Efficient Notification of Meeting Points for Moving Groups via Independent Safe Regions

Jing Li, Jeppe Rishede Thomsen, Man Lung Yiu, and Nikos Mamoulis

Abstract—In applications like social networking services and online games, multiple moving users which form a group may wish to be continuously notified about the best meeting point from their locations. A promising technique for reducing the communication frequency of the application server is to employ safe regions, which capture the validity of query results with respect to the users' locations. Unfortunately, the safe regions in our problem exhibit characteristics such as irregular shapes and inter-dependencies, which render existing methods that compute a single safe region inapplicable to our problem. To tackle these challenges, we first examine the shapes of safe regions in our problem's context and propose feasible approximations for them. We design efficient algorithms for computing these safe regions. We also study a variant of the problem called the sum-optimal meeting point and extend our solutions to solve this variant. Experiments with both real and synthetic data demonstrate the effectiveness of our proposal in terms of computational and communication costs.

Index Terms—Query processing, spatial databases

1 Introduction

Recently, social networking services in the ad-hoc mobile environment have attracted significant attention [1]. Such services exist in many popular social websites including *Facebook* and *Foursquare*.[1] Managing the moving data arising from such services brings new challenges due to both spatial and social constraints.

In this paper, we propose a novel monitoring problem, Meeting Point Notification (MPN) for multiple moving users: given a group of moving users U and a set of points of interest (*POI*) P, MPN continuously reports the optimal meeting point $p^o \in P$ to users in U such that the maximum distance between any user and p^o is minimized. MPN is motivated by many applications in social networks, location-based games and massively multiplayer online (MMO) games [2], [3].

A real application relevant to MPN is EchoEcho,[2] invented by Google Venture. EchoEcho assists users to browse their friends' real-time locations and share their own. As a highlight feature, EchoEcho allows a user to continuously observe her friends' locations regarding to a predetermined meeting point. Mobile users with such interests have also been investigated in the collaborative system research [4].

Furthermore, many popular social networking applications, e.g., *event calendar* in *Facebook*,[3] assist users to share and synchronize event updates. These applications are designed to detect updates and suggest the necessary rearrangements automatically. As an example, consider a new event created in the event calendar, e.g., enjoying Italian food together. A group of users $\{u_1, u_2, u_3\}$ are interested and participate in it (see Fig. 1a for illustration). The event calendar initially recommends a restaurant, i.e., p_1, based on the current locations of these users at timestamp t_1. However, due to unpredictable traffic, the velocities of different users may change and thus the optimal meeting point may also change. In Fig. 1a, the locations of users change from $u_i(t_1)$ to $u_i(t_2)$. Due to a traffic jam, user u_1 advances toward p_1 with low speed and reaches $u_1(t_2)$. Thus, at timestamp t_2, the optimal meeting point becomes p_2. With the help of MPN, such a change of the optimal meeting point can be detected and thus subsequent events in the event calendar can be rearranged in advance.

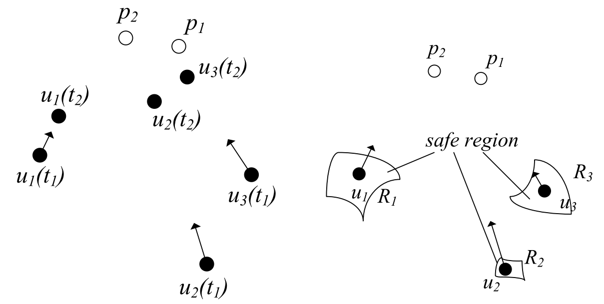
Besides social networking services, MPN also finds application in location-based games, such as the famous outdoor GPS game, Tourality.[4] To win this game, the distributed players of a team should reach one of geographically defined spots (*POI*s) by running, biking or driving as fast as possible. During a game, MPN can be used to dynamically adjust the first meeting spot based on real-time locations of players and thus shorten the meeting time.

Limitations of bandwidth and battery power raise challenges for mobile search problems, including MPN. Thus, the main optimization goal for these applications is to minimize communication frequency [5], [6], [7], [8], [9]. This goal also reduces unnecessary computational workload at the server because the communication cost between the clients and the server is reduced. We adopt the same goal:

1. www.foursquare.com
2. www.echoecho.me
3. apps.facebook.com.
4. www.tourality.com.

- *J. Li and N. Mamoulis are with the Department of Computer Science, University of Hong Kong, Hong Kong. E-mail: {jli, nikos}@cs.hku.hk.*
- *J.R. Thomsen and M.L. Yiu are with the Department of Computing, Hong Kong Polytechnic University, Hong Kong. E-mail: {csjrthomsen, csmlyiu}@comp.polyu.edu.hk.*

Manuscript received 27 Oct. 2013; revised 4 Apr. 2014; accepted 13 June 2014. Date of publication 30 June 2014; date of current version 1 June 2015. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TKDE.2014.2334304



(a) Auto-detection (b) Irregular safe regions

Fig. 1. Motivation.

minimize the communication frequency, i.e., the frequency by which users issue update messages to the server.

A straightforward solution is to force each client (i.e., user) to communicate with the server periodically (e.g., every second). However, this solution incurs huge computation and communication costs at the server side. We need to develop an efficient solution that reduces the communication cost between the server and the users. Previous work [10], which considers similar applications under road networks, only develops techniques that reduce road network distance computations but does not consider minimizing the communication cost. Thus, these techniques are inapplicable to our problem.

Motivated by this, we propose novel solutions based on the *safe region* concept. *Safe regions* are a set of geographical regions, one for each user, such that if each user stays inside her region, the query result will remain the same. For instance, in Fig. 1b the optimal meeting point is p_1 as long as all users stay in their own safe regions (R_1, R_2, R_3). The use of safe regions for multiple users raises several challenges. First, existing safe region computation techniques for a single user are not applicable for computing safe regions for a group of users, because these regions are not independent. Second, the safe regions have irregular shapes (as we demonstrate in Section 3.2), unlike simple-shaped safe regions considered in previous work (e.g., a Voronoi cell [11]). Third, it is infeasible to pre-compute the safe regions for multiple users because multiple safe regions depend on the multiple locations of moving users, which are unpredictable.

In our preliminary work [12], we have proposed circular safe regions that are easy to compute, and tile-based safe regions that offer better approximations of maximal safe regions. In this paper, our new contributions include:

- a buffering optimization that avoids repeated index accesses (see Section 5.4),
- a problem variant called the sum-optimal meeting point and our solutions for it (see Section 6), and
- additional experiments that demonstrate effectiveness and efficiency of our new contributions (see Section 7).

The paper is organized as follows. First, we review related work in Section 2. Then, we introduce our notations and define the problem formally in Section 3. Next, we present our solutions in Sections 4 and 5, together with their optimizations. We study the problem variant for the sum-optimal meeting point in Section 6. Our methods are evaluated using real and synthetic data in Section 7. Finally, we conclude our paper in Section 8.

2 Related Work

Previous work on processing moving queries over mobile data can be classified into two categories: i) report query results to a single user continuously, e.g., kNN queries [11], [13], [14], [15], [16], circular range queries [17], moving window (rectangle range) queries [5], [18]; ii) detect relationships among moving objects, e.g., proximity detection [9] and constraints monitoring [19].

The safe region concept has been widely used in moving query processing to reduce the communication cost between clients and servers. When a user registers a continuous query, the server will return POIs along with a safe region. The query result remains the same if the user stays inside the current safe region. Upon leaving the safe region, the user requests from the server a updated result together with a new safe region. The shape of the safe region depends on the query type, e.g., an order-k Voronoi cell for a kNN query [18], or an arc-based region for a range query [17]. Defining safe regions for our problem is challenging because: 1) the safe regions for MPN have irregular shapes and are thus hard to compute; 2) the safe regions of users are interdependent and the users change their locations dynamically and unpredictably, rendering pre-computation techniques (e.g., as Voronoi cells [20]) inapplicable.

Proximity detection [9] helps a user to maintain a list of friends who are within a distance threshold from her. Since both the user and her friends are moving, Yiu et al. [9] proposes self-tuning policies to automatically assign an adjustable safe region for each user. However, the work of [9] does not consider POIs where the users are supposed to meet.

The snapshot version of our problem is equivalent to the group nearest neighbor (GNN) query [21], which attempts to find a POI p that minimizes total distance between p and a set of users' locations. The group enclosing query [22] is a specialized GNN, which minimizes the maximum distance among a POI and the users. Contrary to these works, we focus on computing safe regions in order to minimize the communication cost.

The most related work to ours is [10], which focuses on monitoring GNN in road networks. Our work is different in two aspects: 1) our problem does not consider the road network; 2) the solutions in [10] aim at minimizing computations at the server side and thus cannot be applied to solve MPN. Finally, a related problem [23] is to continuously identify the object from a given set of moving objects, which is superior to others with respect to its aggregate distance toward a set of selected *POI*s. This problem and its solutions are also different from our work.

3 Problem Setting

We first introduce the preliminary concepts and the system architecture. Then, we illustrate the unique characteristics of the search space and safe regions in our problem. In the end, we state our main objectives in this paper.

TABLE 1
Notation

Notation	Meaning
U	a group of users
u_i	a user or its location
P	points of interest
$\|p,u\|$	Euclidean distance from p to u
$\|p,S\|_{max}$	max. dist. from p to a set S, i.e., S is R or U
$\|p,S\|_{min}$	min. dist. from p to a set S, i.e., S is R or U
p^o	the current optimal meeting point
$\|p,U\|^\dagger$	the dominant distance under U
$\|p,\mathcal{R}\|^\top$	the dominant max. distance under $\mathcal{R}$
$\|p,\mathcal{R}\|^\perp$	the dominant min. distance under $\mathcal{R}$
$u_p^\top$	the dominant user that contributes to $\|p,\mathcal{R}\|^\top$
$u_p^\perp$	the dominant user that contributes to $\|p,\mathcal{R}\|^\perp$
$\mathcal{R}$	a set of safe regions for U
$\mathcal{R}^*$	a set of maximal safe regions

3.1 Preliminaries and System Architecture

We first provide the definitions for distances, the optimal meeting point, and safe regions. Unless otherwise stated, we denote both a user and her location by u_i. Table 1 summarizes the notations to be used throughout the paper.

Definition 1 (Distances). *Let $\|p,l\|$ be the Euclidean distance between points p and l. The* minimum distance *and the* maximum distance *from a point p to a set/region S are*

$$\|p,S\|_{min} = \min_{l\in S} \|p,l\| \quad (1)$$

$$\|p,S\|_{max} = \max_{l\in S} \|p,l\|. \quad (2)$$

Definition 2 (Optimal meeting point). *Given a group of users U and a data set of points P, the* optimal meeting point *p^o is the point in P with the smallest $\|p^o,U\|_{max}$. It is also called MAX-GNN [21].*

Definition 3 (Independent safe region group). *Let m be the number of users in U. A group of regions $\mathcal{R} = \langle R_i|_{i=1}^m\rangle$ is said to be* independent *if the optimal meeting point p^o is the same for every instance of user locations $\forall \langle l_1, l_2, \cdots, l_m\rangle \in R_1 \times R_2 \times \cdots \times R_m$.*

Definition 4 (Maximal safe region group). *$\mathcal{R}^* = \langle R_i^*|_{i=1}^m\rangle$ is said to be a set of maximal safe regions if no other (independent) set of safe regions $\mathcal{R}' = \langle R_i'|_{i=1}^m\rangle$ satisfies: $\mathcal{R}' \neq \mathcal{R}^*$ and $R_i^* \subseteq R_i'$ $\forall\, i = 1\cdots m$.*

As an example, Fig. 2a illustrates the minimum distances (from p_1 to a circle, and from p_2 to a square) and the maximum distances (from p_3 to a circle, and from p_2 to a square). At timestamp t_1 (t_2) in Fig. 1a, the optimal meeting point for the locations of u_1–u_3 at t_1 is p_1 (p_2). As shown in Fig. 1b, the independent safe regions for three users u_1–u_3 are R_1-R_3. Note that the safe regions (for the optimal meeting point) can have irregular shapes; we will elaborate on this issue shortly.

In this paper, we adopt the client-server architecture which is widely used in moving query processing [7], [17], [18]. Fig. 3 illustrates this architecture. The server manages a data set P of points-of-interest (e.g., restaurants, cafes) and indexes it by an R-tree. A group of users U wish to receive notifications of their optimal meeting point $p^o \in P$ from the server continuously. Besides the result p^o, the server also reports a safe region R_i to each user $u_i \in U$. By Definition 3, the optimal meeting point remains unchanged if every user u_i moves within her safe region R_i. Therefore, these safe regions serve to reduce the communication frequency of the server (and its computational overhead) significantly.

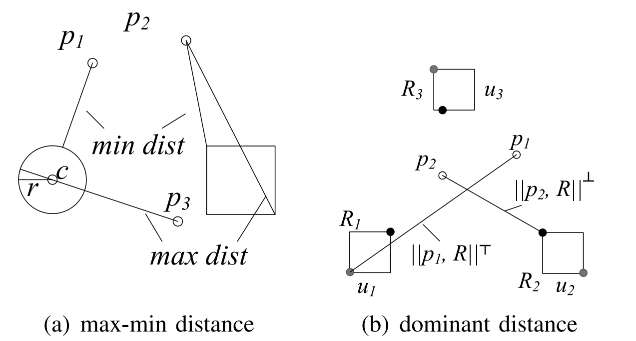


Fig. 2. Distance.

The system is triggered when a user $u_i \in U$ leaves her safe region R_i. Then, u_i sends her current location to the server (Step 1). Next, the server probes the current locations of other users in the group U (Step 2). Having received replies from all users in U, the server recomputes and notifies each user u_i about the optimal meeting point p^o and her corresponding safe region R_i (Step 3). In summary, the server and users communicate via three types of messages.

As we will show in Section 3.2, maximal safe regions have irregular shapes and raise challenges in computation and representation. Our objectives are as follows:

1) Design concise representations for safe regions;
2) Develop efficient algorithms for computing them.

In this paper, we will investigate conservative approximations for maximal safe regions. Specifically, we will study circular safe regions in Section 4 and tile-based safe regions in Section 5.

3.2 Characteristics of Safe Region Group

This section describes the unique characteristics exhibited by the safe regions in our problem. By Definition 3, the possible groups of safe regions indeed form a huge search space: a $m \cdot d$ dimensional space, where m is the number of users and d is the number of spatial dimensions. For example, for two users ($m = 2$) and the planar space ($d = 2$), the search space becomes four-dimensional.

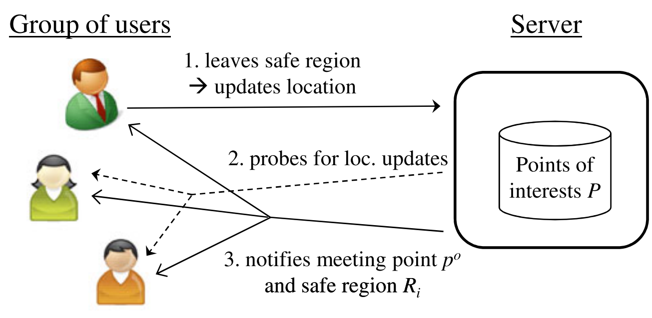


Fig. 3. System architecture.

Users				u			v			
Objects		b			a					c
Location	0	1	2	3	4	5	6	7	8	9

(a) locations of users and objects in 1D space

9	a	a	a	a	a	c	c	c	c	c
8	a	a	a	a	a	a	c	c	c	c
7	a	a	a	a	a	a	a	c	c	c
6	a	a	a	a	a	a	a	a	c	c
5	a	a	a	a	a	a	a	a	a	c
4	b	a	a	a	a	a	a	a	a	a
3	b	b	a	a	a	a	a	a	a	a
2	b	b	b	a	a	a	a	a	a	a
1	b	b	b	b	a	a	a	a	a	a
0	b	b	b	b	b	a	a	a	a	a
$v \backslash u$	0	1	2	3	4	5	6	7	8	9

(b) optimal meeting point for each group of user locations

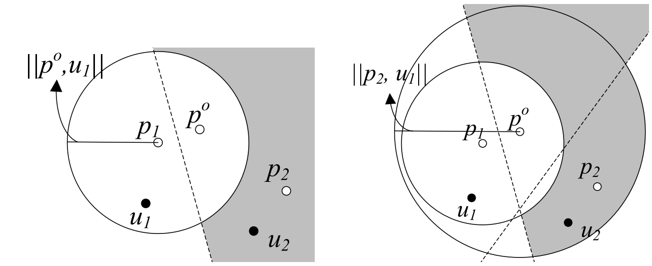
Fig. 4. Optimal meeting point for a two-user group, with objects in 1D space.

We first conduct a case study to visualize the search space for the case $m = 2$ and $d = 1$ (i.e, each user location is just a single value). Fig. 4a shows the locations of two users u, v and three points-of-interest a, b, c. Fig. 4b illustrates the optimal meeting point for every group of locations for user u, v. Each cell (at ith column, jth row) contains the optimal meeting point when $u = i$ and $v = j$. For instance, the current user locations are $u = 3$ and $v = 6$, so the current optimal meeting point is a (see the cell at third column, sixth row). For readability, the cells are colored based on their optimal meeting points (see Fig. 4b). It appears that the cells with the same color form a connected 'hyper-region' in the high-dimensional search space, e.g., the diamond-like 'hyper-region' for point a. Unfortunately, we are unable to decompose such a high-dimensional 'hyper-region' into an independent safe region group $\langle R_i|_{i=1}^{m}\rangle$ for the users. First, two cells with the same color are not necessarily connected in the spatial domain. For instance, both groups $\langle 3, 9\rangle$ and $\langle 5, 0\rangle$ for $\langle u, v\rangle$ take a as the optimal meeting point. However, user v cannot travel from location 9 to 0 directly without visiting locations 1-4, which have other optimal meeting points. Second, the maximal safe region of a user is restricted by that of another user. For instance, if the safe region for v is the interval 5-9, the safe region for u can only be the interval 0-4. Otherwise, if $u = 5$ but $v = 9$, the optimal meeting point is no longer a. Third, the groups of maximal safe regions obtained from the search space are not unique. For instance, consider two groups of safe regions: i) $\langle$2-4, 3-9$\rangle$, and ii) $\langle$0-4, 5-9$\rangle$. Both groups are valid and they take a as the optimal meeting point. Finally, the safe regions have irregular shapes, which we will elaborate shortly.

All these are unique characteristics in our problem, rendering existing safe region techniques [7], [17], [18] inapplicable.

Shapes of maximal safe regions. We proceed to illustrate the fact that the maximal safe regions in our problem have irregular shapes. Fig. 5 shows an example in the two-dimensional space ($d = 2$) with two users u_i ($m = 2$) and three data points. The current optimal meeting point is marked as p^o.

The entire search space cannot be visualized here as it has $m \cdot d = 2 \cdot 2 = 4$ dimensions. For the sake of illustration,



(a) 1-point partition (b) 2-point partition

Fig. 5. Safe region example.

we consider the special case that u_1 has a fixed location and then attempt to find the maximal safe region of u_2.

Let us examine how the point p_1 affects the safe region of u_2 (see Fig. 5a). Consider i) the bisector line between points p_1 and p^o, and ii) the circle at center p_1 with radius $\|u_1, p^o\|$. If u_2 moves across the bisector line in i), then both u_1 and u_2 become closer to p_1 than to p^o. If u_2 moves inside the circle in ii), then the optimal meeting point will be decided by the 'further-away' u_1, who is closer to p_1 than p^o. Thus, the safe region (in gray color) is bounded by the shapes i) and ii).

Following a similar argument, we can derive the boundaries of the safe region of u_2 with respect to the point p_2. The maximal safe region of u_2 is restricted by both p_1 and p_2. Fig. 5b shows that this region (in gray color) has an irregular shape.

In general, the maximal safe regions in our problem have irregular shapes, especially in typical applications which involve many more users and data points than in the above example. These irregular safe regions raise two challenges: i) they are time-consuming to compute, and ii) they are hard to be represented in a concise manner.

4 Circular Safe Region Approach

In this section, we approximate the maximal safe regions of users by circles due to simplicity. We first study the condition for verifying a set of safe regions. Then, we design an algorithm for computing circular safe regions.

4.1 Verification of Safe Regions

An essential task in our problem is to verify whether a set of regions $\{R_i|_{i=1}^{m}\}$ satisfies Definition 3. By definition, there are infinitely many instances of user locations in those regions. It is infeasible to test all the instances one-by-one.

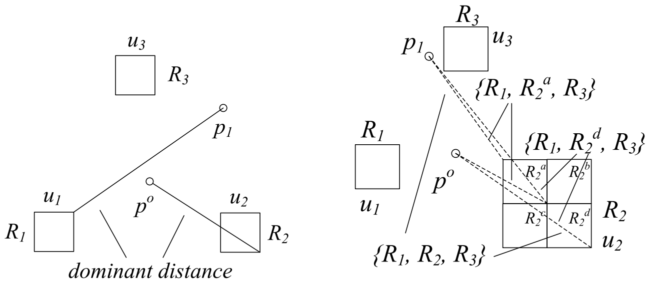
In this section, we plan to establish a conservative condition for verifying safe regions in an efficient manner. Before that, we first define *dominant distances* and *dominant user*:

Definition 5. *Given a data point $p \in P$ and a user set U, the dominant distance is defined as*

$$\|p, U\|^{\dagger} = \max_{u_i \in U} \|p, u_i\|.$$

Given a data point $p \in P$ and a set of safe regions $\mathcal{R}$, the dominant minimum and maximum distances are defined as:

$$\|p, \mathcal{R}\|^{\perp} = \max_{R_i \in \mathcal{R}} \|p, R_i\|_{min}, \tag{3}$$



(a) conservative verification (b) divide-and-conquer

Fig. 6. Verifications of safe regions.

$$\|p, \mathcal{R}\|^{\top} = \max_{R_i \in \mathcal{R}} \|p, R_i\|_{max}. \tag{4}$$

A user is denoted as $u_p^{\dagger}$ if he contributes to the dominant distance with respect to point p.

Observe that the optimal meeting point is the point with the smallest dominant distance $\|p, U\|^{\dagger}$. Regardless of the actual locations of users (in their safe regions), $\|p, \mathcal{R}\|^{\perp}$ serves as a lower-bound of $\|p, U\|^{\dagger}$, and $\|p, \mathcal{R}\|^{\top}$ serves as an upper-bound of $\|p, U\|^{\dagger}$. As an example, in Fig. 2b, $\|p_2, \mathcal{R}\|^{\perp}$ is the maximum over the minimum distances from p_2 to each region (corner in black), and $\|p_1, \mathcal{R}\|^{\top}$ is the maximum over the maximum distances from p_1 to each region (corner in gray).

We now establish a conservative test (Lemma 1) for verifying a set of safe regions with respect to a given data point $p \in P$ and the optimal meeting point p^o. This test is conservative in the sense that it has no false positives but it may have false negatives, i.e., i) if the test returns true, then p^o is definitely optimal when the users remain in $\mathcal{R}$; ii) if the test returns false, then p^o may not be optimal. We denote this test as *Verify*$(\mathcal{R}, p^o, p)$ throughout the paper. This test is efficient as its time complexity is $O(m)$.

Lemma 1 (Conservative verification). *Given a set of regions $\mathcal{R} = \{R_i|_{i=1}^m\}$, if for a point $p \in P$ and $p \neq p^o$*

$$\|p^o, \mathcal{R}\|^{\top} \leq \|p, \mathcal{R}\|^{\perp}, \tag{5}$$

then the dominant distance of p^o must be smaller than or equal to that of p.

Proof. For any instance $\{l_i|_{i=1}^m\}$ of $\mathcal{R}$, by definition of dominant max. (min.) distance, we have

$$\|p^o, \{l_i|_{i=1}^m\}\|^{\dagger} \leq \|p^o, \mathcal{R}\|^{\top}$$

and

$$\|p, \mathcal{R}\|^{\perp} \leq \|p, \{l_i|_{i=1}^m\}\|^{\dagger}.$$

Combining both equations with Equation (5), we derive

$$\|p^o, \{l_i|_{i=1}^m\}\|^{\dagger} \leq \|p, \{l_i|_{i=1}^m\}\|^{\dagger},$$

which means that all instances in $\mathcal{R}$ are valid. □

As an example, Fig. 6a shows two data points and three users (with their safe regions). Note that $\|p^o, \mathcal{R}\|^{\top} = \|p^o, R_2\|_{max}$ and $\|p_1, \mathcal{R}\|^{\perp} = \|p_1, R_1\|_{min}$. Since $\|p^o, R_2\|_{max} < \|p_1, R_1\|_{min}$, by Lemma 1, we conclude that p_1 cannot replace p^o as the optimal meeting point (and thus the safe regions are valid).

4.2 Algorithm

Although maximal safe regions have irregular shapes, they can be conservatively approximated as circles. We now assign each user u_i a circular safe region $R_i = \odot(u_i, r)$, where u_i is the current user location and r is the radius. Note that the same radius r is used across different R_i.

To reduce the communication cost between the server and the users, the value r should be as large as possible. The following theorem decides the maximum radius r such that the safe regions remain valid.

Theorem 1 (Maximal circles). *The maximum radius of circles for safe regions is*

$$r_{max} = \frac{\min_{p \in P - \{p^o\}}(\|p, U\|_{max}) - \|p^o, U\|_{max}}{2}. \tag{6}$$

Proof. Let $R_i = \odot(u_i, r)$, a circle with radius r and center as the current user location u_i. We have: $\|p, R_i\|_{max} = \|p, u_i\| + r$ and $\|p, R_i\|_{min} = \|p, u_i\| - r$.

By substituting these equations into Equation (5) in Lemma 1, for any point $p \in P - \{p^o\}$, we have

$$\max_{u_i \in U}(\|p^o, R_i\|_{max}) \leq \max_{u_j \in U}(\|p, R_j\|_{min}),$$

$$\max_{u_i \in U}(\|p^o, u_i\| + r) \leq \max_{u_j \in U}(\|p, u_j\| - r).$$

By rearranging the terms, we obtain

$$2r \leq \max_{u_j \in U}(\|p, u_j\|) - \max_{u_i \in U}(\|p^o, u_i\|),$$

which is equivalent to

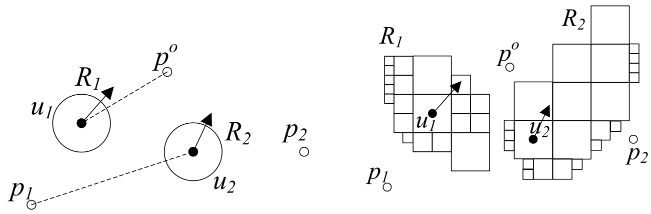
$$r \leq \frac{\|p, U\|_{max} - \|p^o, U\|_{max}}{2}. \tag{7}$$

Note that Equation (7) holds for any point $p \in P - \{p^o\}$. By taking the minimum value of all $\|p, U\|_{max}$, we obtain: $r_{max} = \frac{\min_{p \in P - \{p^o\}}(\|p, U\|_{max}) - \|p^o, U\|_{max}}{2}$. □

Algorithm 1 is the pseudo-code for computing circular safe regions for users. Assume that the data set set P is indexed by an R-tree. First, the algorithm finds the best two meeting points by calling an existing algorithm [24] on the R-tree of P. Note that the second best meeting point is the point p that contributes to $\min_{p \in P - \{p^o\}}(\|p, U\|_{max})$. Then, it computes the maximum radius r_{max} by Equation (6) and returns the corresponding circular safe regions to the users.

Algorithm 1. Circle-MSR (Set of users U, Dataset P)

$p^o, p \leftarrow$ FindMaxGNN(U, P, 2) $\triangleright$ apply algo. in [24]
compute the radius r_{max} $\triangleright$ apply Equation (6)
for each user $u_i \in U$ **do**
  return the safe region $\odot(u_i, r_{max})$ to u_i



(a) circular safe regions (b) tile-based safe regions

Fig. 7. Comparisons of safe regions.

Discussion. Although circular safe regions can be computed efficiently, they may not tightly capture maximal safe regions. For instance Fig. 7a contains two users u_1-u_2 and three points p_1-p_2 and p^o. Since p_1 is the next optimal meeting point, according to Equation (6), the radius for circles are determined by two distances $\|p^o, u_1\|$ and $\|p_1, u_2\|$. Thus, the circular safe regions are depicted in Fig. 7a. In the next section, we propose a tighter approximation of maximal safe regions, named *the tile-based safe regions*, in order to further reduce the communication frequency. As illustrated in Fig. 7b, the tile-based safe regions are much more tighter than the circular safe regions in Fig. 7a.

We are aware of a tight pruning technique [25] that utilizes half-spaces for deciding whether every point in a query rectangle R is closer to an object rectangle O_a rather than another object rectangle O_b. Nevertheless, this technique is not applicable to our problem because: i) we use multiple safe regions for multiple users respectively, instead of using a single rectangle R, and ii) a safe region group (e.g., $\langle R_1, R_2\rangle$) can be valid even when some part of R_1, R_2 do not take p^o as the nearest neighbor (see Fig. 7b).

5 Tile-Based Safe Region Approach

A *tile*, as its name implies, is a square region (with side-length δ). Tiles can be assembled to represent an irregular shape and thus serve as a tighter approximation of maximal safe regions. A tile-based safe region can be represented in a concise manner, as shown in our preliminary work [12]; we omit these techniques here due to space limitations. In the remainder of this section, we first show a tighter verification method for tiles. Next, we design an algorithm for computing such tile-based safe regions. Then, we propose techniques to optimize the efficiency of tile verification. Finally, we suggest a buffering optimization that avoids repeated accesses to an R-tree.

5.1 Divide-and-Conquer Verification for Tiles

We start by showing that the verification condition in Lemma 1 is not tight. Fig. 6b shows three users u_1, u_2, u_3 and two data points p^o and p_1. Here, u_2 is the dominant user for both points p^o and p_1. Consider the safe region group $\mathcal{R} = \langle R_1, R_2, R_3\rangle$. As depicted in Fig. 6b, the max. distance for $p^o(\|p^o, R_2\|_{max})$ is larger than the min. distance for p_1 ($\|p_1, R_2\|_{min}$). By Lemma 1, $\mathcal{R}$ cannot be verified. This phenomenon happens due to the dominant min. and max. distances for the same dominant user (e.g., u_2), yet they are contributed by two different locations inside R_2.

On the other hand, if we divide R_2 into four smaller tiles ($R_2^a, R_2^b, R_2^c, R_2^d$) as shown in Fig. 6b, then $\mathcal{R}$ can pass the verification. Consider the safe region group $\mathcal{R}' = \langle R_1, R_2^a, R_3\rangle$ for example. $\mathcal{R}'$ passes the verification since $\|p^o, \mathcal{R}'\|_{max}$ is less than $\|p_1, \mathcal{R}'\|_{min}$. Similarly, the safe region group $\mathcal{R}'' = \langle R_1, R_2^d, R_3\rangle$ passes the verification since $\|p^o, \mathcal{R}''\|_{max} \le \|p_1, \mathcal{R}''\|_{min}$. After applying Lemma 1 to the remaining two groups of safe regions ($\langle R_1, R_2^b, R_3\rangle$, $\langle R_1, R_2^c, R_3\rangle$), we conclude that $\mathcal{R}$ is valid.

Our next question is how to determine a suitable size δ for a tile s. If δ is too small, then many tiny tiles are examined and incur significant computation cost. If δ is too large, then $\mathcal{R}$ may not be able to pass the verification.

To tackle this problem, we propose a divide-and-conquer method for verification (Algorithm 2). The initial size of the tile s will be discussed in the next section. The parameter L is used to control the number of recursion levels (and thus the computation cost). Suppose that $\mathcal{R} = \langle R_1, R_2, \ldots, R_i, \ldots, R_m\rangle$ is a valid safe region group (i.e., passed the verification). The algorithm aims to check whether s is a valid safe region for user u_i with respect to the existing safe regions $R_1, \ldots, R_{i-1}, R_{i+1}, \ldots, R_m$ of other users in $\mathcal{R}$. If yes, then we can guarantee that $R_i \cup \{s\}$ is also a valid safe region for user u_i.

At lines 1-3, we apply a function *Tile-Verify* to verify the tile s for the user u_i with respect to the safe regions of other users in $\mathcal{R}$. Efficient implementations of *Tile-Verify*, and index pruning techniques (on R-tree), will be studied in Section 5.3. If s passes the verification, then we add it into the safe region of u_i. Otherwise, we divide s into four sub-tiles s', and then call the method recursively on s' (see lines 5-8). Note that recursion stops when the recursion level L reaches 0.

Algorithm 2. Divide-Verify (Safe region group $\mathcal{R}$, User u_i, Tile s, Optimal point p^o, Dataset P, Level L)

```
1:  if ∀ p ∈ P − {p^o}, Tile-Verify (R, u_i, s, p, p^o) is true
    then
2:      R_i ← R_i ∪ {s}
3:      return true
4:  flag ← false
5:  if L > 0 then              ▷ control the recursion level
6:      divide s into four sub-tiles
7:      for each sub-tile s' of s do
8:          if Divide-Verify (R, u_i, s', p^o, P, L − 1) then
9:              flag ← true
10: return flag
```

5.2 Algorithm

Having introduced a divide-and-conquer verification method *Divide-Verify*, we are ready to present an algorithm for computing tile-based safe regions (Algorithm 3). Each safe region R_i is modeled as a set of tiles, so it can be used to approximate an irregular shape (see Fig. 7a). The main idea of the algorithm is to browse the tiles around each user u_i in a systematic way, apply verification on them, and then add valid tiles into a safe region R_i.

Recall that Algorithm 1 computes the safe region of each user u_i as a circle $\odot(u_i, r_{max})$. The maximal tile (square) in each circle must also be a valid safe region. Thus, we set the

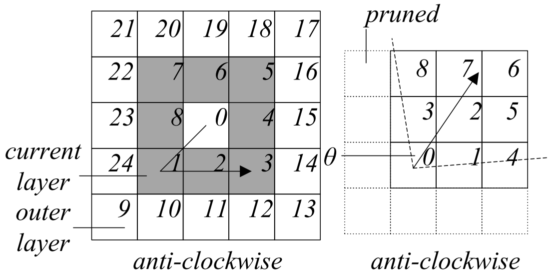


Fig. 8. Ordering for tiles.

tile size $\delta = \sqrt{2} \cdot r_{max}$ and add a tile $\boxdot(u_i, \delta)$ into its corresponding safe region R_i (lines 1-4).

The parameter α specifies the (maximum) number of tiles to be assigned to each safe region R_i. It can also be used to bound the number of iterations in lines 5-11. In each iteration, the algorithm examines the safe regions of users in a round-robin manner.

We call a function *Next-Tile* to get the next tile s for user u_i. The implementation of *Next-Tile* will be discussed shortly. Then, it tests the new tile s with other users' safe regions by calling *Divide-Verify* (line 9). The loop terminates either when i) the test returns true, or ii) s is empty, i.e., *Next-Tile* has exhausted all tiles for u_i.. At the end, the algorithm returns a safe region R_i to each user u_i.

Algorithm 3. Tile-MSR (Set of users U, Dataset P, Tile limit α, Split level L)

```
1:  compute p^o and r_max                 ▷ apply Algorithm 1
2:  δ ← √2 · r_max                        ▷ initial tile size
3:  for each user u_i in U do
4:      R_i ← { ⊡(u_i, δ) }               ▷ initial safe region
5:  for τ ← 1 to α do                     ▷ control running time
6:      for each user u_i ∈ U do          ▷ round robin
7:          repeat
8:              s ← Next-Tile ( u_i, δ )   ▷ by tile ordering
9:              flag ← Divide-Verify ( R, u_i, s, p^o, P, L)
10:         until flag = true or s = ∅
11: for each user u_i ∈ U do
12:     return the safe region R_i to u_i
```

We examine two possible orderings for *NextTile* to select the next tile. In Fig. 8, the tiles are numbered by the their insertion order. The first tile centered at u_i is numbered as 0.

*Undirected ordering***.** This approach picks the next tile based on the anti-clockwise order as shown in Fig. 8. When all tiles in the current layer have been exhausted, it checks whether some tile in the current layer has been inserted in the safe region. If yes, then it picks the next tile in an outer layer and repeats the process. Otherwise, it returns a `null` tile, meaning that any subsequent tile cannot become a valid tile for the user.

Directed ordering. Existing studies [26] show that the travel direction of a user u_i in the near future has a limited angle deviation θ from his current one. θ is learned from u_i's recent travel directions. We can exploit this feature and examine only the tiles whose subtended angles at u_i deviate by less than θ. By incorporating this idea into the above *undirected ordering*, we are able to select more tiles that are likely to cover the future locations of u_i. Fig. 8 shows an example of this directed ordering.

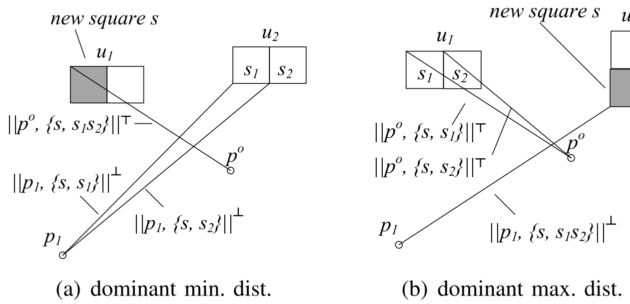


Fig. 9. Examples of GT-Verify.

5.3 Efficient Implementation of Tile Verification

The running time of Algorithm 3 is dominated by the time for verifying tiles, i.e., the recursive *Divide-Verify* function. This function needs to invoke the *Tile-Verify* function for every point $p \in P - \{p^o\}$ (line 1). In this section, we optimize this step in order to reduce the overall running time. We first study how the *Tile-Verify* function can be implemented efficiently. Then, we propose a technique for pruning a large portion of points in $P - \{p^o\}$ without processing them one-by-one.

Individual Tile Verification (IT-Verify). This is a basic technique for verifying a new tile s to be allocated to user u_i. Given a valid safe region group $\langle R_i|_{i=1}^{m}\rangle$ for all users, consider a *tile group* $\langle s_1 \in R_1, \ldots, s_i = s, \ldots, s_m \in R_m\rangle$ which contains a tile from each user, where (i) $s_i = s$, and (ii) s_j is a tile from R_j for any other user $u_j \neq u_i$.

IT-Verify would enumerate all possible tile groups (as defined above) and verify them. If any group fails, then s is not valid as part of the safe region of user u_i. However, such an implementation suffers from high computation cost due to the huge number of tile groups formed by the safe regions of other users $u_j \neq u_i$. The number of such groups is $O(\Pi_{i=1}^{m}|R_i|)$, where $|R_i|$ is the number of tiles in the safe region R_i.

Group Tile Verification (GT-Verify). Instead, we propose an optimized verification method for the new tile s. The main idea of *GT-Verify* is to group tiles and test entire groups collectively, reducing the total number of checks significantly.

We illustrate two main types of grouping strategies. Fig. 9 depicts two users u_1 and u_2, the optimal meeting point p^o, and a candidate point p_1. The new tile s is colored in gray. In Fig. 9a, the maximum distance between the new tile and p^o ($\|p^o, s\|_{max}$) is the dominant max. distance for two tile groups $\langle s, s_1\rangle$ and $\langle s, s_2\rangle$. $\langle s, s_1\rangle$ ($\langle s, s_2\rangle$) has the dominant min. distance to p_1 incident to s_1 (s_2). If $\langle s, s_2\rangle$ fails in the verification test, so does $\langle s, s_1\rangle$ since $\|p_1, \langle s, s_1\rangle\|^{\perp} < \|p_1, \langle s, s_2\rangle\|^{\perp} < \|p^o, s\|_{max}$. Thus, we can group s_1 and s_2 and test $\langle s, s_1 \cup s_2\rangle$ instead of testing each group individually. In Fig. 9b, the minimum distance between the new tile and p_1 ($\|p_1, s\|_{min}$) is the dominant min. distance for two tile groups $\langle s_1, s\rangle$ and $\langle s_2, s\rangle$. $\langle s_1, s\rangle$ ($\langle s_2, s\rangle$) has the dominant max. distance to p^o incident to s_1 (s_2). If $\langle s_2, s\rangle$ fails the verification test, so does $\langle s_1, s\rangle$ since $\|p^o, \langle s_1, s\rangle\|^{\top}\|p^o, \langle s_2, s\rangle\|^{\top} >$

$\|p_1, s\|_{min}]] >$. Thus, we can group s_1 and s_2 and test $\langle s_1 \cup s_2, s\rangle$ instead of testing each group individually.

The key observation is that we can categorize tile groups involving s based on two dominant distances: $d^o = \|p^o, s\|_{max}$ and $d_p = \|p, s\|_{min}$. Using these distances, the tiles inside a safe region R_j are partitioned into four groups as shown below

$$G_j^{\downarrow\downarrow} = \langle s' \in R_j \mid \|p^o, s'\|_{max} < d^o \wedge \|p, s'\|_{min} < d_p\rangle$$
$$G_j^{\uparrow\downarrow} = \langle s' \in R_j \mid \|p^o, s'\|_{max} \geq d^o \wedge \|p, s'\|_{min} < d_p\rangle,$$
$$G_j^{\downarrow\uparrow} = \langle s' \in R_j \mid \|p^o, s'\|_{max} < d^o \wedge \|p, s'\|_{min} \geq d_p\rangle,$$
$$G_j^{\uparrow\uparrow} = \langle s' \in R_j \mid \|p^o, s'\|_{max} \geq d^o \wedge \|p, s'\|_{min} \geq d_p\rangle.$$

The following theorem establishes test conditions for these groups and ensures that they cover all possible tile groups.

Theorem 2. *Let $u_{p^o}^{\top}$ and $u_p^{\perp}$ be the users that realize the dominant max. distance of p^o and the dominant min. distance of p, respectively. Let $\{s\}_i$ be the new tile s to be allocated as the safe region of user u_i. If all tile groups are valid, then the testing for the following safe region groups must be valid:*

1) *Safe region group $R' = \langle G_1^{\downarrow\downarrow}, \ldots, \{s\}_i, G_m^{\downarrow\downarrow}\rangle$. u_i is $u_{p^o}^{\top}$ and also $u_p^{\perp}$.*
2) *Safe region group $R' = \langle G_1^{\downarrow\downarrow} \cup G_1^{\uparrow\downarrow}, \ldots, \{s\}_i, \ldots, G_m^{\downarrow\downarrow} \cup G_m^{\uparrow\downarrow}\rangle$. u_i is $u_p^{\perp}$ and another user u_j $(u_i \neq u_j)$ is $u_{p^o}^{\top}$.*
3) *Safe region group $R' = \langle G_1^{\downarrow\downarrow} \cup G_1^{\downarrow\uparrow}, \ldots, \{s\}_i, \ldots, G_m^{\downarrow\downarrow} \cup G_m^{\downarrow\uparrow}\rangle$. u_i is $u_{p^o}^{\top}$ and another user u_j $(u_i \neq u_j)$ is $u_p^{\perp}$.*
4) *If u_i is not a dominant user and $s' \in R_i$ exists such that $\|p^o, s'\|_{max} \leq d^o$ and $\|p, s'\|_{min} \leq d_p$, then all the tile groups R'' that are not covered in above safe region group are valid. Otherwise, test all these R'' by calling Verify(R'', p^o, p).*

Proof. It is easy to see that each tile group is included in the four types. We prove the converse-negative proposition of this theorem.

If 1) fails the verification, there exists a tile group $\langle s_1 \in G_j^{\downarrow\downarrow}, \ldots, s_i = s, \ldots, s_m \in G_m^{\downarrow\downarrow}\rangle$ that have user u_i as the dominant users, which fails the verification.

If 2) fails, there exists $s' \in G_j^{\uparrow\downarrow}$ for a tile group $\langle s_1 \in G_1^{\downarrow\downarrow} \cup G_j^{\uparrow\downarrow}, \ldots, s_i = s, \ldots, s_j = s', \ldots, s_m \in G_m^{\downarrow\downarrow} \cup G_m^{\uparrow\downarrow}\rangle$ (u_i as $u_p^{\perp}$ and user u_j as $u_{p^o}^{\top}$), which fails the verification.

If 3) fails, there exists $s' \in G_j^{\downarrow\uparrow}$ for a tile group $\langle s_1 \in G_1^{\downarrow\downarrow} \cup G_1^{\downarrow\uparrow}, \ldots, s_i = s, \ldots, s_j = s', \ldots, s_m \in G_m^{\downarrow\downarrow} \cup G_m^{\downarrow\uparrow}\rangle$ (u_i as $u_{p^o}^{\top}$ and user u_j as $u_p^{\perp}$), which fails the verification.

For 4), all tile groups R'' involving u_j and u_k ($j \neq i$ and $k \neq i$) as the dominant users share the same verifications. If there exists a tile $s' \in R_i$ s.t. $\|p^o, s'\|_{max} \leq d^o$ and $\|p, s'\|_{min} \leq d_p$, the group R'' with s' as the safe region for user u_i is valid in the previous verifications. Thus, R'' with s as the safe region for user u_i is valid as well. Otherwise, we check these remaining tile groups R'' by calling *Verify(R'', p^o, p)*. □

Based on the above theorem, we design the *GT-Verify* (Algorithm 4) that applies the grouping strategy. First, *GT-Verify* directly call *Verify(R', p^o, p)* to verify the new tile s together with all other users' safe regions in line 1-2. Otherwise, it partitions each safe region $R_j \in \mathcal{R}$ into four groups as described previously (line 3). From line 4-11, *GT-Verify* behaves as described in Theorem 2 by calling *Verify (R', p^o, p)* on the partitioned group.

Algorithm 4. GT-Verify(Safe region group $\mathcal{R}$, User u_i, Tile s, Point p, Optimal point p^o)

```
1:  if R' = ⟨R_1, ..., {s}_i, ..., R_m⟩ is valid then
2:      return true
3:  partition each safe region R_j ∈ R into four groups
4:  if ⟨G_1^↓↓, ..., s, G_m^↓↓⟩ is invalid
    or ⟨G_1^↓↓ ∪ G_1^↑↓, ..., {s}_i, ..., G_m^↓↓ ∪ G_m^↑↓⟩ is invalid
    or ⟨G_1^↓↓ ∪ G_1^↓↑, ..., {s}_i, ..., G_m^↓↓ ∪ G_m^↓↑⟩ is invalid then
5:      return false
6:  if ∃s' ∈ R_i s.t. ||p^o, s'||_max ≤ d^o and ||p, s'||_min ≤ d_p then
7:       return true
8:  for group R'' not covered in above groups do
9:      if Verify(R'', p^o, p) = false then
10:          return false
11: return true
```

Index Pruning. Recall that the *Divide-Verify* function invokes the *Tile-Verify* function (e.g., *IT-Verify* or *GT-Verify*) for every point $p \in P - \{p^o\}$ (line 1). In fact, many of such point p cannot become candidates to replace the optimal meeting point p^o.

Motivated by this, we formulate the following theorem to detect unpromising points that cannot become candidates.

Theorem 3. *Given a safe region group $\mathcal{R}$, a point p cannot yield better dominant distance than p^o if for any $u_i \in U$,*

$$\|p, u_i\| > \|p^o, \mathcal{R}\|^{\top} + r_i^{\dagger}, \tag{8}$$

where $r_i^{\dagger}$ is the maximum distance between user u_i's current location and its safe region boundary.

Proof. By Equation (8), we have

$$\begin{aligned}
\|p, \mathcal{R}\|^{\perp} &= \max_{R_i \in \mathcal{R}} \|p, R_i\|_{min} && \text{by Equation (3)}\\
&> \max_{u_i \in U}(\|p, u_i\| - r_i^{\dagger})\\
&> \max_{u_i \in U}(\|p^o, \mathcal{R}\|^{\top}) && \text{by Equation (8)}\\
&= \|p^o, \mathcal{R}\|^{\top}.
\end{aligned}$$

By Lemma 1, we conclude that p cannot replace p^o as the optimal meeting point. □

In order to retrieve the candidates from P, we traverse the R-tree (of P) while pruning candidates disqualified by the above theorem. For example, in Fig. 10, p_2 is a candidate but p_1 is not a candidate. Similarly, the pruning technique can be extended to the MBRs in the R-tree. For instance, MBR_2 can be pruned since its min. distance to u_1 is larger than $\|p^o, \mathcal{R}\|^{\top} + r_1^{\dagger}$. On the other hand, MBR_1 contains the potential points since it

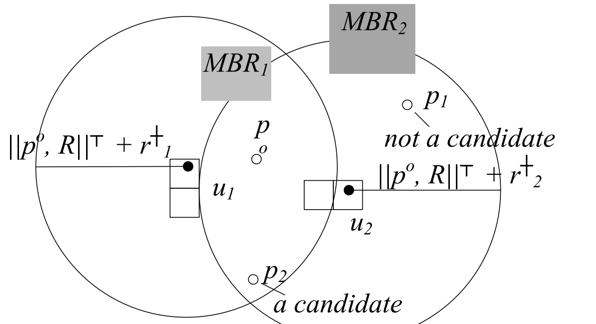


Fig. 10. Index pruning.

overlaps the circle with radius $\|p^o, \mathcal{R}\|^\top + r_1^\dagger$ and that with radius $\|p^o, \mathcal{R}\|^\top + r_2^\dagger$.

5.4 Buffering Optimization for Index Access

Observe that the computation of tile-based safe regions (Algorithm 3) invokes the *Divide-Verify* function multiple times, causing frequent accesses to the R-tree (of data set P). In this section, we present an optimization method so that Algorithm 3 accesses the R-tree exactly once, regardless of the number of calls to *Divide-Verify*.

5.4.1 Buffering Points for Verification

Our idea is to retrieve a subset of points from the R-tree and only use them in subsequent calls to the *Divide-Verify* function. Given a parameter β, we define a distance threshold λ_β as follows. We will elaborate how to reduce the sensitivity of β later.

Definition 6 (Distance threshold). *The distance threshold λ_β is defined as follows:*

$$\lambda_\beta = \frac{\|p^{\beta+1}, U\|_{max} - \|p^o, U\|_{max}}{2}. \tag{9}$$

where point p^j denotes the j-th MAX-GNN of U.

Theorem 4 states that the best β MAX-GNNs (of U) are sufficient for verifying a group location instance L, provided that each $l_i \in L$ is within distance λ_β from u_i.

Theorem 4 (Buffering condition). *Let $P^*_{1..j} = \{p^1(=p^o), p^2, \ldots, p^j\}$ be the set of the best j MAX-GNNs. Given a group location instance $L = \langle l_1, \ldots, l_m \rangle$, if $\|l_i, u_i\| \leq \lambda_\beta$ holds for every $1 \leq i \leq m$, then the MAX-GNN of L cannot be any point in $P - P^*_{1..\beta}$.*

Proof. Let p' be an arbitrary point in $P - P^*_{1..\beta}$. Note that $\|p^{\beta+1}, U\|_{max} \leq \|p', U\|_{max}$. Combining this with equation (9), we derive the following:

$$\max_{u_i \in U}(\|p^o, u_i\|) + 2\lambda_\beta \leq \max_{u_i \in U}(\|p', u_i\|). \tag{10}$$

From the given condition $\|l_i, u_i\| \leq \lambda_\beta$, we can obtain: $\max_{u_i \in U}(\|p^o, u_i\|) \geq \max_{u_i \in U}(\|p^o, l_i\|) - \lambda_\beta$ and $\max_{u_i \in U}(\|p', u_i\|) \leq \max_{u_i \in U}(\|p', l_i\|) + \lambda_\beta$. Combining these two inequalities with equation (10), we get:

$$\max_{u_i \in U}(\|p^o, l_i\|) \leq \max_{u_i \in U}(\|p', l_i\|).$$

Thus, the MAX-GNN of L cannot be p'. □

We are now ready to present our buffering method. Specifically, before computing safe regions, we first retrieve the best $\beta + 1$ MAX-GNN of U. When we verify a tile s for user i (*Divide-Verify*, Algorithm 2), we only process s if $\|s, u_i\|_{max} \leq \lambda_\beta$. This guarantees that the condition $\|l_i, u_i\| \leq \lambda_\beta$ in Theorem 4 is always satisfied. Then, we use the point set $P^*_{1..\beta}$ (instead of the entire P) in the verification function. We need not access the R-tree again since we have retrieved $P^*_{1..\beta+1}$ (which contains $P^*_{1..\beta}$).

5.4.2 Reducing the Sensitivity of Parameter β

Observe that the parameter β exhibits a tradeoff between the verification cost and the extent of safe regions. A small β limits the extent of safe regions significantly (due to the distance threshold λ_β). To avoid overly small safe regions, we recommend to use a sufficiently large β. [5] However, the verification cost is directly proportional to β.

In the following, we provide an efficient implementation (Algorithm 5) whose verification cost is less sensitive to β. Now, we consider all β possible distance thresholds: $\lambda_1, \lambda_2, \ldots, \lambda_\beta$. To reduce the verification cost, we pick the smallest distance threshold λ_z such that it satisfies the condition of Theorem 4 for the current safe region group $\mathcal{R}$ and the new tile s. This can be implemented efficiently in $O(\log \beta)$ time by binary search (line 2). If such a distance threshold λ_z cannot be found, then the verification returns false as the new tile s violates the condition of Theorem 4.

Algorithm 5. Buffer-Divide-Verify (Safe region group $\mathcal{R}$, User u_i, Tile s, Optimal point p^o, Set $P^*_{1..\beta+1}$, Level L)

1: $dist \leftarrow \max\{\|u_i, s\|_{max}, \max_{R_j \in \mathcal{R}} \|u_j, R_j\|_{max}\}$
2: find the minimum slot z such that $dist \leq \lambda_z$ ▷ binary search
3: **if** no such z exists **then**
4: return false
5: $\forall\, p \in P^*_{1..z} - \{p^o\}$, *Tile-Verify* $(\mathcal{R}, u_i, s, p, p^o)$ is true **then**
6: $R_i \leftarrow R_i \cup \{s\}$
7: return true
8: apply lines 4-10 of Algorithm 2

6 Sum-Optimal Meeting Point

In this section, we study a problem variant for the sum-optimal meeting point, which aims to minimize the sum of distances traveled by users, rather than their meeting time. We call this problem as Sum-optimal Meeting Point Notification (Sum-MPN). We first provide a formal definition for this problem, and then present extensions of our solutions for this problem.

6.1 Problem Definition

We first provide the definitions for the sum distance and the sum-optimal meeting point.

5. We set $\beta = 100$ based on our experimental results.

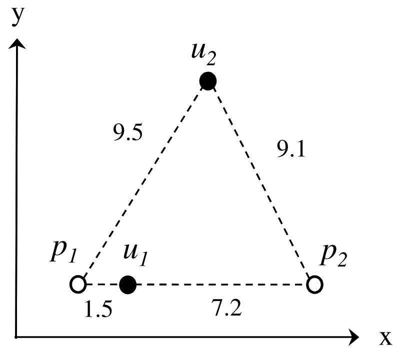


Fig. 11. Example for the sum-optimal meeting point.

Definition 7 (Sum distance). *The* sum distance *from a point* p *to a group of users* U *is*

$$\|p, U\|_{sum} = \sum_{u_i \in U} \|p, u_i\|.$$

Definition 8 (Sum-optimal meeting point). *Given a group of users* U *and a data set of points* P, *the* sum-optimal meeting point p^o *is the point in* P *with the smallest* $\|p^o, U\|_{sum}$. *It is also called SUM-GNN [21].*

The sum-optimal meeting point is more suitable when a group of users wishes to minimize the sum of their travel distances (and thus their total fuel cost). As for the incentive, the users in a group may agree on sharing the total fuel cost evenly when they reach the meeting point. Specifically, for those having fuel cost less than the average, they would contribute the cost difference (from the average) to other users in the group.

We illustrate an example of the sum-optimal meeting point in Fig. 11. Assume that the user group is $U = \{u_1, u_2\}$ and the data set is $P = \{p_1, p_2\}$. The sum-optimal meeting point is p_1 with the value $\|p_1, U\|_{sum} = 1.5 + 9.5 = 11$.

The definitions for independent safe region group and maximal safe region group (Definitions 3 and 4) are still applicable in the context of the sum-optimal meeting point. We proceed to extend our solutions to compute a safe region group for the sum-optimal meeting point.

Observe that Papadias et al. [21] have studied the snapshot version of our problem, i.e., computing the sum-optimal meeting point (called SUM-GNN in their work). In contrast, we focus on computing safe regions for such a meeting point.

6.2 Circular Safe Region Approach

Algorithm 1 can be easily adapted to compute a safe region group for the sum-optimal meeting point. At line 1, we call the "FindSumGNN" algorithm in [24]. At line 2, we compute the value of r_{max} by Equation (11). Its correctness is guaranteed by the following theorem.

Theorem 5 (Sum-optimal maximal circles). *The maximum radius of circles for safe regions is*

$$r_{max} = \frac{\min_{p \in P - \{p^o\}}(\|p, U\|_{sum}) - \|p^o, U\|_{sum}}{2m}. \tag{11}$$

Proof. Let $R_i = \odot(u_i, r)$, a circle with radius r and center as the current user location u_i. We have: $\|p, R_i\|_{max} = \|p, u_i\| + r$ and $\|p, R_i\|_{min} = \|p, u_i\| - r$.

By applying the definition of safe regions for Sum-optimal meeting point, we derive the following inequality for any point $p \in P - \{p^o\}$:

$$\sum_{u_i \in U}(\|p^o, R_i\|_{max}) \leq \sum_{u_j \in U}(\|p, R_j\|_{min}),$$

$$\sum_{u_i \in U}(\|p^o, u_i\| + r) \leq \sum_{u_j \in U}(\|p, u_j\| - r).$$

By rearranging the terms, we obtain

$$2m \cdot r \leq \sum_{u_j \in U}(\|p, u_j\|) - \sum_{u_i \in U}(\|p^o, u_i\|),$$

which is equivalent to

$$r \leq \frac{\|p, U\|_{sum} - \|p^o, U\|_{sum}}{2m}. \tag{12}$$

Note that Equation (12) holds for any point $p \in P - \{p^o\}$. By taking the minimum value of all $\|p, U\|_{sum}$, we obtain: $r_{max} = \frac{\min_{p \in P - \{p^o\}}(\|p, U\|_{sum}) - \|p^o, U\|_{sum}}{2m}$. □

6.3 Tile-Based Safe Region Approach

Algorithm 3 can be applied to compute a safe region group for the sum-optimal meeting point. Also, we adopt the divide-and-conquer method (Algorithm 2) to check whether a tile s should be inserted into the safe region R_i of user u_i. It remains to discuss how to extend the optimizations in Sections 5.3 and 5.4 for the sum-optimal meeting point.

6.3.1 Group Tile Verification

Let $\langle R_i|_{i=1}^m \rangle$ be a valid safe region group obtained so far. Given a new tile s for user u_x, we want to verify efficiently whether the above safe region group is valid after inserting s into R_x. Let $L = \langle l_1, \ldots, l_m \rangle$ be a group location instance, where $l_x \in s$ and $l_i \in R_i$ for all $i \neq x$.

Specifically, we want to verify that, for every instance of users' locations L (as stated above), whether $\|p^o, L\|_{sum} \leq \|p', L\|_{sum}$ holds for every non-result point $p' \in P - \{p^o\}$. We define the comparison function $F(p', p^o, L)$ as

$$\begin{aligned} F(p', p^o, L) &= \|p', L\|_{sum} - \|p^o, L\|_{sum} \\ &= \sum_{l_i \in L}(\|p', l_i\| - \|p^o, l_i\|). \end{aligned} \tag{13}$$

The verification returns false if $F(p', p^o, L) < 0$ for some non-result point $p' \in P - \{p^o\}$ and some group location instance L.

For a given point $p' \in P - \{p^o\}$, we minimize the value of $F(p', p^o, L)$ in order to check whether it can become negative. Observe that, in Equation (13), we can minimize the term $\|p', l_i\| - \|p^o, l_i\|$ for each user u_i independently.

It turns out that the loci of $\|p', l\| - \|p^o, l\| = r$ can be described by hyperbola curves, as shown in Fig. 12. In this example, $p^o = (1, 0)$ and $p' = (-1, 0)$. Given a square tile s, our task is to find the minimum value of $\|p', l\| - \|p^o, l\|$ among all location l of s. First, we divide the space by the axis $\overline{p'p^o}$ into the upper half-plane and the lower

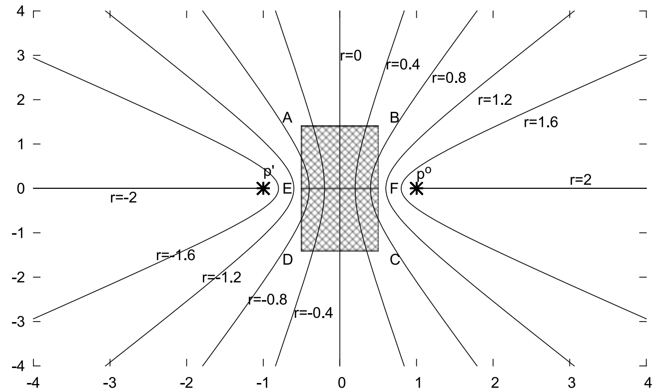


Fig. 12. Hyperbola curves for $\|p', l\| - \|p^o, l\| = r$.

half-plane. Observe that, within the same half-plane, the same hyperbola curve can be either a decreasing curve or an increasing curve, but not both. As such, the minimum value along a straight line must occur at either of its end vertices. To find the minimum value of a tile s, it suffices to compute the value $\|p', v\| - \|p^o, v\|$ at: i) each corner v of s (e.g., A,B,C,D), and ii) any intersection v between s and the axis $\overline{p'p^o}$ (e.g., E, F).

This verification function is summarized as Algorithm 6. Observe that there are redundant computations during different calls of the algorithm (lines 6-8). We can apply memorization techniques to avoid such redundant computations. The idea is to employ m hash tables: $H_1, H_2, \ldots, H_m$. For each user u_i, the minimum F_i value for point p' can be maintained at the hash entry $H_i(p')$. Then, we make two changes to the algorithm:

- replace lines 6–8 by the statement: $F_i \leftarrow H_i(p')$
- at line 12, we also execute $H_x(p') \leftarrow \min\{F_x, H_x(p')\}$ because the tile s will be inserted into the safe region of user u_x

Algorithm 6. Sum-GT-Verify(Safe region group $\mathcal{R}$, User u_x, Tile s, Point p, Optimal point p^o)

1: $F_x \leftarrow \infty$
2: **for** each vertex or intersection v of tile s **do**
3: $\quad F_x \leftarrow \min\{F_x, \|p', v\| - \|p^o, v\|\}$
4: **for** each user u_i except u_x **do**
5: $\quad F_i \leftarrow \infty$
6: $\quad$ **for** each tile s_i of safe region R_i **do**
7: $\quad\quad$ **for** each vertex or intersection v of tile s_i **do**
8: $\quad\quad\quad F_i \leftarrow \min\{F_i, \|p', v\| - \|p^o, v\|\}$
9: **if** $\sum_{i=1..m} F_i < 0$ **then**
10: $\quad$ return false
11: **else**
12: $\quad$ return true

6.3.2 Index Pruning

Since it is expensive to invoke the above verification function for every point $p \in P - \{p^o\}$, we derive the following theorem to detect unpromising points that cannot become candidates.

Theorem 6. *Given a safe region group $\mathcal{R}$, a point p cannot yield better result than p^o if,*

$$\|p, U\|_{sum} > \|p^o, U\|_{sum} + 2 \cdot \sum_{u_i \in U} r_i^{\dagger}, \tag{14}$$

where $r_i^{\dagger}$ is the maximum distance between user u_i's current location and its safe region boundary.

The above pruning technique can also be extended to the MBRs in the R-tree. For instance, a MBR can be pruned if the value $\sum_{u_i \in U} d_{min}(MBR, u_i)$ is larger than the right-side of Equation (14).

6.3.3 Buffering Optimization for Index Access

The same buffering technique in Section 5.4 can also be applied here, except that the distance threshold λ_β is now obtained from Equation (15) in the following theorem. The proof is similar to that of Theorem 4 and it is omitted due to lack of space.

Theorem 7 (Sum-optimal buffering condition). *Without loss of generality, assume that point p^j is the jth SUM-GNN of U, and the set $P^*_{1..j}$ contains the best j SUM-GNNs. Given a parameter β, we define the distance threshold λ_β as follows:*

$$\lambda_\beta = \frac{\|p^{\beta+1}, U\|_{sum} - \|p^o, U\|_{sum}}{2m}. \tag{15}$$

Given an instance of users' locations $L = \langle l_1, \ldots, l_m \rangle$, if $\|l_i, u_i\| \leq \lambda_\beta$ holds for every $1 \leq i \leq m$, then the SUM-GNN of L cannot be any point in $P - P^*_{1..\beta}$.

7 EXPERIMENTS

7.1 Settings

In this section, we experimentally evaluate the performance of our proposed techniques. All methods were implemented in C++ and the experiments were performed on an Intel Core2Duo 2.66 GHz CPU machine with 8 GB memory, running on Ubuntu 10.04.

Data set and query workload. We obtain a real data set from *www.pocketgpsworld.com*, which consists of $N = 21{,}287$ POIs. We simulate the movement of query users by using both synthetic and real trajectories: i) *GeoLife*, a real trajectory set of taxi drivers released by Microsoft[6]; ii) *Oldenburg*, a synthetic trajectory set generated from Brinkhoff's generator [27]. Each trajectory set consists of 60 trajectories that have above 10,000 timestamps. We partition each trajectory set into 10 user groups and then report the average performance on these user groups.

Measures. We evaluate our performance in three aspects: i) *update frequency*, which reflects the frequency for users to issue update messages to the server, and ii) *average running time*, which is the computation time for safe regions per update. iii) *Communication cost (packet count)*, measures the number TCP packets for messages sent between the server and the clients. A packet contains at most $(576 - 40)/8 = 67$ (double-precision) values since the typical maximum transmission unit (MTU) over a network is 576 bytes and a packet has a 40-byte header.[7] To represent a shape, we use

6. www.microsoft.com.
7. http://tools.ietf.org/html/rfc879.

TABLE 2
Parameter Values in Experiments

Parameter	Default	Range
Data size n	N	$0.25N, 0.5N, 0.75N, 1.0N$
User group size m	3	$2, 3, 4, 5, 6$
User speed	V (speed limit)	$0.25V, 0.5V, 0.75V, 1.0V$
Tile limit α	30	/
Split level L	2	/

three values per a circle, three values per a square, and four values per a rectangle.

Configurations. We study our proposed solutions with different variations. *Circle* denotes the *Circle-MSR* method in Section 4. *Tile* denotes the *Tile-MSR* method in Section 5 using undirected ordering on tiles and *lossless compression* in [12]. *Tile-D* is a variant of *Tile* using directed ordering on tiles. Both *Tile* and *Tile-D* apply the *GT-Verify* function and index pruning technique. Table 2 presents the default values and ranges of parameters in our experiments.

Our proposed methods require two extra parameters: i) the tile limit α, and ii) the split limit L. In our preliminary work [12], we have investigated the performance of our methods with respect to these parameters. As the default setting in [12], we set $\alpha = 30$ and $L = 2$ as they achieve a good trade-off between the the running time and the update frequency.

7.2 Scalability Experiments (for MPN)

In this section, we compare the circle-based safe regions and the tile-based safe regions.

Effect of user group size m. We vary the group size m in experiments on both *Geolife* and *Oldenburg* (see Fig. 13). The update frequency of *Tile* is less than half of *Circle*. *Tile-D* reduces the update frequency further, since it applies the directed ordering and covers more tiles for future possible locations. Due to the lossless safe-region compression technique in [12], our methods require only a few packets per sending a tile-based safe region. Thus, our methods still incur lower communication cost than *Circle*, as shown in Figs. 13c and 13d. As expected, the running time grows with m in Figs. 13e and 13f. *Circle* is efficient to compute but has a larger update frequency than tile-based safe regions; our tile-based safe regions are much more effective in optimizing the update frequency. Among these methods, *Tile-D* is the best in terms of update frequency.

Effect of data size n. We vary the data size (i.e., the number of *POIs*) in Fig. 14. As depicted in both data sets, the update frequencies of the methods increases because more *POIs* become as the candidates for the optimal points. Besides, *Circle* has a larger increase than those of methods based on the tile-based safe regions. Note that the communication costs of the methods are proportional to their corresponding update frequencies.

Effect of user speed. We proceed to vary the speed of users in this experiment. Recall that previous experiments use trajectories with 10,000 timestamps traveling at the speed limit V. To ensure consistent trajectories, when we generate trajectories for the speed $x \cdot V$, we pick the trajectory segments under the first x fraction of timestamps and then

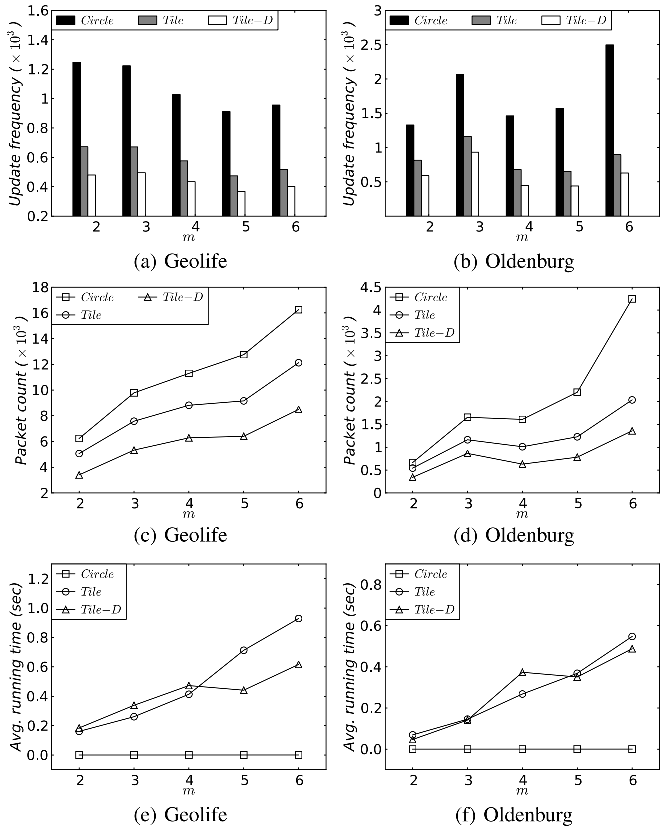


Fig. 13. Vary group size m.

sample 10,000 locations uniformly on those segments. Fig. 15 shows the update frequency and the communication cost of the methods with respect to the speed $(x \cdot V)$. Intuitively, as users move faster, they escape their safe regions quickly. Thus, all the methods have a large update frequency and communication cost at a high speed.

Effect of buffering parameter β. We proceed to study the effectiveness of the buffering optimization technique (see Section 5.4). Since *Tile-D* outperforms *Tile*, we do not include *Tile* in this experiment. *Tile-D-β* denotes the version

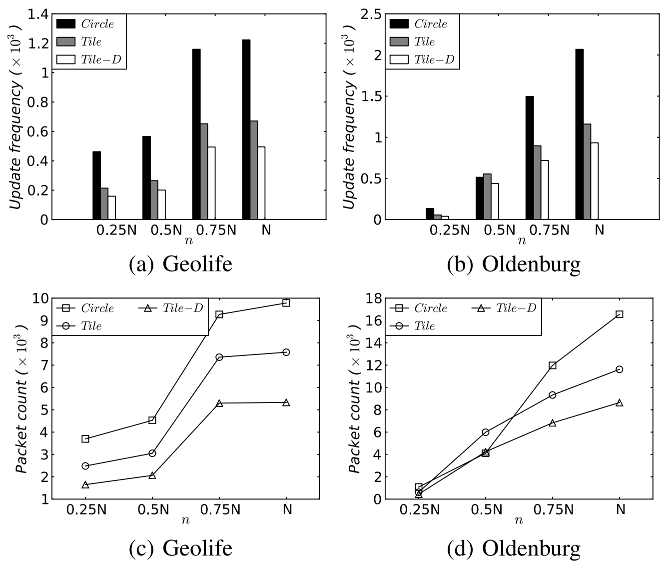


Fig. 14. Vary POI number n, as a fraction of data size N.

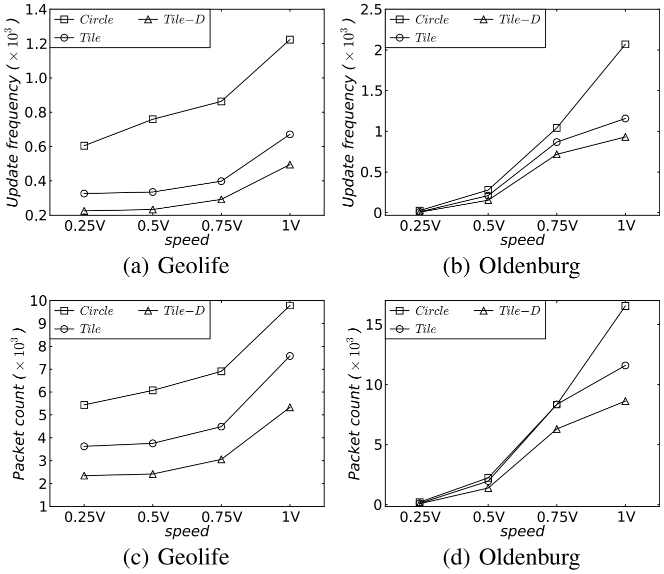


Fig. 15. Vary *speed*, as a fraction of speed limit V.

of *Tile-D* using the buffering optimization, which requires the parameter β. Fig. 16 plots the performance of the methods as a function of β. The CPU time of *Tile-D-*β is lower than that of *Tile-D* by an order of magnitude. This is because *Tile-D-*β avoids multiple accesses on the object R-tree. Recall that β determines a distance threshold (in Definition 6) that limits the extent of safe regions. When β increases, *Tile-D-*β obtains larger safe regions and thus its update frequency drops. Furthermore, its update frequency converges fast to that of *Tile-D*. We conclude that it is not hard to tune the

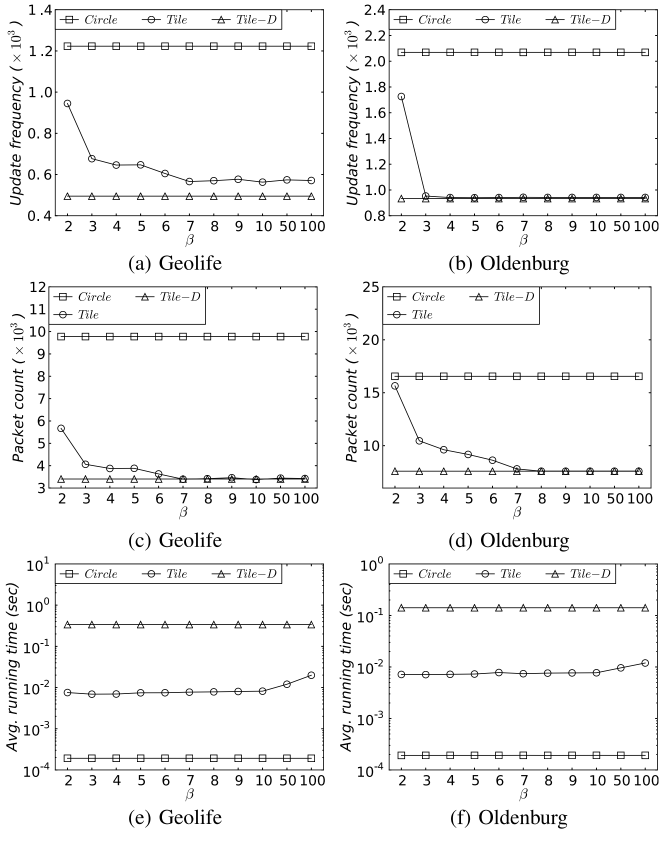


Fig. 16. Vary buffering parameter β.

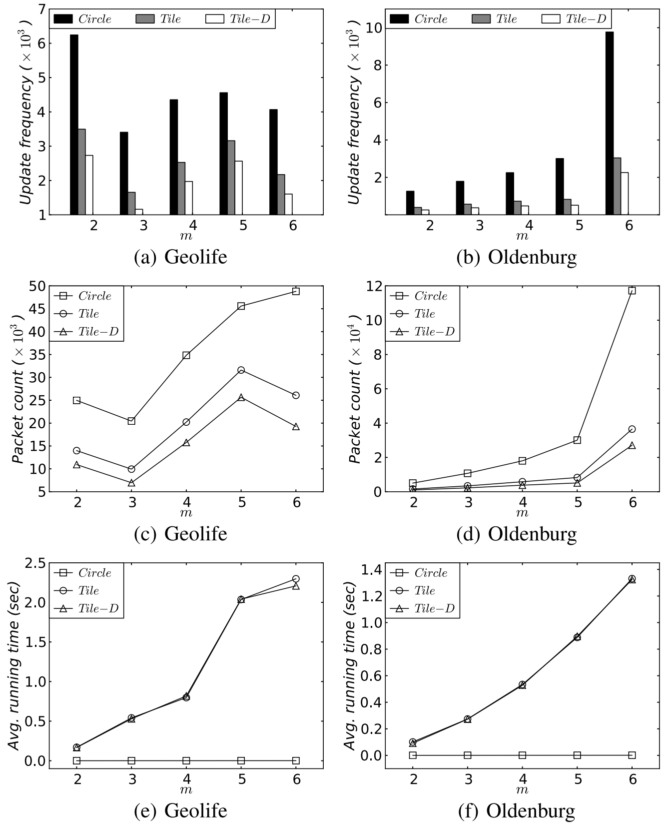


Fig. 17. Vary group size m (for Sum-MPN).

parameter β. In general, it is safe to set β to any value between 10 and 100.

7.3 Scalability Experiments (for Sum-MPN)

This section studies the scalability of our methods for the Sum-MPN problem.

Effect of user group size m. We vary the group size m in experiments on both data sets in Fig. 17. The trend is similar to that in corresponding experiments in the previous subsection. Again, tile-based safe region methods are

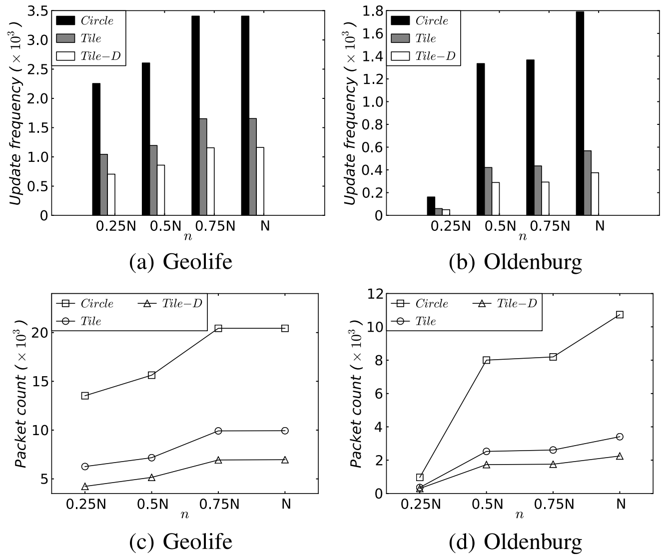


Fig. 18. Vary POI number n (for Sum-MPN), as a fraction of data size N.

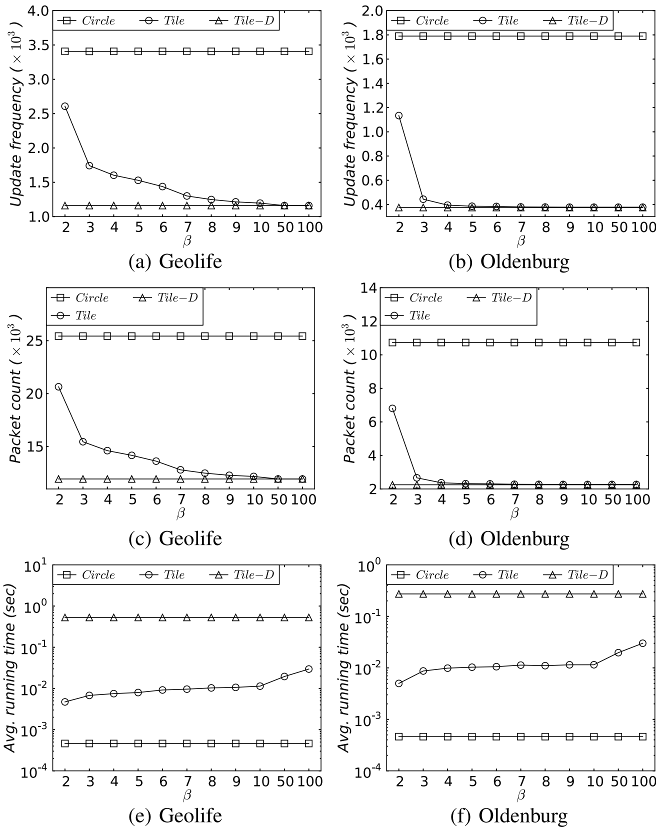


Fig. 19. Vary buffering parameter β (for Sum-MPN).

effective in optimizing the update frequency and the communication cost.

Effect of data size n. Next, we vary the data size (i.e., the number of *POI*s) in Fig. 18. When n is large, the data density in the space is high so all the methods have high update frequency. Nevertheless, the update frequency and the communication cost of tile-based methods increase at a slower rate than the circle-based method.

Effect of buffering parameter β. Fig. 19 shows the performance of *Tile-D* and *Tile-D-β*. Again, the trend is similar to that in corresponding experiments in the previous subsection. *Tile-D-β* achieves a much smaller CPU time, while its update frequency stays close to *Tile-D* for a wide range of β values. Thus, it is safe to tune the parameter β to any value between 10 and 100.

7.4 Summary of Experimental Results

Circle has the lowest running time, but it incurs higher update frequency and communication cost (packet count) than our tile-based methods.

Tile-D achieves the best update frequency and communication cost. Furthermore, our buffering optimization offers a substantial saving in the running time while only slightly increases the update frequency.

8 Conclusion

In this paper, we focus on minimizing the communication cost for monitoring the optimal meeting point for a group of users. We propose the concept of *independent safe region group*, in order to reduce the communication frequency of users. We design efficient algorithms and various optimizations to compute these safe regions. Also, we have studied a problem variant of the optimal meeting point based on the sum of distances.

In future, we plan to extend our techniques to the road network space. For *Circle*, we may replace a circular region by a range search region over road segments. For *Tile*, we may replace recursive tiles by recursive partitions of the road network. Also, we will develop a cost model for estimating the update frequency, the communication cost, and the running time of our methods.

Acknowledgments

The work of Nikos Mamoulis was supported by grant HKU 714712E from Hong Kong RGC. The work of Man Lung Yiu was supported by grant PolyU 5302/12E from Hong Kong RGC.

References

[1] E. Sarigöl, O. Riva, P. Stuedi, and G. Alonso, "Enabling social networking in ad hoc networks of mobile phones," *Proc. VLDB Endowment*, vol. 2, no. 2, pp. 1634–1637, 2009.

[2] N. Gupta, A. J. Demers, and J. Gehrke, "Semmo: A scalable engine for massively multiplayer online games," in *Proc. ACM SIGMOD Int. Conf. Management Data*, 2008, pp. 1235–1238.

[3] A. J. Demers, J. Gehrke, C. Koch, B. Sowell, and W. M. White, "Database research in computer games," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2009, pp. 1011–1014.

[4] C. Lampe, N. B. Ellison, and C. Steinfield, "A face(book) in the crowd: Social searching versus social browsing," in *Proc. 20th Anniv. Conf. Comput. Supported Cooperative Work*, 2006, pp. 167–170.

[5] H. Hu, J. Xu, and D. L. Lee, "A generic framework for monitoring continuous spatial queries over moving objects," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2005, pp. 479–490.

[6] K. Mouratidis, D. Papadias, S. Bakiras, and Y. Tao, "A threshold-based algorithm for continuous monitoring of k nearest neighbors," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 11, pp. 1451–1464, Nov. 2005.

[7] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik, "The v*-diagram: A query-dependent approach to moving knn queries," *Proc. VLDB Endowment*, vol. 1, no. 1, 2008, pp. 1096–1106.

[8] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee, "Location-based spatial queries," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2003, pp. 443–454.

[9] M. L. Yiu, L. H. U, S. Saltenis, and K. Tzoumas, "Efficient proximity detection among mobile users via self-tuning policies," *Proc. VLDB Endowment*, vol. 3, no. 1, 2010, pp. 985–996.

[10] L. Qin, J. X. Yu, B. Ding, and Y. Ishikawa, "Monitoring aggregate k-nn objects in road networks," in *Proc. 20th Int. Conf. Sci. Statist. Database Manage.*, 2008, pp. 168–186.

[11] Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search," in *Proc. 28th Int. Conf. Very Large Data Bases*, 2002, pp. 287–298.

[12] J. Li, M. L. Yiu, and N. Mamoulis, "Efficient notification of meeting points for moving groups via independent safe regions," in *Proc. IEEE Int. Conf. Data Eng.*, 2013, pp. 422–433.

[13] G. S. Iwerks, H. Samet, and K. P. Smith, "Continuous k-nearest neighbor queries for continuously moving points with updates," in *Proc. 28th Int. Conf. Very Large Data Bases*, 2003, pp. 512–523.

[14] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias, "Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2005, pp. 634–645.

[15] X. Yu, K. Q. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in *Proc. IEEE Int. Conf. Data Eng.*, 2005, pp. 631–642.

[16] X. Xiong, M. F. Mokbel, and W. G. Aref, "Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases," in *Proc. IEEE Int. Conf. Data Eng.*, 2005, pp. 643–654.

[17] M. A. Cheema, L. Brankovic, X. Lin, W. Zhang, and W. Wang, "Multi-guarded safe zone: An effective technique to monitor moving circular range queries," in *Proc. IEEE Int. Conf. Data Eng.*, 2010, pp. 189–200.
[18] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee, "Location-based spatial queries," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2003, pp. 443–454.
[19] Z. Xu and H.-A. Jacobsen, "Adaptive location constraint processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2007, pp. 581–592.
[20] B. Zheng and D. L. Lee, "Semantic caching in location-dependent query processing," in *Proc. 7th Int. Symp. Adv. Spatial Temporal Databases*, 2001, pp. 97–116.
[21] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui, "Aggregate nearest neighbor queries in spatial databases," *Trans. Database Syst.*, vol. 30, no. 2, pp. 529–576, 2005.
[22] F. Li, B. Yao, and P. Kumar, "Group enclosing queries," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 10, pp. 1526–1540, Oct., 2011.
[23] H. G. Elmongui, M. F. Mokbel, and W. G. Aref, "Continuous aggregate nearest neighbor queries," *GeoInformatica*, vol. 17, pp. 63–95, 2011.
[24] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *Proc. IEEE Int. Conf. Data Eng.*, 2004, pp. 301–312.
[25] T. Emrich, H.-P. Kriegel, P. Kröger, M. Renz, and A. Züfle, "Boosting spatial pruning: On optimal pruning of mbrs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 39–50.
[26] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu, "Prediction and indexing of moving objects with unknown motion patterns," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2004, pp. 611–622.
[27] T. Brinkhoff, "A framework for generating network-based moving objects," *GeoInformatica*, vol. 6, no. 2, pp. 153–180, 2002.

Jing Li received the bachelor's degree in computer science and engineering from Nanjing University, Nanjing, China, in 2008, and the PhD degree in computer science from the University of Hong Kong, Hong Kong, in 2012. He is currently at Morgan Stanley.

Jeppe Rishede Thomsen received the master's degree in computer science from Aalborg University, Aalborg, Denmark, in 2010. He is currently working toward the PhD degree in Hong Kong Polytechnic University, Hong Kong, under the supervision of Dr. Man Lung Yiu.

Man Lung Yiu received the bachelor's degree in computer engineering and the PhD degree in computer science from the University of Hong Kong, Hong Kong, in 2002 and 2006, respectively. Prior to his current post, he was at Aalborg University, Aalborg, Denmark, for three years starting in the Fall of 2006. He is currently an assistant professor in the Department of Computing, Hong Kong Polytechnic University, Hong Kong. His current research interests include management of complex data, in particular, query processing topics on spatiotemporal data and multidimensional data.

Nikos Mamoulis is currently a professor in the Department of Computer Science, University of Hong Kong, Hong Kong, which he joined in 2001. His research interests include management and mining of complex data types, including spatial, spatiotemporal, object-relational, multimedia, text and semistructured data. He has served on the program committees of over 90 international conferences and workshops on data management and data mining. He was the general chair of Statistical Database Management Conference 2008 and the PC chair of the Symposium on Advances in Spatial Temporal Databases 2009. He is an associate editor for the *IEEE Transactions on Knowledge and Data Engineering* and the *Very Large Data Bases* Journal.