Continuous Proximity Detection via Predictive Safe Region Construction

Ying Xu [*1], Dongxiang Zhang [#2], Meihui Zhang [†3], Dongsheng Li [*4] Xiaoling Wang [$5], Heng Tao Shen [#6]

[*] School of Computing, National University of Defense Technology, China
[#] Center for Future Media and School of Computer Science & Engineering, UESTC, China
[†] School of Computer Science and Technology, Beijing Institute of Technology, China
[$] Institute of Massive Computing, East China Normal University, China
[1] exwhy_up123@163.com [2] zhangdo@uestc.edu.cn [3] meihui_zhang@bit.edu.cn [4] lds1201@163.com
[5] xlwang@sei.ecnu.edu.cn [6] shenhengtao@hotmail.com

***Abstract*—Continuous proximity detection monitors the real-time positions of a large set of moving users and sends an alert as long as the distance of any matching pair is smaller than the threshold. Existing solutions construct either a static safe region with maximized area or a mobile safe region with constant speed and direction, which cannot not capture real motion patterns. In this paper, we propose a new type of safe region that relies on trajectory prediction techniques to significantly reduce the communication I/O. It takes into account the complex non-linear motion patterns and constructs a stripe to enclose the sequence of future locations as a predictive safe region. The stripe construction is guided by a holistic cost model with the objective of maximizing the expected time for the next communication. We conduct experiments on four real datasets with four types of prediction models and our method reduces the communication I/O by more than 30% in the default parameter settings.**

I. Introduction

Due to the high availability of location positioning sensors and rising popularity of location sharing services, novel applications in real-time trajectory analyzing and monitoring have emerged as hot research topics and attracted great attention. In this paper, we study the problem of continuous proximity detection among a large group of moving users that are connected by an interest graph. The problem tracks user locations and sends alert messages to the matching pairs as long as their distance is smaller than a specified threshold. It finds a wide scale of applications:

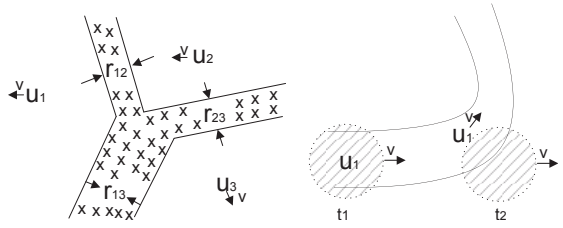
1) **Social friendship tracking**. There have been a number of friend-locator services such as Ipoki[1] that allow users to share their locations with friends. The proximity detection functionality is a natural extension of such services to automatically detect and alert whether there are friends nearby.
2) **Location-based social discovery**. Proximity detection can serve as an attractive feature in location-based dating apps such as Tinder[2]. Alarming messages will be sent to each pair of users with matching profiles when their physical distance is small enough.
3) **Item trading in MMOG**. In massively multiplayer online games, two users can be virtually connected in an interest graph when one is equipped with an item desirable for the other. A trading request can be sent to the matching pairs if they are visible to each other in the virtual world.

Corresponding Author: Dongxiang Zhang
[1]https://appcircus.com/apps/ipoki
[2]https://www.gotinder.com/

In spite of practical usefulness, continuous proximity detection is both computationally and communicationally expensive. It is more challenging than traditional continuous spatial kNN/range queries [2], [6], [22] because in our problem setting, each object is considered as a moving query and we need to track the dynamic pair distance in all the timestamps. Thus, a straightforward solution has to update the locations of all the moving users in every timestamp for distance validation to ensure no missing alert. To resolve the efficiency issues, the essential tactic among previous work of continuous query processing is to customize a type of safe regions for a particular query. Each moving query object is then associated with a safe region in a tailored shape such that no communication is required to report its latest position as long as it is still located within the safe region.

There have been two types of safe regions proposed for continuous proximity detection. In [3], Amir et al. proposed a static safe region in the form of a polygon. As shown in Figure 1(a), three users $\{u_1, u_2, u_3\}$ are friends to each other. Each pair (u_i, u_j) is associated with an alert radius, denoted by r_{ij}. There is a warning area marked with crossings between each pair of users, with width set to r_{ij}. In this way, each user is enclosed by a polygon whose edges are the boundaries of the warning areas. The next communication occurs when a user crosses one of the boundaries. If there are several friends nearby, the safe region may update frequently, incurring many times of communication I/O. Later on, an alternative type of safe region in the form of moving circles was proposed in [19]. As shown in Figure 1(b), a mobile safe region is constructed for o_1 at timestamp t_1 and moves in the same direction as o_1 with constant speed. The next communication occurs at t_2 when the user moves out of the mobile region. It is noticeable that such an assumption of linear motion pattern

is not realistic. If the driving/walking trajectory is curving, the user is likely to deviate the original moving direction many times, incurring frequent safe region re-construction.



(a) Static safe region (b) Mobile safe region

Figure 1. Examples of Safe Region Construction.

In this paper, we propose a new type of safe region that leverages trajectory prediction techniques to capture the dynamic and complex motion patterns of real users, and exhibits effectiveness in reducing the number of communication between moving users and backend servers. More specifically, our safe region is represented by a stripe with fixed-radius, enclosing a sequence of predictive points derived from an imperfect model. To determine the optimal length and radius of the stripe, we need to consider simultaneously two types of communication that have conflicting requirements in terms of stripe area: one is location update from user to server (which can be reduced by larger stripe area) and the other is probing request from server to user (which can be reduced by smaller stripe area). We propose a holistic cost model to achieve a good tradeoff, with the goal of maximizing the expected time for the next communication.

We conduct experiments on four real datasets that cover the motion patterns of taxi drivers, pedestrians and trucks. In the experiments, we examine four existing trajectory prediction models, including RMF [15], HMM [13], R2-D2 [23] and Kalman filter [20]. Our objective is to verify the effectiveness of our proposed safe region built on top of various prediction models. The results show that, compared to the state-of-the-art solution, our predictive safe region can reduce the communication I/O by more than 30% in the default parameter settings.

To sum up, we make the following contributions in this paper:

- We propose a new type of safe region construction, which is able to capture the non-linear motion patterns, to solve the problem of continuous proximity detection.
- We propose a holistic cost model to guide the safe region construction, with the objective of minimizing the communication cost.
- We conduct extensive experiments on four datasets and examine the effectiveness of our prediction safe region construction on top of four types of prediction models. The results verified the superiority of our methods.

The rest of the paper is organized as follows. We present problem definition in Section II and review related work in Section III. In Section IV, we present our high-level processing framework for continuous proximity detection. In Section V, we propose to use fixed-radius stripe as our predictive safe region and a cost model for stripe construction. Extensive experiments are conducted in Section VI to compare the accuracies of prediction models and evaluate the performance of our predictive safe region in terms of communication I/O. Section VII concludes the paper.

II. Problem Definition

Our system monitors a large group of moving users virtually connected by an *interest graph* $G = (V, E)$. G is an undirected graph and is used as a general data model to capture various matching semantics between two users for a wide scale of applications. For example, in the social friendship tracking, two users are connected according to their relationship in the online social networks; in the location-based dating apps, we are interested in investigating whether a user's profile matches the other's dating requirement in terms of age, salary and other attributes; and in the item trading application in MMOG, there is an edge between user u and user w in the interest graph if one is equipped with an item that is desirable to the other. For the ease of presentation, we call u and w "*friends*" if they are connected by an edge in the interest graph.

Each edge (u, w) in the interest graph is associated with an alerting radius denoted by $r_{u,w}$, which can be either a system parameter or specified by users. If u and w have different preference on the distance for alerting, say r_u and r_w, we set $r_{u,w} = \min\{r_u, r_w\}$ to avoid spam messages. In other words, a message is sent only if u and w are matching users and their current distance meets the alarming requirements of both users. We assume that the temporal dimension is split into fixed-size intervals, each with length Δ_t, and all the moving users report their current locations at the beginning of each interval for proximity detection. Let $d(u, w, t_i)$ denote the Euclidean distance between u and w at timestamp $t_i = i\Delta_t$. We use Euclidean distance instead of road network distance because in the real-time monitoring application, it is very challenging to support road network distance. First, the calculation of network distance is much more expensive than Euclidean distance, even with the support of indexing techniques. Second, the safe region techniques are difficult to be applied as the calculation of network distance between a point and a region is also very expensive. Thus, we only consider the Euclidean distance as a practical solution for real-time proximity tracking.

In the following, we formally define the continuous proximity problem:

Definition 1: Continuous Proximity Detection
Given a group of moving users connected by an interest graph $G = (V, E)$, the continuous proximity detection

problem sends alert messages to u and w at timestamp t_i if $(u, w) \in E$, $d(u, w, t_i) < r_{u,w}$ and $d(u, w, t_{i-1}) \geq r_{u,w}$.

In our definition, we avoid spam messages by only sending an alarming message when two matching users meet their distance requirements for the first time. This can be implemented by maintaining a status for each pair of friends. When u and w are alerted in t, the status for (u, w) is set to be active. In the subsequent timestamps, if the distance is still smaller than $r_{u,w}$, no alert messages will be sent. Otherwise, we reset the status of (u, w).

In our system implementation, we adopt the client-server architecture used in many spatial continuous query processing applications [14], [19], [5]. As shown in Figure 2, the server maintains the interest graph and the alert radius of each user. There are six moving users with different interest radii, represented in circles with different sizes. In each timestamp, all the moving users report their latest status information to the server. The server then checks the distance between all pairs of friends in the interest graph and sends alert messages to those pairs with distance smaller than the alert radius. In this example, u_1 and u_2 will be alerted in timestamp t_2 because their distance is smaller than both of the alert radii.

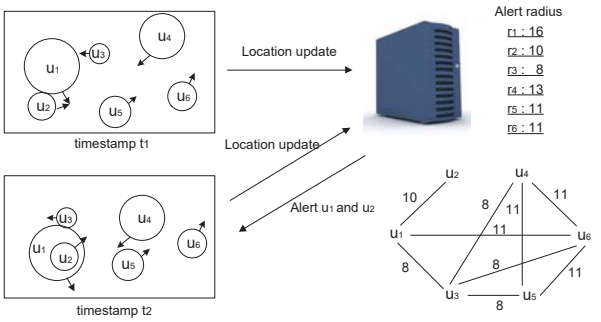


Figure 2. A client-server architecture for continuous proximity detection.

III. Related Work

A. Continuous Proximity Detection

The problem of continuous proximity detection was first proposed in [3] and solved by constructing a static safe region in the form of a polygon for each moving user. As long as the user is in the safe region, he/she does not need to report the location in the subsequent timestamps. The mobile safe region technique [19] was proposed to construct a circle that moves along with the user with constant speed and direction. It works well if the users are moving with linear motion patterns. However, when driving/walking in a curving path, the user is likely to deviate the original moving direction many times, incurring a large amount of communication I/O. In [9], given a large number of concurrent proximity queries within the same timstamp, Kazemitabar et al. focused on determining an appropriate sequence of probing requests to reduce communication I/O. Intuitively, a moving object within a larger safe region is more likely to be probed.

In this paper, we propose a novel concept named predictive safe region for continuous proximity detection. To the best of our knowledge, this is the first work to capture the dynamic and complex moving patterns in a cost model to support a new type of safe region construction.

B. Trajectory Prediction Approaches

Trajectory prediction is a well-studied topic and various methods have been proposed. In [15], Tao et al. proposed recursive motion function (RMF) to support a broad class of non-linear motion patterns, which eventually can improve the efficiency of answering predictive range query in spatio-temporal databases. RMF does not make any assumption on prior movement patterns and can derive the particular motion of each object from its recent history. Its generality is built on the observation that any polynomial motion function of degree D can be converted to a linear recurrence after $D+1$ differentiations. The parameters to be learned are stored in a motion transition matrix, which can be learned by applying singular value decomposition (SVD). The advantages of RMF are that it only requires a small number of recent points for prediction and the model is friendly for implementation. However, its prediction accuracy is the major concern.

R2-D2 [23] was designed for dynamic environments where the motion patterns may change frequently. Unlike RMF that only relies on the recent locations of a single trajectory, R2-D2 needs to first obtain a set of similar trajectories from the historical database, with the help of a grid-based index to improve the efficiency. Then, it applies sophisticated machine learning techniques such as particle filter [1] on the selected reference trajectories to construct a local model for prediction. It shows better prediction accuracy than RMF because the similar reference trajectories can provide useful and reliable information.

Hidden Markov Model (HMM) [13] is a classic model for sequence prediction. To predict future paths, we use the basic version of HMM in the discrete form. We split the space into grid cells and treat each cell as a state. Our objective is to learn the transition probability between the cells, which is trained by the Baum-Welch algorithm. Then, we use forward algorithm to infer a future path with the highest probability.

We also examine the model of Kalman filter [20] in the application of trajectory prediction. It is similar to a hidden Markov model except that the state space of the latent variables is continuous and all latent and observed variables have Gaussian distributions. The procedure of Kalman filter consists of two steps: predict and update. The predict phase uses the previous state information to produce an estimation of the current state. Subsequently in the update phase, the current a priori prediction is combined with the new observation data to refine the estimation. The performances of these prediction models will be examined in Section VI-B.

Table I
NOTATION TABLE

v_u	the moving speed of user u
N_u	the friends of u in the interest graph
r_u	the alert radius set by user u
l_u	the latest precise location of user u
$d(u, w)$	the distance between u and w
$r_{u,w}$	$\min(r_u, r_w)$
S_u	the stripe of user u as a safe region
s^u	the stripe radius
m	the stripe length
$M_{u,w}$	the match region between user u and w

C. Continuous Query Processing

Another problem that is closely related to our research is the processing of continuous spatial queries such as continuous spatial kNN/range queries [6], [22], continuous spatial keyword queries [17], continuous shortest path query [18], [21] and continuous boolean expression matching [5], [4]. To reduce the communication overhead, these methods typically construct safe regions for the moving users and require users to update their locations only when they move out of their safe regions. Moreover, users within the safe regions can safely disconnect from the server as long as there is no matching event in their neighborhood.

The above works assume that one of the matching parities is moving and the other is located in a static position. In contrast, our continuous proximity detection problem is more challenging because it assumes that both matching parties are moving objects.

D. Real-time Trajectory Tracking

In real-time trajectory tracking , each moving user reports his location to the server when the distance between the real location and the predicted location at the server side exceeds the threshold. TRAX [7] is a system supporting various object tracking techniques with low communication cost. EnTracked [10] and [11] are two works that track mobile devices of pedestrians and provide a configurable interface to adjust the tradeoff between energy consumption and tracking accuracy. These works cannot be applied in our proximity detection problem because we do not need to track user locations all the time. When there are no matching users nearby, it wastes communication I/O to track the precise locations that of no help to the proximity detection.

IV. CONTINUOUS MATCH FRAMEWORK

In this section, we present a high-level framework for continuous proximity detection based on two concepts, namely *safe region* and *match region*.

A. Safe Region

In many continuous spatial-query applications [22], [6], [2], [5], the query location has been updating as each user moves around, causing the results to change accordingly. A naive processing framework for this type of query is to request all the users to report their location updates in each timestamp. With the new location information, the backend server re-calculates the results and notifies the users if there is an update. Obviously, the framework is wasteful in both network and CPU resources, incurring a huge amount of unnecessary communication and calculation overhead, especially when the query results of two consecutive timestamps are identical.

To mitigate the issue, most applications construct a safe region for each user such that the next communication is incurred when the user moves out of the safe region or the server actively sends a probing request to retrieve the exact user location. Since there is no need to report location update in each timestamp, a great amount of network communication I/O can be saved. Moreover, there are fewer number of requests sent to the server and the computation cost in the server side can be reduced as well.

The construction of a safe region is normally query-dependent. In our application, a user is said to be safe if his/her distance to all the neighbours in the interest graph is larger than $r_{u,w}$. Consequently, we formally define the concept of safe region in the problem of continuous proximity detection as follows:

Definition 2 (Safe Region): The safe region S_u for a user u is a region such that u's location $l_u \in S_u$ and for any friend w, we have $d(l_u, S_w) > r_{u,w}$.

In the implementation, the backend server constructs a safe region for each user. From the server's perspective, it only knows the fact that the user is located within the safe region. However, the precise user location is not available unless the server actively sends a probing request for location update. The optimal safe region construction is a challenging task. On the one hand, we prefer a larger region such that the user can stay within it as long as possible. On the other hand, all the users are moving objects. When the minimum distance between the safe regions of two friends u and w is smaller than $r_{u,w}$, the server has to send a probing request for location update of u and w to guarantee that no matching alert is missed. We will present our safe region representation and construction in the next section. In this section, we simply treat them as black-boxes and focus on the high-level search framework.

B. Match Region

When a matching pair (u, w) is detected with distance smaller than $r_{u,w}$, an alert notification is sent. At this moment, the concept of safe region is not working because we cannot construct a region for u and w satisfying the constraints in Definition 2. The two users have to report location update at the subsequent timestamps until their distance is larger than $r_{u,w}$ again.

To save the communication cost for the above matching scenario, we propose a new concept named *match region* which is complementary to the safe region. Given a user u

with a group of matching friends, we remove each matching pair (u, w) from the interest graph and maintain a match region $M_{u,w}$. When the distance $d(u, w)$ is larger than $r_{u,w}$, we remove the match region and fill the edge back to the graph; for the remaining non-matching friends, the original safe region can still be applied.

The intuition behind the match region is that as long as two matching friends u and w are located within their match region $M_{u,w}$, their distance must be smaller than $r_{u,w}$. Thereby, we define our match region as follows:

Definition 3 (Match Region): The match region for a matching pair (u, w) is defined as a circle with center at $\frac{l_u+l_w}{2}$ and radius $\frac{r_{u,w}}{2}$.

Figure 3 shows that u and w are matching at timestamp t_i. Then, we construct a match region in the form of a circle for the matching pair. The center is $(\frac{u.x+w.x}{2}, \frac{u.y+w.y}{2})$ and the radius is $\frac{r_{u,w}}{2}$. At timestamp t_{i+1}, u and w will compare their new locations with the match region. Since both locations are within the match region, no communication is triggered. If u and w share the same trajectory after they meet, the match region can help save a considerable amount of communication I/O. Finally, at timestamp t_{i+2}, both u and w move out of the match region and their distance is now larger than $r_{u,w}$. They will report their locations to the server for new safe region construction.

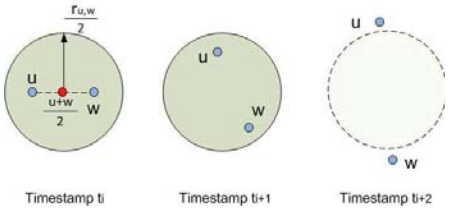


Figure 3. An example of match region construction.

C. High-Level Search Framework

The search framework for continuous proximity detection is illustrated in Algorithm 1. In the initialization step (lines 1-7), all the users report their current locations to the backend server. When receiving the location update, the server first detects whether there are friend-matching events. If yes, a match region $M_{u,w}$ is constructed for each matching pair and sent together with an alert notification to u and w. If a user does not match any friend in the first timestamp, no match region will be maintained. In the meanwhile, the server initializes a safe region S_u for each user. The details of safe region initialization will be introduced in Section V-C.

After the initialization, each user u is associated with m match regions $(0 \leq m \leq |N(u)|)$ and one safe region S_u. At the subsequent timestamps, user u will compare his location with the maintained match region and safe region. If u moves out of any match region $M_{u,w}$, he/she will update the location to the server. Then, the server sends a probing request to the friend w to examine whether their precise distance is still within the alert radius. If yes, $M_{u,w}$ is updated based on the new locations of u and w. Otherwise, $M_{u,w}$ is deleted. Meanwhile, we need to update the safe region S_u and S_w based on their new locations.

After the match region update, user u also needs to check whether he/she moves out of the safe region. If yes, the server needs to collect the latest location of u and update the safe region. We will present an algorithm for safe region update in Section V-D.

Algorithm 1: Continuous Proximity Detection

```
1.  if timestamp is t_1 then
2.      all the users report their locations to server
3.      for each matching pair (u, w) do
4.          construct M_{u,w}
5.          send alter notification and M_{u,w} to u and w
6.      for each user u do
7.          init S_u
8.  for each subsequent timestamp t_i do
9.      for each user u do
10.         for each match region M_{u,w} do
11.             if u moves out of M_{u,w} then
12.                 u and w report their locations to server
13.                 set loc[u][t_i] ← 1 and loc[w][t_i] ← 1
14.                 if d(u, w) ≤ r_{u,w} then
15.                     update M_{u,w} based on the new locations
16.                 else
17.                     delete M_{u,w} from users u and w
18.                     update S_u and S_w
19.         if u moves out of safe region S_u then
20.             u reports the location to server if loc[u][t_i] ≠ 1
21.             update S_u
```

V. Predictive Safe Region

The static safe region method [3] assumes that users are random walkers and moving in all the directions with the same probability. When constructing a new safe region for a user, it scans all the friends and builds a barrier w.r.t each friend. These barriers or boundaries constitute a polygon as the safe region. However, the construction method only utilizes the location information between each pair of friends while neglecting the user motion pattern.

In this paper, we look at the problem from a new perspective and propose a novel concept named *predictive safe region*. If the future locations of a user are predictable, we can construct a safe region along the predicted path. Obviously, the higher the prediction accuracy is, the longer the user will stay in the safe region. We treat the trajectory prediction algorithm as a black-box and assume that any predicting techniques [15], [8], [23] that produce a sequence of locations in the subsequent timestamps can be applied. The predictive safe region is then built on top of the the sequence of future locations. In this section, we represent the predictive safe region in the form of a time-independent stripe with fixed radius.

A. Fixed-Radius Stripe

Let the output of a trajectory prediction algorithm at timestamp t_i be a sequence of locations $p_1, p_2, \ldots, p_m$,

where p_j is the predicted location at timestamp t_{i+j}. In the ideal scenario, the prediction algorithm is perfect and we can model the predictive safe region as a sequence of locations. Since we have known the precise future locations of all the users, there is no more probing overhead and the communication cost is optimal. However, prediction error occurs frequently in the real scenario and the safe region must be designed to be error-tolerant. We propose to build a fixed-radius stripe along the predicted path as our predicted safe region.

Definition 4: Stripe with Fixed Radius
Given a sequence of future locations $p_1, p_2, \ldots, p_m$ for a user u, the fixed-radius stripe extends the line segments $\overline{p_i p_{i+1}}$ by radius s^u, i.e., $R = \{o| \min_{1 \le i \le m-1} \{dist(o, \overline{p_i p_{i+1}})\} \le s^u\}$.
where s^u and m are two parameters to determine the shape of the stripe. An example of stripe derived from the next seven locations is illustrated in Figure 4. Note that when we determine whether the user is in the predictive safe region, we do not consider the temporal attribute. In the next timestamp, we compare the user location with the whole stripe instead of the predicted location. This provides more error tolerance when a user moves along the predicted path but violates the speed assumption. For example, existing prediction approaches may assume the user move forward with constant speed in Figure 4. Even though the distance between the real locations and the counterparts in the predicted path is quite large, the user is still considered within the predicted safe region.

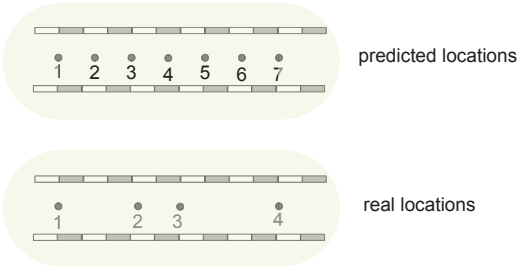


Figure 4. An example of fixed-radius stripe

We assume that the prediction error of existing approaches [15], [8], [23], measured by the distance between the predicted location and the precise location, follows Gaussian distribution $N(0, \sigma^2)$. Each approach is associated with a particular parameter σ to measure its intrinsic predication accuracy. If an approach is superior in predicting future locations, it is associated with a smaller value of σ. Under such assumptions, we present how to determine a good fixed-radius stripe by proposing a cost model.

B. Cost Model for Stripe Construction

The primary objective of our system is to minimize the number of communication I/O. We first identify all the possible cases in which new communication will be triggered for a moving user.

1) The user moves out of his current safe region. He/She needs to report the new precise location to the server. At the server side, a new safe region is calculated and sent back to the user.
2) The user is still in the safe region but receives a probing request from the server. This occurs when the minimum distance of a friend is smaller than the alert radius and the exact distance between the two users is uncertain. The user in the safe region has to report the precise location to avoid missing alert. After the probing, the server will construct a new safe region for the probed user.
3) The user receives an alert message from the server. This occurs when the distance of a friend is smaller than the alert radius.
4) The user moves out of a match region and needs to report the new location.

Note that the third type of communication I/O is inevitable and cannot be optimized. In addition, the safe region and match region are functioned upon different groups of friends. Thus, the next timestamp when a user moves out of a match region will not be affected by how the safe region is constructed. Thereafter, we only need to consider the first two types of communications in our cost model. The first type prefers a larger safe region so that the user can stay in the safe region for a longer time. However, the second type prefers a smaller safe region so that its minimum distance to the friends will not be too small. This can help trigger fewer probing requests from the server.

To handle the conflicting requirements, we propose a new cost model to minimize an objective cost function, which is measured by the expected number of the two types of communication I/O for a user. We assume that the underlying trajectory prediction model can predict the next m steps of a user with high confidence. Our goal is to determine a good radius s^u of the stripe with m segments such that the elapsed time before the next communication is maximized. Let E_m denote the elapsed time for the user to move out of the stripe with radius s^u and E_p denote the elapsed time for the next probing request from the server. Let $f_{obj}(s^u) = \min(E_m, E_p)$, our objective function is

$$\arg\min_{s^u} f_{obj}(s^u) \tag{1}$$

Intuitively, when s^u increases, the safe region becomes larger and more error-tolerant. It takes a longer time for a user to move out of the stripe, resulting in larger E_m. However, it becomes more likely for the user to be probed by friends nearby, resulting in smaller E_p. Therefore, the best s^u occurs when $E_m = E_p$.

C. Safe Region Initialization

In the system initialization, the server collects the locations of all the users and checks the spatial distance between each pair of friends. If the distance is found to be smaller than $r_{u,w}$, two alert messages are sent to u and w. In the meanwhile, the server constructs a static safe region, denoted by S_u, for each user u. The safe region is a circle centered at user u's current position with radius s^u. For each pair of friends (u, w), the radii s^u and s^w have to meet the following requirement to ensure safety:

$$s^u + s^w + r_{u,w} \leq \tau_{u,w} \tag{2}$$

where, $\tau_{u,w}$ is the distance between the center points of the two circles. Once the constraint is satisfied for all the users, we can prove the safety of circle S_u via the following lemma.

Lemma 1: If $s^u + s^w + r_{u,w} \leq \tau_{u,w}$ is satisfied for all the users, there is no matching pair whose distance is smaller than their alert radius.

Proof: Let $M(u)$ denote the set of friends of u in the interest graph. For any $w \in M(u)$, we know that $u \in S_u$ and $w \in S_w$. Thus, their distance $dist(u, w) \geq dist(S_u, S_w) = \tau_{u,w} - s^u - s^w \geq r_{u,w}$. ■

Since this is the initialization step and we lack recent trajectories for prediction, we assume that the users move with constant speed and direction in the following timestamps. The expected elapsed time to leave the circle is

$$E_m = \frac{s^u}{v_u}, \tag{3}$$

where v_u is the moving speed of user u.

To estimate E_p, the worst case occurs when all the friends are moving towards user u. In this case, the elapsed time for the next probing request is

$$E_p = \min_{w \in M(u)} \frac{\tau_{u,w} - s^u - r_{u,w}}{v_w} \tag{4}$$

Let $E_m = E_p$, we have

$$s^u = \min_{w \in M(u)} \frac{v_u(\tau_{u,w} - r_{u,w})}{v_u + v_w} \tag{5}$$

We prove that the above assignment satisfies the distance constraint in Equation 2.

Lemma 2: If we set $s^u = \min_{w \in M(u)} \frac{v_u(\tau_{u,w} - r_{u,w})}{v_u + v_w}$, we have $s^u + s^w + r_{u,w} \leq \tau_{u,w}$ for each friend w.

Proof: For each pair of friend (u, w), we have

$$\begin{aligned} s^u + s^w &\leq \frac{v_u(\tau_{u,w} - r_{u,w})}{v_u + v_w} + \frac{v_w(\tau_{u,w} - r_{u,w})}{v_u + v_w} \\ &= \tau_{u,w} - r_{u,w} \end{aligned}$$

■

D. Safe Region Update

In Section V-C, we have studied safe region initialization when the precise locations of all the users are available. It corresponds to the case of $m = 1$ and the prediction accuracy is 100%. Here, the estimation of E_m and E_p for new safe region construction is more challenging. First, the precise locations of friends may not be available. If the friends are in the safe region, the server is not aware of their precise locations and uses the stripes as an approximation. Otherwise, they also move out of the safe region at the current timestamp and report their new locations to the server. Second, we need to take into account the prediction error when estimating E_m and E_p. We propose a simplified model to estimate the probability of staying in the stripe with radius s^u. Since the predicted distance error is measured by the Gaussian distribution, if the user is located in the stripe at the timestamp t_i, we set the probability of staying in the stripe in the next timestamp as

$$p = \int_0^{s^u} \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{x^2}{2\sigma^2}} dx \tag{6}$$

The simplified probabilistic process for the next m steps is shown in Figure 5. At timestamp t_i, the user moves out of the old safe region and the precise location is available. At timestamp t_{i+1}, the user has probability $1 - p$ to move out of the safe region and probability p to be within distance s^u to the predicted location a_1. The probability analysis in the subsequent timestamps is similar to that in t_{i+1}. Hence, the probability to stay in the safe region in the m-th step is p^m.

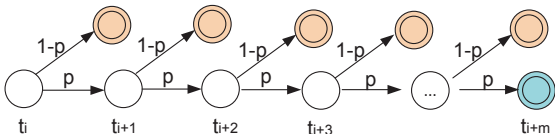


Figure 5. Probabilistic process of path predicting

Based on the probabilistic process, we start to estimate E_m and E_p for the new constructed stripe. At timestamp t_i, the user has probability $1 - p$ to deviate the path and move out of the safe region. The estimated staying time is $T_0 = (1 - p)\frac{s^u}{v_u}$. At timestamp t_{i+1}, the estimated staying time is $T_1 = p(1 - p)(\frac{s^u}{v_u} + \Delta_t)$. At timestamp t_{i+2}, the estimated staying time is $T_2 = p^2(1 - p)(\frac{s^u}{v_u} + 2\Delta_t)$. The process repeats with $T_i = p^i(1 - p)(\frac{s^u}{v_u} + i\Delta_t)$. Finally, at timestamp t_{i+m}, we use $T_m = (\frac{s^u}{v_u} + m\Delta_t)p^m$ to estimate the elapsed time of reaching the m-th step and then leaving the current safe region to trigger the next round of safe

region construction. By summing all the T_i up, we have

$$\begin{aligned} E_m &= T_0 + T_1 + T_2 + \ldots + T_m \\ &= \frac{s^u}{v_u} + \Delta_t(\sum_{i=1}^{m-1} ip^i(1-p) + mp^m) \\ &= \frac{s^u}{v_u} + \Delta_t(p + p^2 + \ldots + p^{m-1} - (m-1)p^m + mp^m) \\ &= \frac{s^u}{v_u} + \frac{\Delta_t p(1-p^m)}{1-p} \end{aligned}$$

The next probing request for user u occurs when the minimum distance of a friend to the new constructed stripe S_u is smaller than the alert radius. If a friend's current location is available, we use o to denote the location and its distance to S_u is defined as

$$d_w = d(o, S_u) = \min_{1 \le i \le m-1} d(o, \overline{a_i a_{i+1}}) - s^u \quad (7)$$

where a_i and a_{i+1} are two predicted locations for user u and $d(o, \overline{p_i p_{i+1}})$ is the minimum distance from a point o to a segment $\overline{p_i p_{i+1}}$. Otherwise, we use the distance between two stripes to estimate E_p.

$$d_w = \min\{\min_{1 \le i \le m} d(a_i, S_w) - s^u, \min_{1 \le j \le n} d(b_j, S_u) - s^w\} \quad (8)$$

where $a_1, a_2, \ldots, a_m$ are the predicted locations for S_u and $b_1, b_2, \ldots, b_n$ are locations for S_w. Then, we estimate the elapsed time E_p for the next probing request as

$$E_p = \min_{w \in M(u)} \{\frac{d_w - r_{u,w}}{v_w}\}$$

If we can find s^u such that $E_m = E_p$, then we have $E_p > 0$ and $d_w > r_{u,w}$ for all the friends of u. In other words, the minimum distance from u to w is larger than the alert radius when they are in the stripe. Hence, the stripe is a safe region.

E. Stripe Construction Algorithm

Although we have proposed a cost model to estimate E_m and E_p, we still face two challenges to construct a new stripe for user u. First, for a given m, it is difficult to solve the equation $E_m = E_p$ directly. Second, we need to determine the number of subsequent timestamps for trajectory prediction.

To solve the first issue, we leverage the properties that $E_p \ge 0$ and E_p decreases monotonically with s^u. We first rewrite d_w in Equations 7 and 8 in the form of $y_0 - s^u$. Then, we have

$$E_p = \min_{w \in M(u)} \{\frac{y_0 - s^u - r_{u,w}}{v_w}\} \ge 0 \quad (9)$$

Thus, $\min_{w \in M(u)} \{y_0 - r_{u,w}\}$ is an upper bound value for s^u. Our algorithm starts by setting s^u to the upper bound value and estimate E_m and E_p. If $E_m \le E_p$, the algorithm can terminate because if we further decrease s^u, the gap between E_m and E_p becomes larger. If $E_m > E_p$, we use binary search by setting $s^u = \frac{s^u}{2}$ and compare the new estimated E'_m and E'_p. If $E'_m \le E'_p$, we set $s^u = \frac{3s^u}{4}$. Otherwise, $s^u = \frac{s^u}{4}$. The process repeats and the algorithm terminates when $|E'_m - E'_p| < \epsilon$.

Algorithm 2: Stripe construction

```
1. m ← ⌊log_p(p_min)⌋; E' ← 0
2. for each friend w of user u do
3.    for i = 1; ; i++ do
4.       if d(a_i, S_w) ≤ r_{u,w} then
5.          m ← min{m, i}
6.          break
7. for i = 1; i ≤ m; i++ do
8.    s^u ← min_{w∈M(u)}{y_0 − r_{u,w}}; min ← 0; max ← s^u
9.    estimate E_m and E_p
10.   if E_m ≤ E_p then
11.      if E_m > E' then
12.         E' ← E_m
13.         m' ← m; s^{u'} ← s^u
14.   else
15.      while |E_m − E_p| ≥ ε do
16.         if E_m ≤ E_p then
17.            s^u ← (min+max)/2; max ← s^u
18.         else
19.            s^u ← (min+max)/2; min ← s^u
20.         estimate E_m and E_p
21.      if E_m > E' then
22.         E' ← E_m
23.         m' ← m; s^{u'} ← s^u
24. return (m', s^{u'})
```

To solve the second issue, our strategy is to first determine an upper bound value for m and iteratively calculate the best s^u for each possible value of m. The one with the maximum E_m or E_p is selected. For example, Figure 6 illustrates the construction of S_u based on the sequence of predicted locations $a_1, a_2, \ldots, a_7$. S_w is an existing stripe of a friend w. In this example, $m = 6$ is the upper bound value because the minimum distance from a_7 is smaller than the alert radius. If there is no such upper bound, we select the maximum m such that $p^m \ge p_{min}$, where p_{min} is set to a small probability. In other words, as m increases, the prediction precision at the m-th step drops and p_{min} indicates the threshold of tolerance of such prediction accuracy. When the upper bound is determined, we starts from $m = 1$ and iteratively find the maximum value of E_m (or E_p) for each option value of m. The pair (m', $s^{u'}$) that derives the maximum E_m is selected and returned. The details of the stripe construction algorithm are shown in Algorithm 2.

VI. Experimental Study

A. Datasets

In the experimental study, we use four real datasets to evaluate the performance of our predictive safe region.

- **GeoLife**. The dataset essentially keeps all the travel records of 182 users for a period of over three

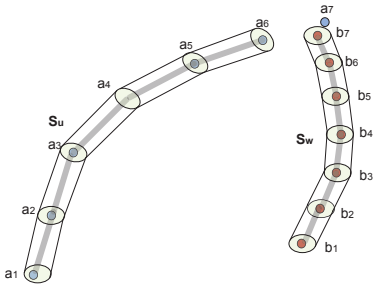


Figure 6. An example to illustrate the upper bound value of m.

years, including multiple kinds of transportation modes (walking, driving and taking public transportation).

- **Bejing Taxi**. It contains GPS trajectories from $33,000$ taxis in Beijing over 3 months. The average sampling interval of the dataset is 3.1 minutes per point.
- **Singapore Taxi** [16]. It contains trajectories from $13,200$ taxies in Singapore over one week. Each taxi continuously reports its locations at a frequency of 20-80 seconds.
- **Truck** [12]: The dataset is the GPS trajectories collected by trucks equipped with GPS sensors in China during a period from Aug. 2015 to Oct. 2015. The sampling rate varied from 1s to 60s.

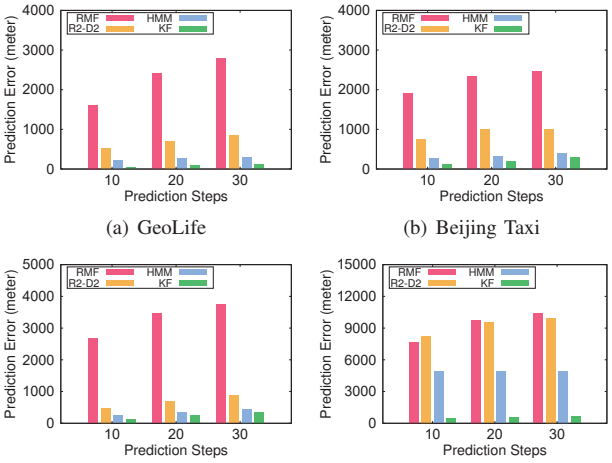
To simulate moving users connected in an interest graph, we interpolate the trajectories with 5 seconds as a step and pick $10,000$ trajectories as moving users from each dataset. We follow [19] to randomly generate the interest graph in a synthetic way. There is a parameter F to control the density of the graph and each user has an average of F friends.

B. Comparison of Prediction Models

The training dataset contains $1,600$ synchronized timestamps of 10K moving objects. The codes for RMF and R2-D2 are provided by the authors and we implement HMM model and Kalman filter by our own. To train HMM models, we split the map into 100×100 grid cells and treat each cell as a state. The number of hidden variables is set to 10. The process noise and measurement noise parameters for the Kalman filter are also tuned for the best performance.

In the following, we use average Euclidean distance (in the unit of meters) to measure the error of a trajectory prediction model. We also report the prediction time as this step is a frequent operator in proximity detection. Each time we re-construct the safe region, we need to call the prediction function to obtain a sequence of future locations. We fix the input length as 10 and vary the output length to 10, 20 and 30. In other words, given a recent history of 10 points, our goal is to predict the locations in the subsequent 10, 20 and 30 steps, respectively.

The prediction errors of four models are depicted in Figure 7. As expected, R2-D2 works much better than RMF in most of the datasets because R2-D2 finds moving objects with similar trajectories in the historical data and leverage their future locations to facilitate prediction, whereas RMF simply attempts to adjust the parameters to fit the recent curve of a single trajectory. In the Truck dataset, their prediction errors are comparable and higher than other datasets. The reason could be that trucks are normally moving in high ways crossing cities or even provinces and there may be not enough historical data for R2-D2 to learn a reliable probabilistic model. HMM and Kalman filter are two classical models to process time series data. The states in HMM are discrete and its probabilistic model learns the transition probabilities between two states. Kalman filter uses continuous states, but with the Gaussian distribution assumption. The results show that Kalman filter achieves smaller errors than HMM, implying that its Gaussian assumption is reasonable in the application of trajectory prediction.



(a) GeoLife (b) Beijing Taxi

(c) Singapore Taxi (d) Truck

Figure 7. Prediction error.

C. Comparison Methods for Proximity Detection

The methods to be evaluated for continuous proximity detection include:

- **Naive**, which updates the locations of all the users at each timestamp. It incurs no probing cost.
- **Static** [3], which constructs static safe regions for the moving users.
- **FMD** [19], which assumes that users are moving in constant direction and speed. It maintains a mobile region for each user as the safe region.
- **CMD** [19], which is an extension of FMD by adjusting the radius of mobile region dynamically. If too many probing requests have occurred in a recent period, the radius would be adjusted smaller. If a moving object frequently report their locations, a larger radius is desired.
- **Stripe+RMF**, which constructs a predictive stripe as the safe region and uses RMF [15] as the underlying prediction model.

- **Stripe+R2-D2** which uses R2-D2 [23] for predictive safe region construction.
- **Stripe+HMM** which uses HMM [13] for predictive safe region construction.
- **Stripe+KF** which uses Kalman filter for predictive safe region construction.

D. Performance Evaluation for Proximity Detection

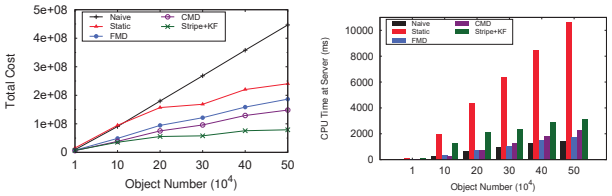
The parameter settings are shown in Table II with the default values in bold. We will evaluate the communication cost of the six methods w.r.t to varying number of friends F (30 by default), number of steps (or timestamps) S (900 by default), moving speed V (8 steps per epoch by default) and alert radius r (6km by default). All the methods are implemented in C++ and run on a CentOS server (Intel i7-3820 3.6GHz CPU with 8 cores and 64GB RAM).

Table II
EXPERIMENT PARAMETERS.

Number of moving objects N	**10K**, 100K, 200K, 300K, 400K, 500K
Average number of friends F	10, 20, **30**, 40, 50
Number of steps S	300, 600, **900**, 1200, 1500
Moving speed V (steps per epoch)	2, 4, 6, **8**, 10, 12, 14, 16
Alert radius r (km)	2, 3, 4, 5, **6**

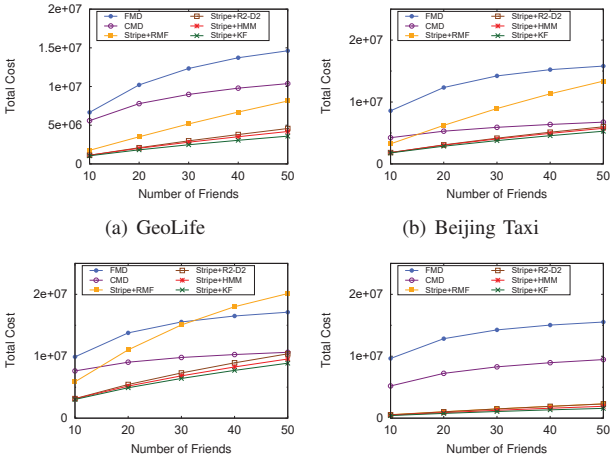
1) Increasing Number of Moving Objects N: Our first proximity experiment examines the I/O and CPU cost with increasing number of moving objects from the Truck dataset. For the Stripe method, we report the performance using Kalman Filter as the prediction model. All the results are shown in Figure 8. As to the CPU cost, the overhead at the client side is negligible because the sampled GPS data only needs to compare with the preserved safe region and check if the point is located within the region. This leads us to focus on the computation cost in the server side as the server needs to monitor a large number of moving objects and check the proximity criterion. The I/O cost of Naive method grows linearly with the number of moving objects and incurs the much higher CPU cost at the server side. The safe region based approaches can reduce both communication I/O and CPU cost at the same time. Among these methods, the static safe region approach performs the worst. When there are friends nearby and the objects are moving at a relatively high speed, it is easy to move out of the safe region and requires to construct a new region. Compared to FMD and CMD, our Stripe+KF reduces the amount of communication I/O significantly. We also observed that the CPU cost of Stripe+KF at the server side is higher than that of FMD and CMD. The reason is that the location prediction model consumes a considerable amount of computing resources.

2) Increasing Number of Friends F: In this experiment, we increase the average number of friends in the interest graph and report the number of communication I/O in Figure 9. Based on the results, we have the following observations. 1) The number of communication I/O among



(a) I/O (b) CPU

Figure 8. Performance with increasing number of moving objects.



(a) GeoLife (b) Beijing Taxi

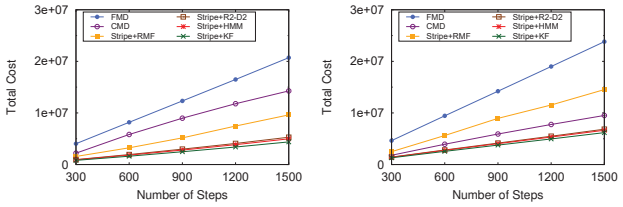
(c) Singapore Taxi (d) Truck

Figure 9. Increasing average number of friends.

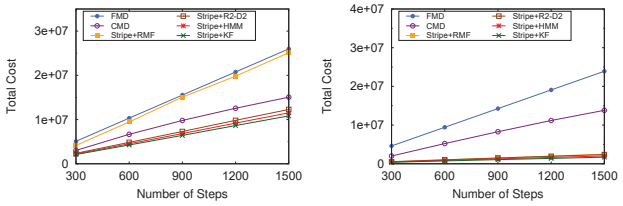
the six methods exhibits a considerable increasing trend as F grows. With more neighbours in the interest network, the probability of two friends in proximity becomes much higher, triggering more probing requests from the server. 2) CMD is superior to FMD in all the datasets as it utilizes a cost model to dynamically adjust the radius of safe region. Thus, we consider CMD as state-of-the-art. 3) The performance of proximity detection is positively related to the accuracy of the underlying prediction model. The more accurate the underlying prediction model is, the less communication I/O is incurred. Since RMF model has the highest prediction error, Stripe+RMF obtains the worst performance. Analogously, Kalman filter is the most accurate prediction model in our experiments and Stripe+KF incurs the minimum amount of communication I/O. In GeoLife, the communication I/O of CMD is 2-5 times higher than that of Stripe+KF. In Truck, this number is 6-12 times higher. These results verify the effectiveness of our predictive safe region strategy and the holistic cost model.

3) Increasing Number of Steps S: Figure 10 illustrates the performance w.r.t. increasing number of steps from 300 to 1500. The patterns are similar to those in the previous experiment. We can see that the total number of communication I/O grows linearly with the number of steps. In the two Taxi datasets, CMD performs better than Stripe+RMF because the taxies are running within a city at

high speed and it is easy for them to move out of the stripe region if the prediction is not accurate. On the other hand, if the prediction error is small enough, the performance of our proposed predictive stripe is considerably better than CMD.



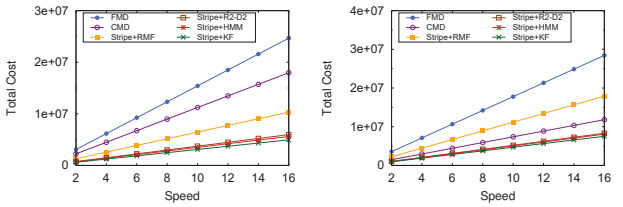
(a) GeoLife (b) Beijing Taxi



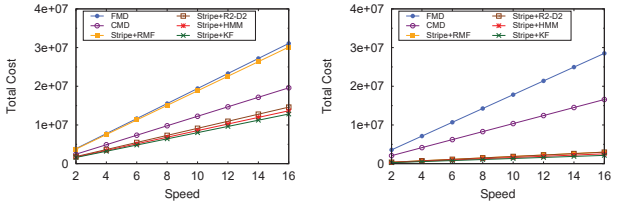
(c) Singapore Taxi (d) Truck

Figure 10. Increasing average number of steps.

4) Increasing Moving Speed V: The performance is also related to the user moving speed. As shown in Figure 11, when the speed grows from 2 steps/epoch to 16 steps/epoch, the number of communication of FMD and CMD increases steadily. It is interesting to observe that the communication I/O of our stripe-based methods only demonstrates a slight increasing pattern in the Truck dataset. Our explanation is that the trucks often run on straight high ways and the derived predictive safe region is likely to enclose the future locations of the trucks. In addition, the trucks are more sparsely distributed in the two-dimensional space, which incur less probing cost.



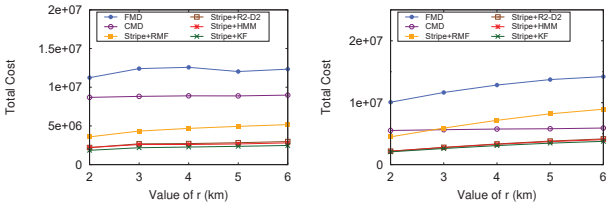
(a) GeoLife (b) Beijing Taxi



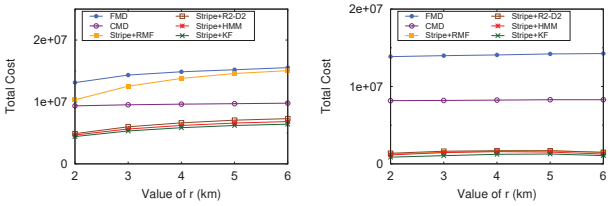
(c) Singapore Taxi (d) Truck

Figure 11. Increasing moving speed.

5) Increasing Alert Radius r: In Figure 12, most of the algorithms degrade slowly as r increases. On one hand, a large alert radius increases the probability of sending a probing request and thus increases the communication I/O. On the other hand, a large radius renders two friends close to each other more likely to be located in the match region. As long as they are in the match region, no more I/O occurs and they will not be considered for each other's safe region construction. Thus, the effect of alert radius depends on the density and motion patterns of moving objects. The two Taxi datasets are more sensitive to r than the remaining two.



(a) GeoLife (b) Beijing Taxi



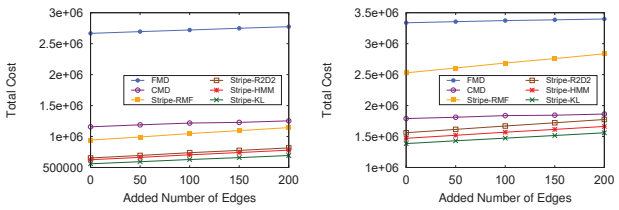
(c) Singapore Taxi (d) Truck

Figure 12. Increasing alert radius.

E. Dynamic Graph Update

In the last experiment, we consider the dynamic scenario with fluctuating user participation during a long-term operation. This renders the interest graph to update on the fly. For instance, users in MMOGs would require a continuous update of the graph since items are acquired and disposed of. The goal is the experiment is to examine how the rate of such a change could affect the communicational overhead.

Given an insertion of an edge (or a pair of new matching friends) in the interest graph, we first calculate the minimum distance of their current safe region. If the distance is larger than the alerting radius, no operation is required. Otherwise, the server sends a probing request to obtain the exact locations in case these two users are matching at this epoch. The safe region of these two probed users may be updated as well. The edge deletion of interest graph is easier to handle. We simply need to update the interest graph and retain their safe regions unchanged. The safe region will be re-constructed in the future epoch when the user moves out of it, but based on the new interest graph. There would be no probing between the deleted pair of friends. Thus, in our experimental setup, we only consider the insertion operator in the interest graph and simulate the dynamic graph update for 100 epochs. In each epoch, we add a number of E edges,



(a) GeoLife (b) Singapore Taxi

Figure 13. Dynamic graph update.

with E increasing from 0 to 200. The results on GeoLife and Singapore Taxi in Figure 13 show that the algorithms scale well with continuous edge insertion in an evolving interest graph. Thus, dynamic update in the interest graph will not become a barrier for the continuous proximity detection.

VII. Conclusion

In this paper, we studied the problem of continuous proximity detection and construct the safe region from a new perspective. We proposed to use stripe as predictive safe region to capture the dynamic and complex user motion patterns. To guide the safe region construction, we proposed a holistic cost model with minimized expected communication I/O. We conducted experiments on four real datasets covering the motion patterns of driving vehicles and walking people. Experimental results showed that our predictive safe region is superior over state-of-the-art solution. It can significantly reduce the communication I/O by more than 30% in the default parameter settings.

References

[1] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *Trans. Sig. Proc.*, 50(2):174–188, Feb. 2002.

[2] B. Bamba, L. Liu, A. Iyengar, and P. S. Yu. Safe region techniques for fast spatial alarm evaluation. *Georgia Institute of Technology*, 2008.

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

[4] L. Guo, L. Chen, D. Zhang, G. Li, K. Tan, and Z. Bao. Elaps: An efficient location-aware pub/sub system. In *ICDE*, pages 1504–1507, 2015.

[5] L. Guo, D. Zhang, G. Li, K. Tan, and Z. Bao. Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In *SIGMOD*, pages 843–857, 2015.

[6] M. Hasan, M. A. Cheema, X. Lin, and Y. Zhang. Efficient construction of safe regions for moving knn queries over dynamic datasets. In *SSTD*, pages 373–379, 2009.

[7] C. S. Jensen and S. Pakalnis. TRAX - real-world tracking of moving objects. In *VLDB*, pages 1362–1365, 2007.

[8] H. Jeung, Q. Liu, H. T. Shen, and X. Zhou. A hybrid prediction model for moving objects. In *ICDE*, pages 70–79, 2008.

[9] S. J. Kazemitabar, F. B. Kashani, S. J. Kazemitabar, and D. McLeod. Efficient batch processing of proximity queries by optimized probing. In *GIS*, pages 84–93, 2013.

[10] M. B. Kjærgaard, J. Langdal, T. Godsk, and T. Toftkjær. Entracked: energy-efficient robust position tracking for mobile devices. In *MobiSys*, pages 221–234, 2009.

[11] R. Lange, F. Dürr, and K. Rothermel. Efficient real-time trajectory tracking. *VLDB J.*, 20(5):671–694, 2011.

[12] X. Lin, S. Ma, H. Zhang, T. Wo, and J. Huai. One-pass error bounded trajectory simplification. *PVLDB*, 10(7):841–852, 2017.

[13] W. Mathew, R. Raposo, and B. Martins. Predicting future locations with hidden markov models. In *Ubicomp*, pages 911–918, 2012.

[14] M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: scalable incremental processing of continuous queries in spatio-temporal databases. In *SIGMOD*, pages 623–634, 2004.

[15] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *SIGMOD*, pages 611–622, 2004.

[16] Y. Wang, D. Zhang, Q. Liu, F. Shen, and L. H. Lee. Towards enhancing the last-mile delivery: An effective crowd-tasking model with scalable solutions. *Transportation Research Part E: Logistics and Transportation Review*, 93:279 – 293, 2016.

[17] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552, 2011.

[18] D. Yang, D. Zhang, K. Tan, J. Cao, and F. L. Mouël. CANDS: continuous optimal navigation via distributed stream processing. *PVLDB*, 8(2):137–148, 2014.

[19] M. L. Yiu, L. H. U, S. Saltenis, and K. Tzoumas. Efficient proximity detection among mobile users via self-tuning policies. *PVLDB*, 3(1):985–996, 2010.

[20] P. Zarchan and H. Musoff. *Fundamentals of Kalman Filtering: A Practical Approach (Progress in Astronautics and Aeronautics)*. AIAA (American Institute of Aeronautics & Ast, 2000.

[21] D. Zhang, D. Yang, Y. Wang, K. Tan, J. Cao, and H. T. Shen. Distributed shortest path query processing on dynamic road networks. *VLDB J.*, 26(3):399–419, 2017.

[22] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee. Location-based spatial queries. In *SIGMOD*, pages 443–454, 2003.

[23] J. Zhou, A. K. H. Tung, W. Wu, and W. S. Ng. A "semi-lazy" approach to probabilistic path prediction. In *SIGKDD*, pages 748–756, 2013.