERIOU, M., AND TSOTRAS, V. J. 2005. Time relaxed spatiotemporal trajectory joins. *In Proceedings of the 13th Annual ACM International Workshop on Geographic Information Systems (GIS'05).* 182–191.

BENKERT, M., GUDDMUNDSSON, J., HUBNER, F., AND WOLLE, T. 2008. Reporting flock patterns. *Comput. Geom. Theory Appl. 41,* 3, 111–125.

CADEZ, I. V., GAFFNEY, S., AND SMYTH, P. 2000. A general probabilistic framework for clustering individuals and objects. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00).* 140–149.

ESTER, M., KRIEGEL, H.-P., SANDER, J.,WIMMER, M., AND XU, X. 1998. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB'98).* 323–333.

ESTER, M., KRIEGEL, H.-P., SANDER, J., AND XU, X. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96).* 226–231.

GAFFNEY, S. AND SMYTH, P. 1999. Trajectory clustering with mixtures of regression models. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99).* 63–72

GIANNOTTI, F., NANNI, M., PEDRESCHI, D., AND PINELLI, F. 2007. Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07).* 330–339.

GONZALEZ, T. 1985. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci. 38,* 293–306.

GUDMUNDSSON, J. AND KREVELD, M. V. 2006. Computing longest duration flocks in trajectory data. In *Proceedings of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems (GIS'06).* 35–42.

GUDMUNDSSON, J., KREVELD, M. V., AND SPECKMANN, B. 2004. Efficient detection of motion patterns in spatiotemporal data sets. In *Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems (GIS'04).* 250–257.

GUNOPOULOS, D. 2002. Discovering similar multidimensional trajectories. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02).* 673–684.

HAN, J. AND KAMBER, M. 2006. *Data Mining: Concepts and Techniques* 2nd Ed. Morgan Kaufmann.

HAR-PELED, S. 2003. Clustering motion. *Discr. Comput. Geom. 31,* 4, 545–565.

HORVITZ, E., APACIBLE, J., SARIN, R., AND LIAO, L. 2005. Prediction, expectation, and surprise: Methods, designs, and study of a deployed traffic forecasting service. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI'05).*

JENSEN, C. S., LIN, D., AND OOI, B. C. 2007. Continuous clustering of moving objects. *IEEE Trans. Knowl. Data Engin. 19,* 9, 1161–1174.

JEUNG, H., YIU, M. L., ZHOU, X., JENSEN, C. S., AND SHEN, H. T. 2008. Discovery of convoys in trajectory databases. *Proc. VLDB Endow. 1,* 1, 1068–1080.

KALNIS, P., MAMOULIS, N., AND BAKIRAS, S. 2005. On discovering moving clusters in spatial-temporal data. In *Proceedings of the 9th International Conference on Advances in Spatial and Temporal Databases (SSTD'05).* 364–381.

KROUT, T. 2007. Cb manet scenario data distribution. Tech. rep., BBN.

LEE, J.-G., HAN, J., AND WHANG, K.-Y. 2007. Trajectory clustering: A partition-and-group framework. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'07).* 593–604.

LEE, M., HSU, W., JENSEN, C. S., CUI, B., AND TEO, K. 2003. Supporting frequent updates in r-trees: A bottom-up approach. *The VLDB J. 18,* 3, 719–738.

LI, X., HAN, J., LEE, J.-G., AND GONZALEZ, H. 2007. Traffic density based discovery of hot routes in road networks. In *Proceedings of the 10th International Conference on Advances in Spatial and Temporal Databases (SSTD'07).* 441–459.

LI, Y., HAN, J., AND YANG, J. 2004. Clustering moving objects. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04)*. 617–622.

LI, Z., DING, B., HAN, J., AND KAYS, R. 2010. Swarm: Mining relaxed temporal moving object clusters accurate discovery of valid convoys from moving object trajectories. *Proc. VLDB Endow. 3,* 723–734.

LIU, L. AND WONG, R. C.-W. 2011. Finding shortest path on land surface. In *Proceedings of the ACM SIGMOD International Conference on Management of data (SIGMOD'11).* 433–444.

LIU, Y., CHEN, L., PEI, J., CHEN, Q., AND ZHAO, Y. 2007. Mining frequent trajectory patterns for activity monitoring using radio frequency tag arrays. In *Proceedings of the 5th IEEE International Conference on Pervasive Computing and Communications (PerCom'07).*

NUTANONG, S., ZHANG, R., TANIN, E., AND KULIK, L. 2008. The v*-diagram: A query dependent approach to moving knn queries. *Proc. VLDB Endow. 1,* 1, 1095–1106.

NUTANONG, S., ZHANG, R., TANIN, E., AND KULIK, L. 2010. Analysis and evaluation of v*-knn: An efficient algorithm for moving knn queries. *VLDB J. 19,* 3, 307–332.

PEARL, J. 1984. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley Longman Publishing Co.

SHAHABI, C., TANG, L. A., AND XING, S. 2008. Indexing land surface for efficient knn query. *Proc. VLDB Endow. 1,* 1, 1020–1031.

TANG, L.-A., YU, X., KIM, S., HAN, J., HUNG, C.-C., AND PENG, W.-C. 2010. Tru-alarm: Trustworthiness analysis of sensor networks in cyber-physical systems. In *Proceedings of the IEEE International Conference on Data Mining (ICDM'10).* 1079–1084.

TANG, L.-A., ZHENG, Y., XIE, X., YUAN, J., YU, X., AND HAN, J. 2011. Retrieving k-nearest neighboring trajectories by a set of point locations. In *Proceedings of the 12th International Conference on Advances in Spatial and Temporal Databases (SSTD'11).* 223–241.

TANG, L.-A., ZHENG, Y., YUAN, J., HAN, J., LEUNG, A., HUNG, C.-C., AND PENG, W.-C. 2012. On discovery of traveling companions from streaming trajectories. In *Proceedings of the 28th IEEE International Conference on Data Engineering (ICDE'12).* 186–197.

TAO, Y., KOLLIOS, G., CONSIDINE, J., LI, F., AND PAPADIAS, D. 2004. Spatio-temporal aggregation using sketches. In *Proceedings of the 20th IEEE International Conference on Data Engineering (ICDE'04).* 214.

TAO, Y., SHENG, C., AND PEI, J. 2011. On k-skip shortest paths. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'11).* 421–432.

XING, S. AND SHAHABI, C. 2010. Scalable shortest paths browsing on land surface. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'10).* 89–98.

XING, S., SHAHABI, C., AND PAN, B. 2009. Continuous monitoring of nearest neighbors on land surface. *Proc. VLDB Endow. 2,* 1, 1114–1125.

YANG, D., RUNDENSTEINER, E. A., AND WARD, M. O. 2009. Neighbor-based pattern detection for windows over streaming data. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'09).* 529–540.

YI, B., JAGADISH, H. V., AND FALOUTSOS, C. 1998. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the 14th International Conference on Data Engineering (ICDE'98).* 201–208.

YOO, J. S. AND SHEKHAR, S. 2004. A partial join approach for mining co-location patterns. In *Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems (GIS'04).* 241–249.

YUAN, J., ZHENG, Y., XIE, X., AND SUN, G. 2011. Driving with knowledge from the physical world. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11).* 316–324.

YUAN, J., ZHENG, Y., ZHANG, C., XIE, W., XIE, X., SUN, G., AND HUANG, Y. 2010. T-drive: Driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances on Geographical Information Systems (GIS'10).* 99–108.

ZHANG, Q. AND LIN, X. 2004. Clustering moving objects for spatial-temporal selectivity estimation. In *Proceedings of the 15th Australasian Database Conference (ADC'04).* 123–130.

ZHANG, R., LIN, D., RAMAMOHANARAO, K., AND BERTINO, E. 2008. Continuous intersection joins over moving objects. In *Proceedings of the 24th International Conference on Data Engineering (ICDE'08).* 863–872.

ZHANG, R., QI, J., LIN, D., WANG, W., AND WONG, R. C.-W. 2011. A highly optimized algorithm for continuous intersection join queries over moving objects. *VLDB J. 21,* 4, 561–586.

ZHENG, K., ZHENG, Y., XIE, X., AND ZHOU, X. 2012. Reducing uncertainty of low-sampling-rate trajectories. In *Proceedings of the 28th IEEE International Conference on Data Engineering (ICDE'12)*. 1144–1155.

ZHENG, Y., XIE, X., AND MA, W. 2010. GeoLife: A collaborative social networking service among user, location and trajectory. *IEEE Data Engin. Bull. 33*, 2, 32–40.

ZHENG, Y. AND ZHOU, X. 2011. *Computing with Spatial Trajectories*. Springer.

ZHU, F., YAN, X., HAN, J., YU, P. S., AND CHENG, H. 2007. Mining colossal frequent patterns by core pattern fusion. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE'07)*. 706–715.

Received February 2012; revised March 2012; accepted May 2012