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

Jing Li #, Man Lung Yiu *, Nikos Mamoulis #

#*Department of Computer Science, The University of Hong Kong*
{jli, nikos}@cs.hku.hk

**Department of Computing, Hong Kong Polytechnic University*
csmlyiu@comp.polyu.edu.hk

***Abstract*— In applications like social networking services and online games, multiple moving users form a group and wish to be continuously notified with the best meeting point from their locations. To reduce the communication frequency of the application server, a promising technique is to apply 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 dependency among multiple safe regions. These unique characteristics render existing safe region methods that focus on a single safe region inapplicable to our problem. To tackle these challenges, we first examine the shapes of safe regions in our problem context and propose feasible approximations for them. We design efficient algorithms for computing these safe regions, as well as develop compression techniques for representing safe regions in a compact manner. Experiments with both real and synthetic data demonstrate the efficiency of our proposal in terms of computation and communication costs.**

I. Introduction

Recently, social networking services in the ad-hoc mobile environment have attracted lots of attention [1], [2]. 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, Efficient Notification of Meeting Points (*ENMP*) for multiple moving users: given a group of moving users U, a set of points of interest (*POI*) P, *ENMP* continuously reports the optimal meeting point $p^o \in P$ to users in U such that their maximum distance toward p^o is minimized. *ENMP* is motivated by many applications in social networks, location-based games and massively multi-player on-line games [3], [4].

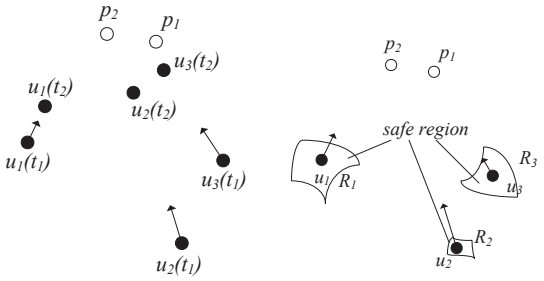
A real application relevant to *ENMP* is EchoEcho[2], invented by Google Venture. EchoEcho assists users to browse their friends' real-time locations and share their own. As a highlight feature, EchoEcho allows a user to continuously observe his/her friends' locations regarding to a predetermined meeting point. Mobile users with such interests have also been investigated in the collaborative system research [5].

Jing Li and Nikos Mamoulis were supported by grant HKU 715711E from Hong Kong RGC. Man Lung Yiu was supported by grant PolyU 5302/12E from Hong Kong RGC.

[1]www.foursquare.com

[2]www.echoecho.me

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 Figure 1(a) for illustration). Event calender initially recommends a restaurant, i.e., p_1, based on current location statuses of these users at time-stamp t_1. However, due to unpredictable traffic, the speeds of different users may change and thus the optimal meeting point may also change. In Figure 1(a), the locations of users change from $u_i(t_1)$ to $u_i(t_2)$. Due to traffic jams, user u_1 advances toward p_1 with low speed and reaches $u_1(t_2)$. Thus, at time stamp t_2, the according optimal meeting point changes to p_2. With the help of *ENMP*, such a change of the optimal meeting point can be quickly detected and thus, the subsequent events in the event calendar can be rearranged in advance.



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

Fig. 1. Motivation

Besides social networking services, ENMP also finds applications in location-based games. As a famous outdoor GPS game, Tourality[4] organizes games for teams with distributed players. To win a Tourality 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. By using *ENMP* in this scenario, it can dynamically adjust the first meeting spot based on real-time locations of players and thus

[3]apps.facebook.com

[4]www.tourality.com

shorten the meeting time. ENMP has similar applications in massively multi-player online (MMO) games.

Limitations of bandwidth and battery power raise challenges for mobile applications, including *ENMP*. Thus, the essential optimization goal for these applications is to minimize communication cost [6]–[10]. Such an optimization goal also reduces unnecessary computational workload in the server because the communication frequency between the clients and the server is alleviated. We consider communication cost in two aspects: (i) update frequency (communication frequency), which measures the frequency for users to issue update messages to the server; (ii) packet count [11], which is the total number of TCP packets sent between the server and the clients.

A straightforward solution is to force each client (i.e., user) to communicate with the server periodically (e.g., every second). Such a brute force solution incurs huge computation and communication costs at the server side. Therefore, it is important to develop an efficient solution that reduces the communication cost between the server and the users. Previous work [12], that considers similar applications under road networks, only develops several methods for reducing road network distance computations but none for the optimization of communication cost. Thus, their solutions are inapplicable to our problem.

Motivated by this, we propose novel solutions based on *safe region* technique. *Safe regions* are a set of geographical regions such that if each user stays inside his/her own, the query result will remain the same, thus avoiding communication between users and the server. For instance, in Figure 1(b) the optimal meeting is p_1 if all users stay in their own safe regions (R_1, R_2, R_3). The usage of safe regions for multiple users raises several challenges. Firstly, existing safe region computation techniques apply to only one user while our work needs to compute safe regions for multiple users. Secondly, the safe regions have irregular shapes (to be demonstrated in Section III-B), unlike previous work that have a simple-shaped safe region (e.g., Voronoi cell [13]). Thirdly, it is infeasible to pre-compute the safe regions for multiple users because multiple safe regions depend on the locations of moving users, which are unpredictable.

In this paper, we first propose circular safe regions that represent each safe region as a circle and are easy to compute. To further reduce communication frequency (i), we propose tile-based safe regions that utilize squares (i.e., tiles) to assemble large regions and approximate maximal safe regions better. We utilize a novel compression technique to alleviate representative bits incurred by irregular shape of safe regions and thus minimize packet cost (ii).

Contributions:

- We propose a new problem *ENMP* with broad applications and aim at minimizing the *communication cost*.
- We design circular safe regions which are efficient to compute.
- We develop tile-based safe regions that focus on minimizing communication cost.
- We discuss several optimizations to enhance the efficiency of the computation.
- We introduce novel compression techniques for representing irregular safe regions in order to reduce their communication frequencies.
- We demonstrate effectiveness and efficiency of our methods by extensive experiments.

The paper is organized as follows. First, we review the related work in Section II. Then, we introduce our notations and define the problem formally in Section III. Next, we present our solutions in Section IV and Section V, together with their optimizations. We evaluate our methods on real and synthetic data in Section VI. Finally, we conclude our paper in Section VII.

II. Related Work

Previous work on moving query processing can be classified into two categories: (i) report query results to a single user continuously, e.g. kNN [6], [13]–[18], circular range queries [19], moving window(rectangle range) queries [6], [20]; (ii) detect relationships among moving objects, e.g., proximity detection [1], [10], [21], [22], constraints monitoring [23].

The safe region has been widely used in moving query processing. When the future movement of a user is unknown, the safe region helps alleviate the communication cost between clients and servers. When a user registers a continuous query, the server will return points of interest along with a safe region. Query result remains the same if the user stays inside current safe region. When 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; it is an order-k Voronoi cell for a kNN query [20] or a region formed by intersections and unions of circles for a range query [19]. All these methods are not applicable to our problem since: 1) multiple safe regions are irregular, which are hard to compute comparing with a single safe region; 2) multiple users change their locations dynamically and unpredictably and thus multiple safe regions cannot be pre-computed as Voronoi cell [24].

Proximity detection [10] helps a user to maintain a list of friends who are within a distance threshold of this user. Since both the user and his/her friends are moving, [10] proposes self-tuning policies to automatically assign an adjustable circle shape safe region for each user. However, their work does not consider POIs where the users are supposed to meet.

The snapshot version of our problem is equivalent to the group nearest neighbor query (GNN) proposed in [25]. It attempts to find a point of interest p that minimizes total distance between p and users' locations. Algorithms in [25] use an R-tree to index points of interest and filter MBRs that contain no better points than the current optimal point. They handle similar queries which minimize the maximum (or minimum) distance among a point of interest and users. *Group Enclosing Query* [26] is a specified GNN, which requires the maximum distance among a point of interest and users to be minimized. [26] indexes the user set U by using furthest

Voronoi diagrams, reducing the distance computation between MBRs of points of interest and the user points. However, our problem emphasizes on response for continuous queries of this type and computing the safe regions to minimize communication cost.

The most related work is [12] which focuses on monitoring GNN in road networks. Their work is different from ours in two aspects: 1) our problem does not consider the road network; 2) their solutions are proposed to optimize the road network distance computation in the server side and thus are not applicable in our work. [27] continuously maintains a moving object from a set of moving objects, which is best according to its aggregate distance toward a set of selected *POI*s. In our problem, we retrieve a point of interest according to the locations of moving objects dynamically. Therefore, [27] is fundamentally different from our work.

III. Problem Setting

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

A. Preliminaries & 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 his location by u_i. Table I 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:

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

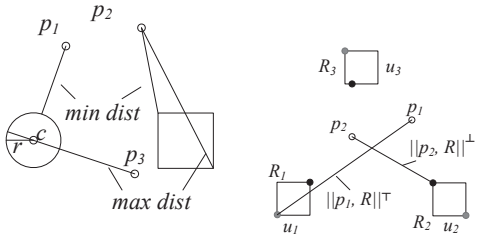
$$\|p, S\|_{max} = \max_{l \in S} \|p, l\| \quad (2)$$

Definition 2 (Optimal meeting point): Given a group of users U and a dataset of points P, the *optimal meeting point* p^o is a point in P that satisfies $\|p^o, U\|_{max} \leq \|p', U\|_{max}$ for any other point $p' \in P - \{p^o\}$.

Definition 3 (Independent safe regions): Let m be the number of users in U. A set of regions $\mathcal{R} = \{ R_i|_{i=1}^m \}$ is said to be a *set of independent safe regions* 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 regions): $\mathcal{R}^* = \{ R_i^*|_{i=1}^m \}$ is said to be a set of maximal safe regions if no other (independent) safe regions $\mathcal{R}' = \{ R_i'|_{i=1}^m \}$ satisfies: $\mathcal{R}' \neq \mathcal{R}^*$ and $R_i^* \subseteq R_i' \; \forall \, i = 1 \cdots m$.

As an example, Figure 2(a) 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 the circle, and from p_2 to the square). At time-stamp t_1 (t_2) in Figure 1(a), the optimal meeting point with the regarding locations of u_1-u_3 is p_1 (p_2). As shown in Figure 1(b), 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 and we will elaborate this issue shortly.



(a) max-min distance (b) dominant distance

Fig. 2. Distance

In this paper, we adopt the client-server architecture which is widely used in moving query processing [8], [19], [20]. Figure 3 illustrates this architecture. The server manages a dataset P of points-of-interest (e.g., restaurants, cafes) and indexes it by an R-tree. A group of users U wishes 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 his safe region R_i. Therefore, these safe regions serve to reduce the communication frequency (and computation effort) of the server significantly.

The system is triggered when a user $u_i \in U$ leaves his safe region R_i. Then, u_i sends his 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 a corresponding safe region R_i (Step 3). In summary, the server and users communicate via three types of messages. [5]

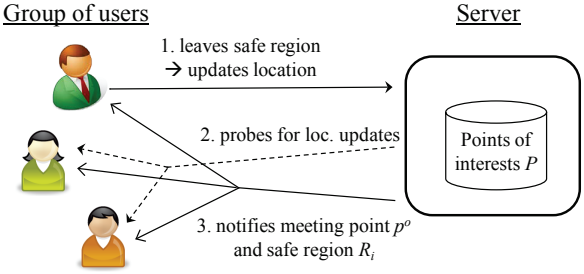


Fig. 3. System architecture

B. Characteristics of the Search Space and Safe Regions

This section describes the unique characteristics exhibited by the safe regions in our problem.

By Definition 3, the combinations of safe regions indeed form a huge search space: $m \cdot d$ dimensional space, where m is the number of users and d is the number of spatial dimensions.

[5] An alternative approach is only to notify the user who moves out of its safe region. We found experimentally that it incurs a larger communication cost due to the moving-out user's relatively smaller new safe region which is restricted by the others' previous large safe regions.

TABLE I

NOTATION

Notation	Meaning
U	a group of users
u_i	a user or its location
P	points of interest
$\|p, u\|$	Euclidean dist. 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

For example, for two users ($m = 2$) and the planar space ($d = 2$), the search space becomes 4-dimensional.

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). Figure 4a shows the locations of two users u, v and three points-of-interest a, b, c. Figure 4b illustrates the optimal meeting point for every combination of locations for user u, v. Each cell (at i-th column, j-th 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 3-rd column, 6-th row). For readability, the cells are colored based on their optimal meeting points (see Figure 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 independent safe regions $\{ R_i|_{i=1}^m \}$ for the users. First, two cells with the same color are not necessarily connected in the spatial domain. For instance, both combinations $\langle 3, 9\rangle$ 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 combinations of maximal safe regions obtained from the search space are not unique. For instance, consider two combinations of safe regions: (i) $\langle$2-4, 3-9$\rangle$, and (ii) $\langle$0-4, 5-9$\rangle$. Both combinations 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 [8], [19], [20] inapplicable to our problem.

Shapes of maximal safe regions. We proceed to illustrate the fact that the maximal safe regions in our problem have irregular shapes. Figure 5 shows an example in the 2-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.

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

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

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

Fig. 4. Optimal meeting point combinations, 1D example

We are unable to visualize the entire search space here as it has $m \cdot d = 2 \cdot 2 = 4$ dimensions. For the sake of illustration, we consider the special case that u_1 has a fixed location and then we attempt to find the maximal safe region of user u_2.

Let's examine how the point p_1 affects the safe region of u_2 (see Figure 5(a)). Specially, we 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).

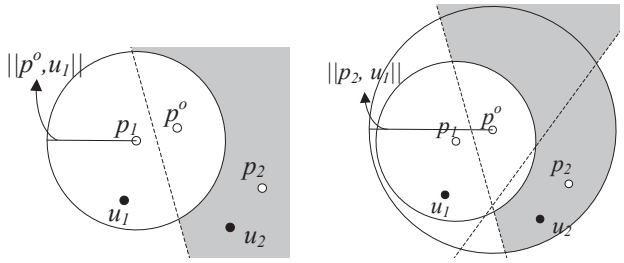
Following similar argument, we can derive the boundaries of the safe region of u_2 with respect to the point p_2. The maximal safe region of u_2 is restricted by both p_1 and p_2. Figure 5(b) shows that this region (in gray color) has an irregular shape.

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.

C. Objectives

As discussed above, maximal safe regions have irregular shapes and raise challenges in computation and representation. In subsequent sections, we will investigate some conservative approximations for maximal safe regions. Specifically, we will study circular safe regions in Section IV and tile-based safe regions in Section V. Our objectives are as follows:

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



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

Fig. 5. Safe Region Example

IV. Computing Circular Safe Regions

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.

A. 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. Thus, 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:

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

$$\|p, \mathcal{R}\|^\top = \max_{R_i \in \mathcal{R}} \|p, R_i\|_{max} \quad (4)$$

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 an lower-bound of $\|p, U\|^\dagger$, and $\|p, \mathcal{R}\|^\top$ serves as an upper-bound of $\|p, U\|^\dagger$. As an example in Figure 2(b), $\|p_2, \mathcal{R}\|^\perp$ 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 then 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 $p \neq p^o$

$$\|p^o, \mathcal{R}\|^\top \leq \|p, \mathcal{R}\|^\perp \quad (5)$$

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

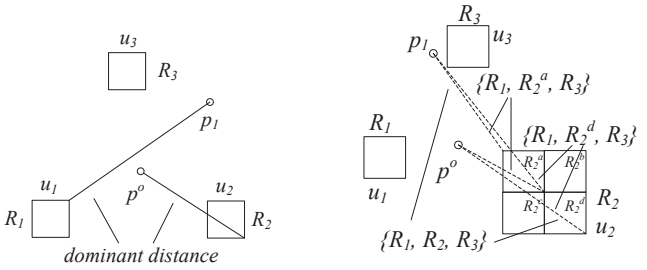
$$\|p, \mathcal{R}\|^\perp \leq \|p, \{l_i|_{i=1}^m\}\|^\dagger$$

Combining these two equations with Equation (5), we derive:

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

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

As an example, Figure 6(a) shows 2 data points and 3 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).



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

Fig. 6. Verifications of safe regions

B. 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:

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

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:

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

which is equivalent to

$$r \leq \frac{\|p, U\|_{max} - \|p^o, U\|_{max}}{2} \quad (7)$$

Note that Equation (7) must hold for any point $p \in P - \{p^o\}$. 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 dataset set P is indexed by an R-tree. First, the algorithm finds the best two meeting points by calling an existing algorithm [28] 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)

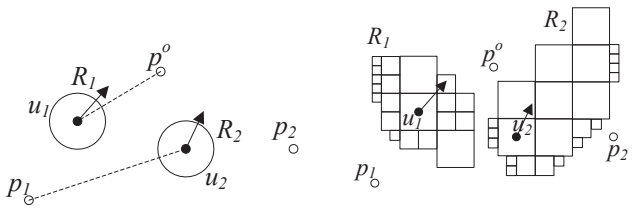
1: $p^o, p \leftarrow$ FindMaxGNN(U, P, 2) ▷ apply algo. in [28]
2: compute the radius r_{max} ▷ apply Equation (6)
3: **for** each user $u_i \in U$ **do**
4: return the safe region $\odot(u_i, r_{max})$ to u_i

Discussion. The advantage of circular safe regions is that they can be computed efficiently. However, they suffer from the drawback that they cannot serve as tight approximations of maximal safe regions. For instance Figure 7(a) 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 Figure 7(a). In the next section, we propose a tighter approximation of maximal safe regions, named *the tile-based safe regions*. As illustrated in Figure 7(b), the tile-based safe regions are much more tighter than the circular safe regions in Figure 7(a).

V. Computing Tile-based Safe Regions

In this section, we study a tighter approximation of maximal safe regions by using tiles. 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.

First, we develop 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 the algorithm. Finally, we develop techniques for compressing tile-based safe regions in order to reduce the number of packets communicated between clients and the server.



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

Fig. 7. Comparisons of safe regions

A. Divide-and-Conquer Verification for Tiles

First, we demonstrate that the verification condition in Lemma 1 is not tight. Then, we propose a divide-and-conquer method for verifying a tile precisely.

Figure 6(b) 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 set $\mathcal{R} = \{R_1, R_2, R_3\}$. As depicted in Figure 6(b), 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 Figure 6(b), then $\mathcal{R}$ can pass the verification. Consider the safe region set $\mathcal{R}' = \{R_1, R_2^a, R_3\}$ 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 set $\mathcal{R}'' = \{R_1, R_2^d, R_3\}$ passes the verification since $\|p^o, \mathcal{R}''\|_{max} \leq \|p_1, \mathcal{R}''\|_{min}$. After applying Lemma 1 to the remaining two groups of safe regions ($\{R_1, R_2^b, R_3\}, \{R_1, R_2^c, R_3\}$), 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 $\mathcal{R} = \{R_1, R_2, \cdots, R_i, \cdots, R_m\}$ is a valid safe region set (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 $R_1, \cdots, R_{i-1}, R_{i+1}, \cdots, R_m$ 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 V-C. 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 set $\mathcal{R}$, 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
```

B. 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 (as discussed in Figure 11 later). 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 tile size $\delta = \sqrt{2} \cdot r_{max}$ and add a tile $\square(u_i, \delta)$ 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.

We proceed to clarify two possible orderings for *NextTile* to select the next tile. In the example of Figure 8, the tiles are numbered by the their adding orders. 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 Figure 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.

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 )
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
```

Directed ordering. Existing studies [29] 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. Figure 8 shows an example of this directed ordering.

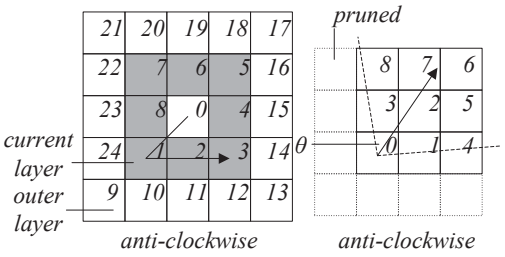


Fig. 8. Ordering for tiles

C. Efficiency Optimizations for Verifications

The running time of our Algorithm 3 is dominated by the 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 running time significantly.

We first study efficient implementations of the *Tile-Verify* function. Then, we propose a technique for pruning a large portion of points in $P - \{p^o\}$ without processing them one-by-one.

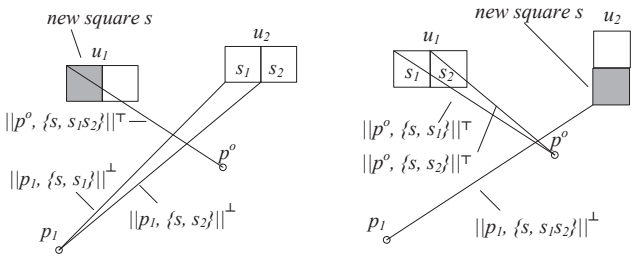
Individual Tile Verification (IT-Verify). This is a basic technique for verifying a new tile s to be allocated to user u_i. We are given that a valid safe region set $\{R_i|_{i=1}^m\}$ of all users. Let's consider a *tile combination* $\{s_1 \in R_1, \ldots, s_i = s, \ldots, s_m \in R_m\}$ contains a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

The goal of *IT-Verify* is to check all possible tile combinations as defined above. If any combination fails, then s is not valid as part of safe region of user u_i. We present the component *IT-Verify* in Algorithm 4. However, this method suffers from high computation cost due to the huge number of tile combinations formed by the safe regions of other users $u_j \neq u_i$. The number of such combinations is $O(\Pi_{i=1}^{m}|R_i|)$, where $|R_i|$ is the number of tiles in the safe region R_i.

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

1: **for** any $\mathcal{R}' = \{s_1 \in R_1, \ldots, s_i = s, \ldots, s_m \in R_m\}$ **do**
2: **if** *Verify*($\mathcal{R}', p^o, p$) = false **then**
3: return false
4: return true

Group Tile Verification (GT-Verify). This is an optimized verification method for the new tile s. Instead of testing each tile combination one-by-one, the main idea of *GT-Verify* is to group tile combinations and perform testings on these groups directly. This helps reduce the number of testings significantly.



(a) dominant min. dist. (b) dominant max. dist.

Fig. 9. Examples of GT-Verify

We illustrate two main types of grouping strategies. Figure 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 Figure 9(a), the maximum distance between the new tile and p^o ($\|p^o, s\|_{max}$) is the dominant max. distance for two tile combinations $\{s, s_1\}$ and $\{s, s_2\}$. $\{s, s_1\}$ ($\{s, s_2\}$) has the dominant min. distance to p_1 incident to s_1 (s_2). If $\{s, s_2\}$ fails in the verification test, so does $\{s, s_1\}$ since $\|p_1, \{s, s_1\}\|^{\perp} < \|p_1, \{s, s_2\}\|^{\perp} < \|p^o, s\|_{max}$. Thus, we can group s_1 and s_2 and test $\{s, s_1 \cup s_2\}$ instead of testing each combination individually. In Figure 9(b), the minimum distance between the new tile and p_1 ($\|p_1, s\|_{min}$) is the dominant min. distance for two tile combinations $\{s_1, s\}$ and $\{s_2, s\}$. $\{s_1, s\}$ ($\{s_2, s\}$) has the dominant max. distance to p^o incident to s_1 (s_2). If $\{s_2, s\}$ fails the verification test, so does $\{s_1, s\}$ since $\|p^o, \{s_1, s\}\|^{\top} > \|p^o, \{s_2, s\}\|^{\top} > \|p_1, s\|_{min}$. Thus, we can group s_1 and s_2 and test $\{s_1 \cup s_2, s\}$ instead of testing each combination individually.

The key observation is that we can categorize tile combinations 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.

$$G_j^{\downarrow\downarrow} = \{s' \in R_j \mid \|p^o, s'\|_{max} < d^o \wedge \|p, s'\|_{min} < d_p\} \quad (8)$$

$$G_j^{\uparrow\downarrow} = \{s' \in R_j \mid \|p^o, s'\|_{max} \geq d^o \wedge \|p, s'\|_{min} < d_p\} \quad (9)$$

$$G_j^{\downarrow\uparrow} = \{s' \in R_j \mid \|p^o, s'\|_{max} < d^o \wedge \|p, s'\|_{min} \geq d_p\} \quad (10)$$

$$G_j^{\uparrow\uparrow} = \{s' \in R_j \mid \|p^o, s'\|_{max} \geq d^o \wedge \|p, s'\|_{min} \geq d_p\} \quad (11)$$

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

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 combinations are valid, then the testing for the following safe region sets must be valid:

1) Safe region set $R' = \{G_1^{\downarrow\downarrow}, \ldots, \{s\}_i, G_m^{\downarrow\downarrow}\}$. u_i is $u_{p^o}^{\top}$ and also $u_p^{\perp}$.
2) Safe region set $R' = \{G_1^{\downarrow\downarrow} \cup G_1^{\uparrow\downarrow}, \ldots, \{s\}_i, \ldots, G_m^{\downarrow\downarrow} \cup G_m^{\uparrow\downarrow}\}$. u_i is $u_p^{\perp}$ and another user u_j ($u_i \neq u_j$) is $u_{p^o}^{\top}$.
3) Safe region set $R' = \{G_1^{\downarrow\downarrow} \cup G_1^{\downarrow\uparrow}, \ldots, \{s\}_i, \ldots, G_m^{\downarrow\downarrow} \cup G_m^{\downarrow\uparrow}\}$. 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 combinations R'' that are not covered in above safe region set are valid. Otherwise, test all these R'' by calling *Verify*(R'', p^o, p).

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

If 1) fails the verification, there exists a tile combination $\{s_1 \in G_j^{\downarrow\downarrow}, \ldots, s_i = s, \ldots, s_m \in G_m^{\downarrow\downarrow}\}$ 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 combination $\{s_1 \in G_1^{\downarrow\downarrow} \cup G_j^{\uparrow\downarrow}, \ldots, s_i = s, \ldots, s_j = s', \ldots, s_m \in G_m^{\downarrow\downarrow} \cup G_m^{\uparrow\downarrow}\}$ (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 combination $\{s_1 \in G_1^{\downarrow\downarrow} \cup G_1^{\downarrow\uparrow}, \ldots, s_i = s, \ldots, s_j = s', \ldots, s_m \in G_m^{\downarrow\downarrow} \cup G_m^{\downarrow\uparrow}\}$ (u_i as $u_{p^o}^{\top}$ and user u_j as $u_p^{\perp}$), which fails the verification.

For 4), all tile combinations 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 combination 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 combinations R'' by calling *Verify*(R'', p^o, p). ■

Based on the above theorem, we design the *GT-Verify* (Algorithm 5) 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 4-15, *GT-Verify*

behaves as described in Theorem 2 by calling *Verify*(R', p^o, p) on the grouped combination.

Algorithm 5 GT-Verify(Safe region set $\mathcal{R}$, 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 not valid then
5:     return false
6: if {G_1^↓↓ ∪ G_1^↑↓, ..., {s}_i, ..., G_m^↓↓ ∪ G_m^↑↓} is not valid then
7:     return false
8: if {G_1^↓↓ ∪ G_1^↓↑, ..., {s}_i, ..., G_m^↓↓ ∪ G_m^↓↑} is not valid then
9:     return false
10: if ∃s' ∈ R_i s.t. ||p^o, s'||_max ≤ d^o and ||p, s'||_min ≤ d_p then
11:     return true
12: for combination R'' not covered in above groups do
13:     if Verify(R'', p^o, p)=false then
14:         return false
15: 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 set $\mathcal{R}$, a point p cannot yield better dominant distance than p^o if for any $u_i \in U$,

$$\|p, u_i\| > \|p^o, \mathcal{R}\|^{\top} + r_i^{\dagger} \tag{12}$$

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

Proof: By Equation (12), we have

$$\begin{aligned} \|p, \mathcal{R}\|^{\perp} &= \max_{R_i \in \mathcal{R}} \|p, R_i\|_{min} && \text{by Equation (3)} \\ &> \max_{u_i \in U}(\|p, u_i\| - r_i^{\dagger}) \\ &> \max_{u_i \in U}(\|p^o, \mathcal{R}\|^{\top}) && \text{by Equation (12)} \\ &= \|p^o, \mathcal{R}\|^{\top} \end{aligned}$$

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

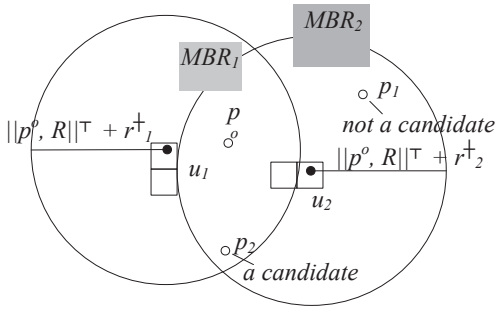


Fig. 10. Index Pruning

In order to retrieve the candidates from P, we traverse the R-tree (of P) while pruning unqualified candidates by the above theorem. For example, in Figure 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 overlaps the circle with the radius $\|p^o, \mathcal{R}\|^{\top} + r_1^{\dagger}$ and that with radius $\|p^o, \mathcal{R}\|^{\top} + r_2^{\dagger}$.

D. Compression of Safe Regions

A tile-based safe region R_i may contain a large number of tiles, thus incurring significant communication cost to report it to the user. For example, Figure 11a shows a region that contains 15 large tiles and 32 small tiles. In this section, we investigate techniques that compress the description of R_i in order to reduce its packets for communication.

Loseless compression. First, we propose a lossless compression technique that reduces the description of R_i while preserving the exact area covered by R_i. The idea is to combine adjacent tiles together to form a rectangle, as shown in Figure 11(b). Interestingly, by allowing overlaps among rectangles, we are able to reduce the total number of rectangles for describing R_i. In this example, R_i can be compressed into 7 rectangles. This lossless compression can be implemented by adapting a greedy algorithm for the set-cover problem.

Lossy compression. We then study a lossy compression technique that offers a trade-off between the the description of R_i and the area covered by R_i. While it is able to shrink the description of R_i aggressively, it may slightly reduce the area covered by R_i and increase the communication frequency.

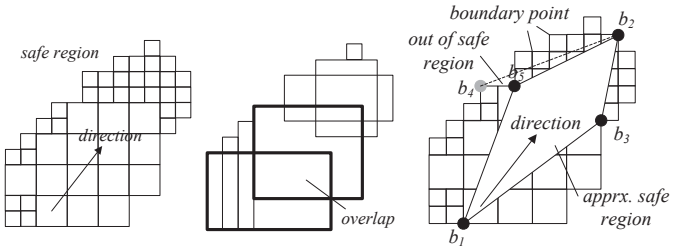
This compression technique offers the aforementioned trade-off via a parameter γ. It attempts to represent R_i by using a polygon formed by γ boundary points of R_i. Figure 11(c) illustrates an example with $\gamma = 4$ boundary points. We choose the first two boundary points based on the user's travel direction: b_1 and b_2. Next, we find a boundary point b'' such that (i) the polygon formed by b' and chosen boundary points is covered by R_i, and (ii) the area of such polygon is maximized. For example, in Figure 11(c), b_4 is not considered because the polygon formed by b_1, b_2, b_4 is not covered by the safe region. On the other hand, b_3 and b_5 are considered as potential boundary points. This procedure is repeated until γ boundary points are chosen. In this example, the region R_i is approximated by the polygon formed by the boundary points b_1, b_2, b_3, b_5.

VI. EXPERIMENTS

A. 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 2.66GHz CPU machine with 8 GBytes memory, running on Ubuntu 10.04.

Dataset and Query Workload. We obtain a real dataset from *www.pocketgpsworld.com*, which consists of $N = 21,287$ POIs. We simulate the movement of query users



(a) original R_i (b) loseless comp. (c) lossy comp.

Fig. 11. Compression for tile-based safe region

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 [30]. Each trajectory set consists of 60 trajectories that have above 10,000 timestamps. We partition each trajectory set into 10 groups and then report the average performance on these groups.

Measures. We evaluate our performance in three aspects: (i) *update frequency*, (ii) *packet count*, (iii) *average running time*. *Update frequency* reflects the frequency for users to issue update messages to the server. *Packet count* measures the number TCP packets for three types of messages between the server and the clients. A package 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 3 values per a circle, 3 values per a square, and 4 values per a rectangle. *Average running time* is the time to compute safe regions when a necessary notification is triggered.

Configurations. We study our proposed solutions with different variations. *Circle* represents *Circle-MSR* proposed in Section IV; *Tile* represents *Tile-MSR* proposed in Section V; it applies *undirected order* and *lossless compression*; *Tile-D* is an extension of *Tile* with *directed order*; *Tile*$^+$ (*Tile-D*$^+$) is an extension of *Tile* (*Tile-D*) with lossy compression. Except for *Circle* all our methods are equipped with *GT-test* and *index pruning*. The default values and ranges of parameters are presented in Table II.

TABLE II

PARAMETER VALUES IN EXPERIMENTS

Parameter	Default	Range
# of users m	3	$2, 3, 4, 5, 6$
# of POIs n	N	$0.25N, 0.5N, 0.75N, N$
Speed	V (speed limit)	$0.25V, 0.5V, 0.75V, V$
Tile limit α	30	$10, 20, 30, 40, 50$
# of boundary pt. γ	16	$4, 8, 16, 32, 64$
Split level L	2	$1, 2, 3, 4$

B. Effect of system parameters

We first investigate the effect of system parameters (α, γ, L) on the performance of our methods. We only present the experimental results on *Geolife* in this section as the experiments on *Oldenburg* have similar trends.

Effect of tile limit α. We present the experimental results by varying α in Figure 12. As α increases, i.e., more tiles are added into the safe regions, users obtain larger safe regions and thus all the tile-based methods incur fewer updates as depicted in Figure 12(a). *Tile-D* and *Tile-D*$^+$ approach stable *update frequency* when $\alpha \geq 30$. Figure 12(b) shows the packet count of our methods. *Tile* and *Tile-D*, which adopt *lossless compression*, increase the count slightly as α approaches 50 due to more tiles for compression. However, *Tile*$^+$ and *Tile-D*$^+$ decrease the packet count significantly as α increases to 30 since fewer packets are required to communicate and the *lossy compression* compresses a set of tiles by fixed number of boundary points (controlled by γ). On the other hand, more tiles are added into the safe regions, more running time is required to verify the safe regions. Thus, the running time increases as α becomes larger as shown in Figure 12(c). When $\alpha = 30$, the running time for all these methods is 0.3s.

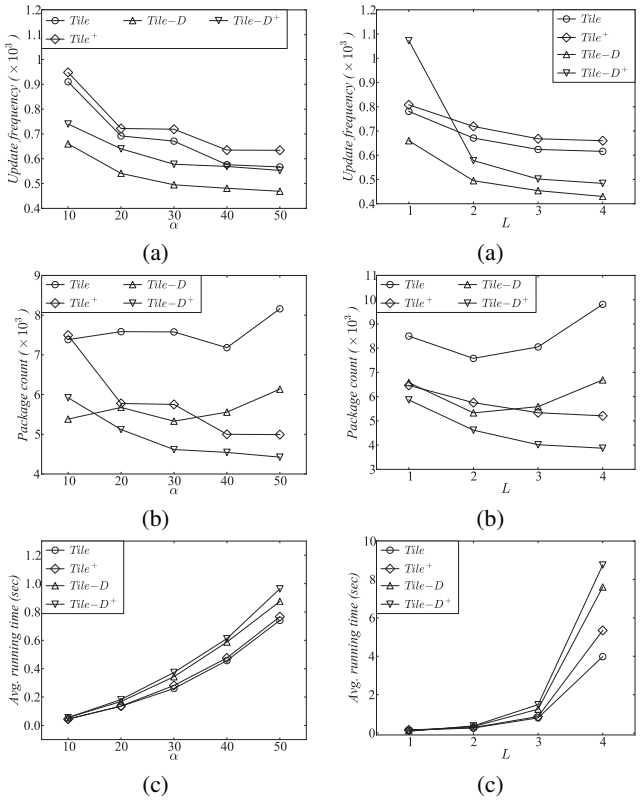


Fig. 12. Vary tile limit α

Fig. 13. Vary split limit L

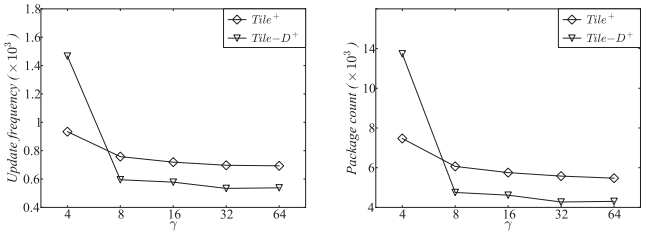
Effects of split limit L. We illustrate the results by varying the split limit L in Figure 13. As demonstrated in Figure 13(a), larger the split limit is, less the update frequency becomes because a larger split limit helps represent the irregular safe region better. *Tile-D*$^+$ benefits the most from $L = 1$ to $L = 2$. When L is larger than 2, the performance gap among these methods shrink. In Figure 13(b), both *Tile* and *Tile-D* decrease the packet count (at $L = 2$) because larger L provides good approximations of safe region and reduces the

[6]www.microsoft.com

[7]http://tools.ietf.org/html/rfc879

update frequency. When L is too large ($L > 2$) and many smaller tiles are added into the safe regions, the *packet count* increases for *Tile* and *Tile-D* because the *lossless compression* has to compress large amounts of tiles. However, *Tile*$^+$ and *Tile-D*$^+$ which utilize the *lossy compression*, always decreases the *packet count* since they use fixed number of boundary points to represent safe regions approximately. In terms of running time, a large split limit results in large amounts of smaller tiles and incurs many verification tests. As shown in Figure 13(c), the running time grows rapidly when $L > 3$.

Effects of boundary point number γ. We illustrate the results by varying γ in Figure 14. As γ changes, the running time varies slightly; thus we ignore the figure for the *running time*. As shown in Figure 14(a), more boundary points in the lossy compression approximate to the safe regions better. Specifically, when γ varies from 4 to 8, the *update frequency* for both *Tile*$^+$ and *Tile-D*$^+$ are reduced significantly. After $\gamma > 16$, the benefits become negligible. Interestingly, the similar trend happens to the *packet count* as shown in Figure 14(b). This is because: 1) the *update frequency* is reduced and thus less packets are required to communicate; 2) among the boundary points of the approximate safe regions, consecutive points that in a line are compressed into a line segment which results in a more compact representation.



(a) Geolife (b) Geolife

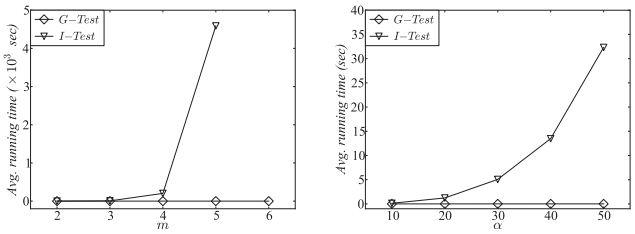
Fig. 14. Vary the number of boundary point γ

Effects of group testing. We demonstrate our optimizations for speeding up the verification of safe regions in Figure 15. The effectiveness of *index pruning* is not presented because without it all methods to compute tile-based safe region exam all the points and cause a long running time (can not finish within the limit time). As depicted in Figure 15(a), when the number of user m increases, the running time for *IT-Verify* grows especially when $m > 4$. Even when $m = 6$, *IT-Verify* can not finish within the limit time. However, *GT-Verify* only changes slightly. In Figure 15(b), we observe that the running time of *IT-Verify* increases exponentially as more tiles are added into the safe regions. *GT-Verify* only varies slightly comparing to *IT-Verify*. Thus, *GT-Test* is efficient.

C. Scalability experiments

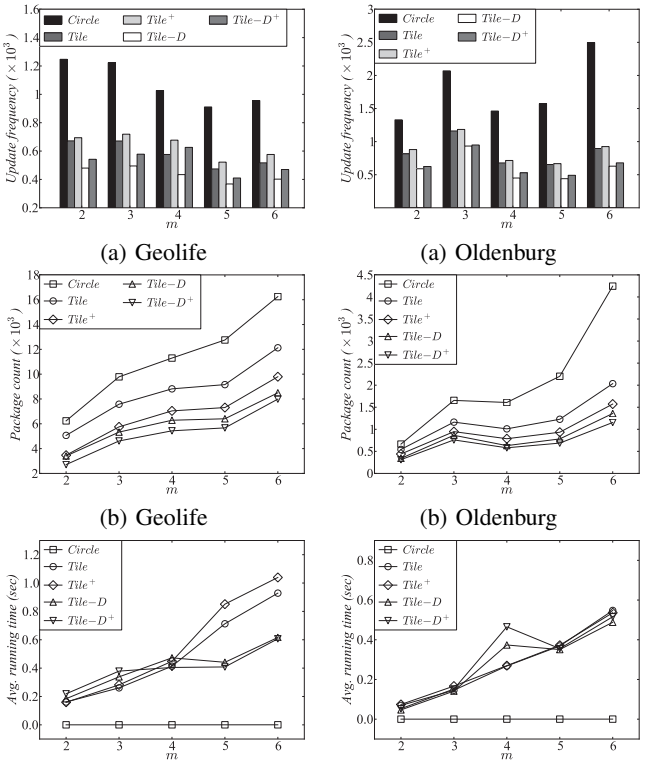
In this section, we compare the circle-based safe regions and the tile-based safe regions.

Effects of user number m. We provide experiments on both *Geolife* (in Figure 16) and *Oldenburg* (in Figure 17) by varying m. Since the safe regions depend on the relative positions of users, no clear trends for the *update frequency*



(a) Geolife (b) Geolife

Fig. 15. Validation evaluation time



(a) Geolife (a) Oldenburg

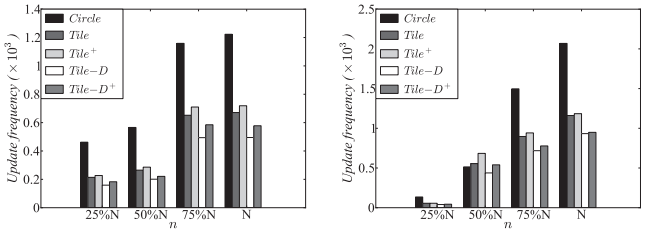
(b) Geolife (b) Oldenburg

(c) Geolife (c) Oldenburg

Fig. 16. Vary user number m Fig. 17. Vary user number m

exist as m varies as shown in Figure 16(a) and Figure 17(a). *Tile* and *Tile*$^+$ have less than half of the *update frequency* than *Circle* for all m. *Tile-D* and *Tile-D*$^+$ reduce the *update frequency* further, since they adopt the *directed ordering* and cover more future possible locations. Comparing with *Tile*(*Tile-D*), *Tile*$^+$(*Tile-D*$^+$) obtains a small increase in the *update frequency* because it utilizes the *lossy compression* and has approximate safe regions but its *packet count* is reduced accordingly as illustrated in Figure 16(b), 17(b). In terms of running time, due to the high complexity of computation of the tile-based safe regions, we observe an increase of the running time when m grows in both Figure 16(c), 17(c). *Circle* is efficient to compute but has a larger *update frequency* and a larger *packet count* than those of tile-based safe regions; thus, our tile-based safe regions are effective for optimizing the communication cost. Among these methods, *Tile-D* is the best in terms of *update frequency* and *Tile-D*$^+$ is the best in terms of *packet count*.

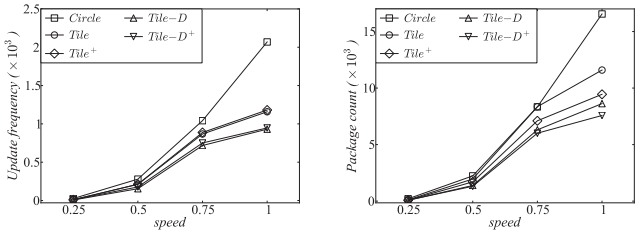
Effects of POI number n. We vary the number of *POI*s in Figure 18. As depicted in both data sets, all the methods increase the update frequency because more *POI*s exist as the candidates for the optimal points. Besides, *Circle* has a larger increase than those of methods based on the tile-based safe regions.



(a) Geolife (b) Oldenburg

Fig. 18. Vary POI number n

Effects of speed. We show the results by varying the speeds of users on the synthetic data set *Oldenburg* in Figure 19. To run this experiment, we need trajectories with similar path but different speeds. Thus, a sampling approach is adopted: given a trajectory with 10, 000 time stamps and speed V, we pick the locations under the first 5, 000 time stamps; from the trajectory segments connecting these 5,000 locations, we sample 10,000 locations uniformly. Thus, we obtain a trajectory with similar path but only half speed, i.e., $0.5V$. Similarly, we have trajectories with $0.25V$ and $0.75V$. Intuitively, as users move faster, they escape their safe regions quickly. All the methods have a larger update frequency (Figure 19(a)) and a larger packet count (Figure 19(b)) during the same time interval when their velocities are high.



(a) Oldenburg (b) Oldenburg

Fig. 19. Vary *speed*

VII. Conclusion

In this paper, we propose a novel problem, *efficient notification of meeting points*, aiming at minimizing the communication cost. We adopt *multi-user safe regions*, which is unique in our problem, to alleviate unnecessary communications. As discussed, such *multi-user safe regions* are difficult to compute since the inherent search space is high-dimensional. To conquer this difficulty, we propose efficient (including *GT-Verify* and *index pruning*) and effective (including *lossy compression* and *lossy compression*) algorithms, and evaluate them by extensive experiments. In the future work, we will investigate other aggregate functions [25] for defining the optimal meeting point.

References

[1] A. Efrat and A. Amir, "Buddy tracking - efficient proximity detection among mobile friends," in *INFOCOM*, 2004.

[2] E. Sarigöl, O. Riva, P. Stuedi, and G. Alonso, "Enabling social networking in ad hoc networks of mobile phones," *PVLDB*, vol. 2, no. 2, 2009.

[3] N. Gupta, A. J. Demers, and J. Gehrke, "Semmo: a scalable engine for massively multiplayer online games," in *SIGMOD*, 2008.

[4] A. J. Demers, J. Gehrke, C. Koch, B. Sowell, and W. M. White, "Database research in computer games," in *SIGMOD*, 2009.

[5] C. Lampe, N. B. Ellison, and C. Steinfield, "A face(book) in the crowd: social searching vs. social browsing," in *CSCW*, 2006.

[6] H. Hu, J. Xu, and D. L. Lee, "A generic framework for monitoring continuous spatial queries over moving objects," in *SIGMOD*, 2005.

[7] K. Mouratidis, D. Papadias, S. Bakiras, and Y. Tao, "A threshold-based algorithm for continuous monitoring of k nearest neighbors," *TKDE*, vol. 17, no. 11, 2005.

[8] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik, "The v*-diagram: a query-dependent approach to moving knn queries," *PVLDB*, vol. 1, no. 1, 2008.

[9] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee, "Location-based spatial queries," in *SIGMOD*, 2003.

[10] M. L. Yiu, L. H. U, S. Saltenis, and K. Tzoumas, "Efficient proximity detection among mobile users via self-tuning policies," *PVLDB*, vol. 3, no. 1, 2010.

[11] P. Engel, F. Gschwandtner, J. Martens, F. Fuchs, and G. Treu, "Optimizing position updates for knn - is it worth it?" in *IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, 2011.

[12] L. Qin, J. X. Yu, B. Ding, and Y. Ishikawa, "Monitoring aggregate k-nn objects in road networks," in *SSDBM*, 2008.

[13] Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search," in *VLDB*, 2002.

[14] G. S. Iwerks, H. Samet, and K. P. Smith, "Continuous k-nearest neighbor queries for continuously moving points with updates," in *VLDB*, 2003.

[15] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias, "Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring," in *SIGMOD Conference*, 2005.

[16] K. Mouratidis, D. Papadias, S. Bakiras, and Y. Tao, "A threshold-based algorithm for continuous monitoring of k nearest neighbors," *TKDE*, vol. 17, no. 11, 2005.

[17] X. Yu, K. Q. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in *ICDE*, 2005.

[18] X. Xiong, M. F. Mokbel, and W. G. Aref, "Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases," in *ICDE*, 2005.

[19] 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 *ICDE*, 2010.

[20] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee, "Location-based spatial queries," in *SIGMOD Conference*, 2003.

[21] A. Küpper and G. Treu, "Efficient proximity and separation detection among mobile targets for supporting location-based community services," *Mobile Computing and Communications Review*, vol. 10, no. 3, pp. 1–12, 2006.

[22] G. Treu, T. Wilder, and A. Küpper, "Efficient proximity detection among mobile targets with dead reckoning," in *MOBIWAC*, 2006.

[23] Z. Xu and H.-A. Jacobsen, "Adaptive location constraint processing," in *SIGMOD*, 2007, pp. 581–592.

[24] B. Zheng and D. L. Lee, "Semantic caching in location-dependent query processing," in *SSTD*, 2001.

[25] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui, "Aggregate nearest neighbor queries in spatial databases," *TODS*, vol. 30, no. 2, 2005.

[26] F. Li, B. Yao, and P. Kumar, "Group enclosing queries," *TKDE*, 2010.

[27] H. G. Elmongui, M. F. Mokbel, and W. G. Aref, "Continuous aggregate nearest neighbor queries," *GeoInformatica*, vol. 17, 2011.

[28] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries," in *ICDE*, 2004.

[29] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu, "Prediction and indexing of moving objects with unknown motion patterns," in *SIGMOD*, 2004.

[30] T. Brinkhoff, "A framework for generating network-based moving objects," *GeoInformatica*, vol. 6, no. 2, 2002.