DistSD: Distance-based Social Discovery with Personalized Posterior Screening

Xiao Pan
Shijiazhuang Tiedao University
Hebei,China
smallpx@stdu.edu.cn

Jiawei Zhang, Fengjiao Wang, Philip S. Yu
University of Illinois at Chicago
Chicago, IL, USA
jzhan9@uic.edu fwang27@uic.edu psyu@cs.uic.edu

***Abstract*—Privacy preservation in location-based proximity services has recently received considerable attention in geo-social networks. Nearby friends notification and social discoveries are two important types of location-based proximity services. A large number of privacy protection methods have been proposed on nearby friends notification, but few on social discoveries. Most of existing protection methods on nearby friends notification cannot be applied in social discoveries, since a secret key needs to be shared between dynamic friends. In this paper, we address the research challenges that the location privacy protection in distance-based social discoveries. We propose a novel framework DistSD for the distance-based social discovery with personalized posterior screening. We also show that the problem that finding an optimal safe group of nearby seeking users is NP-hard. Two heuristic privacy enhanced social discovery algorithms are proposed, which protect users' locations from a service result perspective. Experiments are conducted based on the real-life data and experimental results validate the effectiveness and efficiency of the proposed algorithms.**

I. Introduction

With the advance of location-acquisition and mobile communication technologies, location based proximity services have become increasingly prevalent. Location-based proximity service is a distance preserving service, notifying users about other nearby users and showing their distances. According to the strength of social relation in the social networks, location based proximity services are classified into *contact-based nearby friends notifications* (e.g., Swarm, Facebook Place, PCube, etc.) and *distance-based social discoveries* (e.g., Wechat, Tinder, Momo, etc.). Many of these services have already been adopted and used by millions of users, and the number keeps growing steadily.

Contact-based nearby friends notifications (called nearby friends for short) are based on the prerequisite that two users are friends mutually. However, distance-based social discoveries (called social discoveries for short) establish the connection of two users based on the physical proximity, even though the two users may not know each other before. For example, assume that a new user just creates an account on Tinder, a popular mobile dating service. The profile of the new user, such as sexual orientation, marital status, salary etc., is visible to any Tinder user who is in the proximity instantly. Therefore, privacy protection is critically needed for social discoveries.

In this paper, we focus on the location privacy protection in social discoveries. Unfortunately, the existing social discoveries are risking the user's location privacy. It is reported that some government had used a mobile dating app to locate and imprison gay users [11]. It has also been proved theoretically and experimentally that social discoveries can render users vulnerability to trilateration attacks and distance-free locating attacks [7], [4], [3], [11]. More sadly, employing the current workflow, it is impossible to provide a safe social discovery service for each user. From the perspective of data management, the existing social discovery service is a typical scenario for k nearest neighbors queries in a constrained space (i.e., the proximity region specified by a user). It has been proved in the recent work [16] that the existing methods for secure nearest neighbor (SNN) queries [6], [14] are not secure, and it is impossible to construct SNN in standard security models [2].

We use an example in Fig. 1 to illustrate the case of location disclosure in social discoveries. Assume that u, u_1, u_2 and u_3 are social discovery app users (e.g., mobile dating). u_1 is interested in u but u refuses to meet u_1 in person. u_1 asks his friends u_2 and u_3 for help, and u is shown to be nearby to these three malicious users (i.e., u_1, u_2, and u_3) concurrently. By utilizing the precise locations and proximity distances of u_1, u_2 and u_3, they can infer that u must locate within the gray area (i.e., the shared proximity regions by u_1, u_2 and u_3) in Fig. 1. If u_1, u_2 and u_3 continue to change their locations, the gray area can shrink to an exact location. A software agent has been developed in [4] to conduct the above attack automatically. Users in a famous social network was pinpointed within 5 meters of their exact locations [11]. An exact location exposure can cause serious problems, such as unwanted advertisement, location-based spams/scams, and even stalking. To differentiate the roles of user u from u_1, u_2, and u_3, u is called a *discovered user*, while u_1, u_2 and u_3 are called *seeking users*. Generally, a user can be a discovered user, a seeking user, or both.

In order to protect user's location in social discoveries, we propose a novel framework DistSD (Distance-based Social Discovery with Personalized Posterior Screening). The intrinsic of DistSD is that a discovered user can show up in front of seeking users selectively, instead of being discovered passively. We illustrate our basic idea with the same example in Fig. 1.

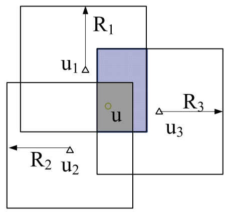


Fig. 1. Location disclosure

User u can choose to be noticeable or not be noticeable by u_1, when u enters u_1's proximity region (PR). On the contrary, in current workflow, the information that u is nearby is sent to u_1, u_2, and u_3 without u's consent or awareness. A nearby record (i.e., u_1 is notified that u is nearby) represents the proximity information that u is near u_1. Moreover, u can choose a subset of nearby friends (e.g., u_1 and u_3 only) to be notified of her appearance. If u appears on the nearby friends list of u_1 and u_3, from the perspective of service results, u's location is inferred to be the blue region (i.e., the shared PRs by u_1 and u_3) in Fig. 1. If u accepts the blue region size, u_1 and u_3 will be notified that u is nearby. DistSD is personalized to screen each discovered user *after* service requests are answered, which is called as a *posterior screening*.

However, we face three main challenges in implementing DistSD with posterior screening. First, the posterior screening mechanism puts us in a dilemma. In a sense, the new workflow in DistSD seems like a reverse range query, in which a discovered user (i.e., u) asks which seeking user's (i.e., u_1, u_2 or u_3) PR covers the discovered user. A service result (i.e., u_1 and u_3) cannot be obtained without the user's location, whereas the user's location is protected from the perspective of the service result. Second, location protection and quality of services (QoS) are a pair of conflicting requirements. As we will show in Section IV, finding an optimal group of seeking users is NP-hard. Thus, how to find a safe seeking users set with a maximum group size effectively and efficiently in mobile devices is challenging. Third, it is impractical to make discovered users notify numerous nearby seeking users manually. How can service providers (SPs) notify seeking users automatically without inferring additional information?

In our proposed framework DistSD, a user doesn't send any location to SPs. The SP computes a user's PR from the user's nearby records. Since the safety of the user location has been verified on user's mobile devices beforehand, the computed PR will not violate the owner's location privacy requirement. The SP organizes all PRs into an index (e.g., HGrid or DGrid). Mobile devices download a fraction of the index periodically. With PRs information, a discovered user can notify seeking users of her appearance selectively and automatically by any of the two proposed algorithms, FBuck and LCF^2. Before connecting to seeking users, posterior privacy screening is conducted to insure that the discovered user's location is protected from seeking users and SPs. No one knows a user's precise location except the user herself.

The contributions of this paper are summarized as follows.

- We propose a novel framework DistSD, in which a user can determine whom she may share her proximity with, rather than automatically give the location and thus leak private information.
- We propose two algorithms FBuck and LCF^2 in mobile devices to achieve a balance between QoS and location privacy protection.
- We propose a posterior screening mechanism in distance-based social discoveries, which aims to protect the location privacy on the basis of service results.
- A series of experiments is conducted on a real dataset to evaluate the performance of our proposed algorithms.

The rest of the paper is organized as follows. We review the related work in Section II. The new framework DistSD is given in Section III. Section IV formally defines the problem. Section V sketches out the basic idea of the proposed algorithms. Two privacy-enhanced social discovery algorithms FBuck and LCF^2 are proposed in Section VI and Section VII respectively. Section VIII presents the performance evaluation results. Finally, the paper is concluded in Section IX.

II. Related Work

A great of research efforts [8], [10], [1], [12] have been devoted in investigating location privacy protection, that preserves the privacy of mobile users and ensures the high quality of LBSs. We categorize the existing location privacy protection mechanisms into a *priori protection* and a *posterior screening*. In a priori protection mechanism, an actual location is replaced with an obfuscated location *before* the query is issued. The mainstream idea for location obfuscation [8], [10], [1] includes generalization (e.g., spatial cloaking), cryptography (e.g., PIR), generating dummies, and adding noise (e.g., geo-indistinguishability). A priori privacy protection has been a very active research area in points of interests retrieval in LBSs in the last decade. In contrast, in a posterior screening mechanism, the user's location is protected from a query result perspective *after* the query is answered. The obfuscated location for each user is an uncertain region in our method. However, the obfuscated location is computed from the service results (i.e., nearby records) instead of being sent by the user themselves. Thus, our work falls into the category of a posterior screening.

Privacy protection on nearby friends has been extensively studied in recent years [9], [13]. The basic idea is to allow two friends to determine if they are in proximity based on both spatial cloaking and encryption, which falls into the category of a priori protection. One drawback of the existing methods is that a proximate distance and a distance ranking list between friends are revealed, which can result in location disclosure [7]. Another one is that each user needs to share a symmetric secret key with each of her friends. However, all nearby users are considered as friends in social discoveries. The set of friends changes dynamically, and the number of potential friends for each user is large. Sharing a secret with each friend dynamically seems to be impractical [9]. Thus, existing privacy protection methods on nearby friends cannot be applied in social discoveries.

III. Framework of DistSD

We employ a client-server architecture, including users and SPs. None of the entities in the service space is trusted. Thus, like most existing work [16], [6], we assume SPs and users are semi-honest, who are honest but potentially curious. Since the SP is un-trusted, the form of a user's obfuscated location owned by the SP has two options, i.e., an encrypted location or an uncertain region. Users may prefer encrypted locations, which indicate a strong privacy protection. However, to the best of our knowledge, secure reverse range queries, that the problem of finding encrypted regions where a user locates, is still an open issue. Thus, a safe uncertain region is used as a user's obfuscated location in this paper. Note that, the obfuscated location of a user is computed by the SP from nearby records instead of being sent by the user herself.

SPs store users' registration information and seeking users' PRs. Each PR is extended from the user's obfuscated location. The SP organizes PRs as a grid based index (details in Section VI-A and Section VII-A). Before using the social discovery service, each user personalizes the minimum size of obfuscated locations at the mobile device. Only the user herself knows the size. The minimum uncertain size indicates the maximum acceptable area, which the user allows other un-trusted entities to know, e.g., SPs or curious friends. Users in DistSD (i.e., mobile devices) download a fraction of PRs of seeking users periodically. When a user uses the social discovery service, the user aims to meet local new friends, that implies a seeking user will stay at a location for a while. Thus, we assume seeking users don't update locations frequently.

When a user opens a social discovery app, a list of selective nearby seeking users, whose PR covers the user's location, is recommended to the user by running any of the proposed methods in mobile devices. The user's location privacy is posterior screened from the recommended seeking users. The proximity information in the form of nearby records is sent to the SP automatically. Based on the nearby records, the SP notifies the corresponding seeking users that the discovered user is nearby. Then, an obfuscated location of each discovered user is inferred by the SP as the attack scenario described in Fig. 1. The safety of the obfuscated location has been verified in mobile devices (i.e., in our proposed methods) beforehand. The SP extends the obfuscated location to a PR as the user's registration information. We use a rectangle for representing a PR to simplify computation. The grid based index is updated when a new seeking user with her correspond PR is inserted. Each user has a valid time interval T, which is a system parameter. If a user has no actions within T, e.g., being notified or chatting, the user's log-in information will expire and be deleted from the SP.

Note that a PR is created from the user's nearby records indirectly. At the initial phase, the system will encounter a cold startup problem. Since none of the users has nearby records initially, the SP can get neither obfuscated locations nor PRs. We resolve this problem in two ways. One method is to employ venues check-in records in check-in services, where a check-in record corresponds to a proximity record. Many nearby friends services and venues check-in services are combined in one app, such as Foursquare, Google+, and etc. It is possible to borrow venues check-in records to start the system. The other method is to employ a prior privacy protection method to obtain an obfuscated location, which is prevented from background knowledge attacks [12].

In summary, the work flow of DistSD is as follows. *At the client side*: (1) The user specifies a minimum size of an obfuscated location as the privacy requirement in the mobile device. (2) A fraction of seeking users' PRs are downloaded periodically. (3) A list of posterior-privacy-screened seeking users is recommended. The user can select to notify the seeking user manually or automatically. (4) The nearby records are sent to the SP. *At the SP side*:(1) The SP notifies the seeking users in the nearby records that the discovered user is nearby. (2) The SP infers an obfuscated location for each discovered user from the nearby records. (3) When a discovered user changes to be a seeking user, a PR is computed based on the obfuscated location, and the gird based index is updated. (4) When a user has no actions for T, the user is deleted from DistSD.

Safety analysis: In our work, the protection target is the user's locations. Most of social discovery services require users to use real name for safety considerations. Thus, we don't consider identity anonymization. *From the SP side:* The locations of a user, that SPs have, include both an obfuscated location and a PR. Since the latter one is extended from the former one, the obfuscated location is the most accurate location that the SP knows. Recall that an obfuscated location is computed from the nearby records of the user. The safety of the obfuscated location has been ensured with posterior screening mechanism in our proposed methods before the nearby records are sent to the SP. Thus, the location privacy is protected from the semi-honest SPs.

From the users side: For a discovered user, the worst case is that all the notified seeking users collude with each other. In such a case, the information that the corrupted users can gather is just the nearby records known by the SP. The exact location of the discovered user can neither be broken down by the SP as the above analysis, nor by the corrupted seeking users. Meanwhile, the seeking user's own locations cannot help them to reduce the uncertainty of the discovered user's location. Thus, the accurate location of a discovered user is protected from seeking users. On the other hand, if a discovered user is malicious, the discovered user can only get the PR of a seeking user, which satisfies the seeking user's privacy requirement. In another word, no one knows a user's precise location in DistSD except the user herself.

IV. Preliminary

A. Attack model

Definition 1. *(Footprints) A set of user u's nearby records is formalized as* $\{\langle u, u_1\rangle, \langle u, u_2\rangle, \ldots, \langle u, u_n\rangle\}$. $\langle u, u_i\rangle$ *indicates*

TABLE I
INTERPRETATION OF SYMBOLS

term	description
pt	the smallest intersection rectangle of PRs
U_{pt}	the set of users related to pt
pr_u	the proximity region of user u
FS_u	the footprints set of a discovered user u
min_u	u's minimum uncertain requirement
OL_{FS_u}	the obfuscated location of u w.r.t. FS_u
loc_u	the location of user u
l, r, b, t	the left, right, bottom and top boundary of a rectangle
δ	the cell size of a DGrid G
ur	a user rectangle
rc	the tightest m-rect where a user locates
dv_{u_i}	the minimum perpendicular distance from u to the rectangle boundary determined by u_i
RM_d^{ty}	the number of remaining related users locating at d far away after ty is opened

u is within u_i's PR, i.e. $loc_u \in pr_{u_i} (1 \leq i \leq n)$, where loc_u is the location of u, and pr_{u_i} is u_i's PR. u is called a discovered user*, and u_i is called a* seeking user*. Seeking users $u_1, u_2, \ldots, u_n$ constitute a* footprints set *FS_u of the discovered user u.*

For the example in Fig. 1, u_1, u_2, and u_3 can be notified that u is nearby without considering location leakage. Then, $FS_u = \{u_1, u_2, u_3\}$. From the view of social discovery services, the users in FS_u are the service results of u. Before defining the attack formally, we give the definition of an obfuscated location from a service result perspective.

Definition 2. *(Obfuscated Location w.r.t. Footprints) Let FS_u be u's footprints set. Each user $u_i \in FS_u$ is associated with a PR pr_{u_i}. The* obfuscated location of u w.r.t. FS_u *is defined as $OL_{FS_u} = \{p | p \in \bigcap_{u_i \in FS_u} pr_{u_i}\}$.*

As the example in Fig. 1, the gray region is u's obfuscated location w.r.t. $FS_u = \{u_1, u_2, u_3\}$.

Recall the attack scenario in Section I, we observe that over-share is an important factor of the location leakage. Without u_1's friends support (i.e., u_2, u_3), u's location cannot be easily found out. A large number of friends based on physical proximity in social discoveries worsen the over-share problem. Thus, we define over-share discovery attacks as follows.

Definition 3. *(Over-share Discovery Attacks) Let OL_{FS_u} be u's obfuscated location w.r.t. the footprints set FS_u. min_u is the minimum uncertain size (i.e. the privacy requirement) specified by u. If size(OL_{FS_u}) $< min_u$, we regard u suffers from an* overshare discovery attack.

In order to prevent overshare discovery attacks, our objective is to find a footprints set FS_u for the discovered user u, such that the size of OL_{FS_u} is not less than min_u. Like most existing privacy protection work [8], we specify the size of OL_{FS_u} as the area of the obfuscated location.

B. Problem Definition

Theorem 1 proves the maximum users that a user u can notify without considering location privacy. For convenience, we first define pure rectangles before the specific proof.

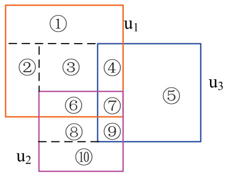


Fig. 2. Pure rectangles

Definition 4. *(Pure Rectangles) A* pure rectangle *pt is a rectangle that hasn't been crossed by the boundaries of any PRs. The set of* users related to *pt is represented as $U_{pt} = \{u | pt \subseteq pr_u\}$.*

Pure rectangles can be obtained through partitioning the PRs along certain borders of PRs. The users U_{pt} related to pt are the users whose PR covers pt. Fig. 2 shows the PRs of u_1, u_2, and u_3 are divided into 10 pure rectangles. The users related to the pure rectangle labeled in ⑦ are $\{u_1, u_2, u_3\}$.

Theorem 1. *Let pt be the pure rectangle where u locates, U_{pt} be the users related to pt, and FS_Set be the set of all possible footprints set of u. For $\forall FS \in FS\_Set$, $FS \subseteq U_{pt}$.*

Proof. We prove it by contradiction. Assume $\exists FS' \in FS\_Set$, but $FS' \nsubseteq U_{pt}$. It must $\exists u_i \in FS'$, but $u_i \notin U_{pt}$. Since $u_i \in FS'$, $loc_u \in pr_{u_i}$. Since $loc_u \in pt$ and pt is a pure rectangle, $pt \subseteq pr_{u_i}$. As Definition 4, $u_i \in U_{pt}$. That is contradictory to our assumption. Proof done. ☐

In the best case, if the area of pt is not less than min_u, U_{pt} is just the target we aim to find. However, we consider the opposite case that the area of pt is less than min_u. We observe that the obfuscated location of a user will become larger when a user notifies a subset of U_{pt}. In the above example, if u notifies $\{u_1, u_3\}$, or $\{u_1\}$, etc., min_u may be satisfied. The objective of our problem can be formally described as follows.

Goal: We aim to find a users set sb_u for a discovered user u, such that (1) $sb_u \subset U_{pt}$; (2) $size(OL_{sb_u}) \geq min_u$; (3) $\forall sb'_u \subset U_{pt}$ $(sb'_u \neq sb_u)$ *and* $size(OL_{sb'_u}) \geq min_u, |sb_u| \geq |sb'_u|$, where $|sb_u|$ is the users number in sb_u. The third condition indicates sb_u is one of the largest subsets of U_{pt} that satisfies the first two conditions.

Theorem 2. *Finding a maximal subset sb_u in U_{pt} with the constraint $size(OL_{sb_u}) \geq min_u$ is NP-complete.*

Our problem can be exactly mapped to a classical frequent item set selection problem by treating U_{pt} as the complete item set and sb_u as the selected subset (whose occurrence frequency is $size(OL_{sb_u})$). min_u corresponds with the minimum frequency requirement. The problem of counting the number of the maximal min_u-frequent itemset in U_{pt} has been proven to be #P-complete in [15], i.e., the correspond maximal frequent itemset mining problem is NP-complete. Therefore, our problem is also NP-complete. For convenience, we list the interpretations of primary symbols throughout the paper in Table I.

V. Proposed Algorithms Sketch

In order to achieve the above goal, our basic idea is to divide the task into two parts. One part is to organize PRs on the SP into an index, which is downloaded by mobile devices periodically. With the help of a PR index, a mobile user can find the covered PRs FS_u efficiently. The complete contents of a user in SPs include numerics, texts, images, graphs (e.g., friends circles), and even videos, whose size could be significant. It is impractical to download the complete contents due to the constrained network bandwidth, device battery, etc. Thus, we propose two grid-based indices (i.e., HGrid and DGrid in Section VI-A and Section VII-A respectively) to organize the PRs in SPs.

The other part is to find a maximum safe footprints in mobile devices, which is the core of our proposed methods. The basic process of finding a maximum safe footprints is as follows. With the index downloaded from SPs and the user's own exact location, the users FS_u whose PR covers the mobile client's location are found. Then, from this subset of FS_u, the corresponding intersection area of their PRs are computed. Next, the intersection area is verified by the user's uncertain requirement min_u. If min_u is not satisfied, another subset of FS_u is checked. Finally, a maximum subset of FS_u whose PRs intersection area is not less than min_u is returned as the safe footprints.

Baseline: The naive method for finding the maximum safe subset of FS_u is the brute-force calculation. Specifically, we enumerate each user combination in FS_u with decreasing size of users set, and compute the intersection area of their PRs. As we know, enumeration each combination of a users set is costly. As the number of seeking users in FS_u increases (e.g., several thousands in our experiments), the efficiency of baseline will deteriorate. In order to improve the efficiency, we propose two grid-based algorithms: a global algorithm FBuck (in Section VI-B) and a local algorithm LCF^2 (in Section VII-B).

VI. FBuck: Global Footprints Finding Algorithm

A. HGrid

A HGrid on the SP contains a grid and a hash table. The grid just stores the seeking users identities, which helps mobile clients to find the seeking users whose PR covers the client's location. Each cell in the grid maintains two linked lists: (1) pl: a list that tracks the users whose PR covers the cell partly. (2) fl: a list that stores the users whose PR covers the whole cell. Meanwhile, a hash table saves the location of a user's PR. Each item in the hash table is (id, x_1, y_1, x_2, y_2), where id is the seeking user identity, (x_1, y_1) and (x_2, y_2) are bottom-left and top-right coordinates of the user's PR respectively.

B. FBuck

Incorporating with the HGrid and the user location (x, y), the users set FS_u whose PR covers (x, y) are found. Different from Baseline, we employ top items in four buckets to compute the intersection area for each user combination from FS_u.

We first show the data structure of the bucket, and then give the main idea of the algorithm. For simplicity, we name four borders of a rectangle as left, right, bottom and top. Each bucket represents one kind of PR bounds, that is, a left bucket l_buk, a right bucket r_buk, a bottom bucket b_buk, and a top bucket t_buk. The item in each bucket includes a user id and a boundary coordinate pos of the PR on the x-axis (for l_buks and r_buks) or the y-axis (for b_buks and t_buks). Using pos as the key, l_buk and b_buk are sorted non-increasingly, and r_buk and t_buk are sorted non-decreasingly. Fig. 3(a) shows an example of four buckets for three users' PRs.

We observe that the top items in the four buckets constitute the smallest intersection rectangle (i.e., a pure rectangle) at present. Based on this observation, the main idea is as follows. The top items in the four buckets constitute a rectangle ur. If $ur.area < min_u$, a user du is selected among the four top items, and is deleted from each bucket. The selection strategy for user removing is in the next subsection. The new top items shape into a new intersection rectangle. If the new rectangle area is still less than min_u, the above steps are repeated until every bucket becomes empty or the area is not less than min_u. Algorithm 1 shows the details of FBuck.

Algorithm 1 FBuck
```
 1: find the covering users $FS_u$ in the cell $c$;
 2: init $l\_buk$,$r\_buk$,$b\_buk$,$t\_buk$ by the PR of each $u_i$ in $FS_u$;
 3: sort items in $l\_buk$ and $b\_buk$ non-increasingly;
 4: sort items in $r\_buk$ and $t\_buk$ non-decreasingly;
 5: **while** $l\_buk$ is not empty **do**
 6:     get the top item from each bucket;
 7:     a rectangle $ur$ is formed from the top items;
 8:     **if** $ur.area < min_u$ **then**
 9:         $du$=the user selected to be removed;
10:         delete $du$ from the four buckets and $FS_u$;
11:         re-sort $l\_buk$, $r\_buk$, $b\_buk$, $t\_buk$;
12:     **else**
13:         **return** remaining users in $FS_u$ as the footprints;
```

Fig. 3 shows a running example of FBuck. The user u (i.e., the small triangle in Fig. 3) is covered by the PRs of $\{u_1, u_2, u_3\}$. Initially, the top items in the four buckets constitute a rectangle ABCD. The users in the current top items are $\{u_1,\ u_2,\ u_3\}$. If the area of ABCD is less than min_u, u_3 is assumed to be deleted from each bucket. Then, each bucket is shown in Fig. 3(b) after u_3 is deleted. Now, the new top items constitute a new rectangle EFGC. If the area is not less than min_u, $\{u_1,\ u_2\}$ is returned as the safe footprints set.

1) Removing Users Selection: Our basic user removing idea is to delete the user whose PR borders limit or will limit the size of the obfuscated location, such that the uncertain requirement will be satisfied as early as possible. Before specifying the selective metrics, we define two distances (i.e., inner distances and outer distances) between a discovered user u and a seeking user u_i.

Definition 5. *(Inner Distance) Let FS_u be a seeking users set, pt be the intersection rectangle of PRs of users in FS_u,*

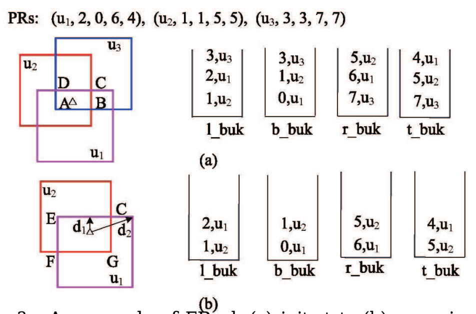


Fig. 3. An example of FBuck (a) init state (b) removing u_3

and the boundary users of pt *be* $FS_u^b(\subseteq FS_u)$. *Assume that a discovered user* u *locates in* pt. *For* $\forall u_i \in FS_u$, *the inner distance*

$$indist(u, u_i) = \begin{cases} dv_{u_i}, & if\ u_i \in FS_u^b, \\ \infty, & if\ u_i \in FS_u - FS_u^b. \end{cases} \tag{1}$$

where dv_{u_i} *is the minimum perpendicular distance from* u *to the* pt *boundary determined by* u_i.

A small $indist(u, u_i)$ implies one of u_i PR's boundaries limits the size of pt. If a user u_i with the small inner distance is removed earlier, the intersection area of PRs tends to become larger.

Definition 6. *(Outer Distance)* $p_{u_i}^{bl}$, $p_{u_i}^{br}$, $p_{u_i}^{tl}$, $p_{u_i}^{tr}$ *are the four corner points of* u_i*'s PR* ($u_i \in FS_u$)*, that is the bottom-left corner, the bottom-right corner, the top-left corner and the top-right corner respectively. For* $\forall u_i \in FS_u$, *the outer distance*

$$outdist(u, u_i) = \underset{p \in \{p_{u_i}^{bl}, p_{u_i}^{br}, p_{u_i}^{tl}, p_{u_i}^{tr}\}\ \wedge\ p \notin pt}{MIN} (dist_{pu})$$

where $dist_{pu}$ *is the distance between the location of* u *and the point* p.

A small $outdist(u, u_i)$ indicates u_i's PR may limit the size of the obfuscated location later. In other words, a small $outdist(u, u_i)$ implies a small available space to increase the intersection area. Thus, the users with small outer distances are expected to be removed earlier. We take u and u_1 in Fig. 3(b) as an example. indist(u, u_1)= d_1, and outdist(u, u_1)= d_2.

The options for selecting a removing user (line 9 in Algorithm 1):

Option 1: A user with the minimum inner distance $indist$ is removed.

Option 2: A user with the minimum outer distance $outdist$ is removed.

Option 3: A user with the minimum $indist \times outdist$ is removed.

VII. LCF^2:Local Cell Footprints Finding Algorithm

In FBuck, the whole PRs are downloaded by mobile clients. The communication cost can be high. We observe that a city is usually divided into independent districts on the basis of different urban functions, geographical position, etc., which is far greater than location uncertain requirements. For example, Beijing contains 16 districts. In order to reduce the communication cost, we propose LCF^2, which downloads a part of PRs in a district, and finds a safe footprints set from local PRs. The problem of how to divide a city is out of our scope.

A. DGrid: A Grid with Corner-Rect Departure Cells

We still use a grid to index the PRs. However, the PRs' location information is stored in the cells of the grid.

1) Cells of a DGrid: Before elaborating the feature of the DGrid, we first define the cell size. We observe that a closed region is formed when a left l meets a right r on the x-dimension, meanwhile, a bottom b meets a top t on the y-dimension. This phenomena still holds when the four borders belong to different users. The cell size is defined as the minimum distance, at which an l meets an r or a b meets a t when x-coordinates and y-coordinates increase respectively. All PR's x-coordinates (y-coordinates) of l and r (b and t) constitute a set PR_l and PR_r (PR_b and PR_t) respectively. PR_l, PR_r, PR_b and PR_t are sorted ascendingly.

Definition 7. *(Cell Size) The* cell size δ *of a grid* G *is* $\delta = MIN(\delta_x,\ \delta_y)$, *where* $\delta_x = \underset{\forall x_l \in PR_l, x_r \in PR_r, x_r > x_l}{MIN} (x_r - x_l)$, $\delta_y = \underset{\forall y_b \in PR_b, y_t \in PR_t, y_t > y_b}{MIN} (y_t - y_b)$. G *is called a* DGrid.

For a cell c in a DGrid, if an r and an l cross c at the same time, the r must locate at the left of l. Similarly, if a t and a b are in c simultaneously, the t must be below the b. In Fig. 4, the dash square represents a cell. Fig. 4(a) shows the cases when four kinds of PR's borders cross c simultaneously, and Fig. 4(b) shows a case which never occurs in a DGrid.

A corner rectangle is formed when a PR intersects with a cell. For instance, a bottom-right corner rectangle (br-rect) is formed when a PR intersects with a cell partly by a PR bottom and a PR right. As shown in Fig. 4 (a), the corner rectangles (i.e., br-rects, bl-rects, tr-rects, and tl-rects) in a cell depart from each other. This property is formalized in Theorem 3.

Theorem 3. *For any cell* c *in a DGrid* G, *if the right of a PR* pr_u *and the left of another PR* $pr_{u'}$ *cross* c *simultaneously,* $pr_u.x_r \leq pr_{u'}.x_l$; *otherwise, the right of* pr_u *and the left of* $pr_{u'}$ *must locate at the right and left boundaries of* c *respectively, where* $pr.x_l$ *and* $pr.x_r$ *are the x-coordinates of* pr*'s left and right respectively. Similarly, if the top of* pr_u *and the bottom of* $pr_{u'}$ *cross* c *simultaneously,* $pr_u.y_t \leq pr_{u'}.y_b$; *otherwise, the top of* pr_u *and the bottom of* $pr_{u'}$ *must locate at the top and bottom boundaries of* c *respectively, where* $pr.y_b$ *and* $pr.y_t$ *are the y-coordinates of* pr*'s bottom and top respectively.*

Theorem 3 can be proved by contradiction. Due to space limitation, the proof is omitted.

Definition 8. *(*m*-rects) An* m-rect *is the intersection of a user's PR and a cell* c. m *is the number of* m*-rect boundaries determined by a user's PR. From Theorem 3,* $m \in \{0, 1, 2\}$. *The other* $4 - m$ *boundaries are cell borders.*

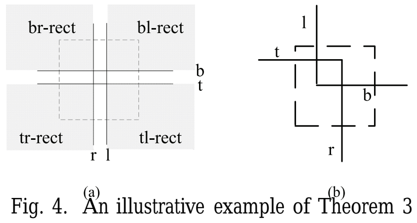


Fig. 4. An illustrative example of Theorem 3

If m=0, the PR fully covers the cell c; otherwise, the PR partly covers c. Fig. 5 (a) shows a DGird with three PRs. The m-rect in the cell (6, 6) is a 0-rect, whose borders are all the cell's boundaries. On the other hand, a rectangle, whose borders are all the boundaries of user's PRs, is called a user rectangle.

A cell contains several m-rects.

Definition 9. *(Information in a Cell) Let G be a DGrid. The information in each cell $c \in G$ is formalized as $(pos,\ rects)$. $pos = (c_x,\ c_y)$ is the location index of the cell in G. $rects$ is a collection of m-rect$=(ty,\ ids,\ dist)$.*

- *If $m = 0$, ids are user identities whose PR covers c fully. ty and $dist$ are both null.*
- *If m=1, $ty \in \{l,\ r,\ b,\ t\}$ represents that a left, right, bottom or top border of PR crosses c; ids are the users whose PR's ty boundary crosses c ; $dist$ is the width wid of m-rect if $ty \in \{l,\ r\}$ or the height hgt of m-rect if $ty \in \{b,\ t\}$.*
- *If $m = 2$, $ty \in \{bl,\ br,\ tl,\ tr\}$ represents a bottom-left, a bottom-right, a top-left, or a top-right corner rectangle in c, as Fig. 4(a) shows. $ids = (id_{lr},\ id_{bt})$, where id_{lr} (id_{bt}) are users whose PR's left or right (bottom or top) crosses c. $dist = (wid,\ hgt)$, where wid (hgt) is the width (height) of m-rect.*

In Fig. 5 (a), the cell (5, 7) has three m-rects, i.e., an l_rect, a t_rect, and a tl_rect.

2) Inserting and Deleting Users: When a user u is deleted from a DGrid G, the cells which are covered by pr_u are found. Then, u is deleted from the corresponding m-rects in the cells. When a new user u is inserted into G, the insertion methods for fully covered cells and partly covered cells are different. For fully covered cells, u is inserted into 0-rects. For partly covered cells, there is a case that an (a) l (b) meets an (a) r (t) in c, which will violate Definition 7. In this case, the affected left (right, bottom, or top) of a user's PR is shifted to the left (right, bottom, or top) boundary of the cell. In other words, an affected PR border is extended by no more than δ. Then, the user is inserted into the corresponding m-rects. The detailed insertion and deletion algorithms are omitted due to the space limit.

B. LCF^2

LCF^2 employs the m-rects in local cells to find the safe footprints. The basic idea is to extend the m-rect in the cell, which the user is in, by the proper m-rects in neighbor cells, such that a safe user rectangle is formed. The users related to the safe user rectangle are the safe footprints.

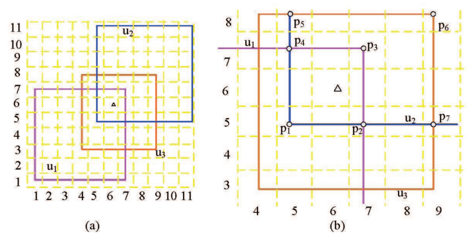


Fig. 5. DGrid

The main procedure is as follows. LCF^2 starts from a tightest m-rect where the user is (details in Section VII-B1). The m-rect is extended to be a user rectangle ur by applying an extending rule (details in Section VII-B2). If $ur.area < min_u$, ur is inserted into a stack st. While st is not empty, the top item tm is got. According to a boundary selection rule (details in Section VII-B3), one border of tm is opened. According to the extending rules at opening points (details in Section VII-B2), a new user rectangle nur is found. If $nur.area \geq min_u$, the users related to nur are returned as the safe footprints; otherwise, nur is inserted into st, and the above steps are repeated.

We use an example to illustrate the main idea. To be clear, we zoom in the cells ($c_x \in [4, 9]$ and $c_y \in [3, 8]$) in Fig. 5(a) and show them in Fig. 5(b). Assume that the user u locates in the cell (6, 6). A 0-rect is found. As the extending rule for a 0-rect (i.e., Rule 0), an r-rect and an l-rect are found in the cell (7, 6) and the cell (5, 6) respectively. A t-rect and a b-rect are found in the cell (6, 7) and the cell (6, 5) respectively. Then, four borders of a user rectangle are fixed $ur = p_1p_2p_3p_4$. If $ur.area < min_u$, ur is inserted into a stack. Now, the top item in the stack is $p_1p_2p_3p_4$. As an opening boundary selection rule, p_3p_4 is assumed to be opened. It means u_1 is removed, which results in p_2p_3 being opened as well. With u_1 being removed, ur changes to a special bl-rect, which traverses more than one cell. As the extending rule at opening points, a new user rectangle $nur = p_1p_7p_6p_5$ is got. Assume $nur.area \geq min_u$, u_2 and u_3 related to $p_1p_7p_6p_5$ are returned as the footprints. LCF^2 is shown in Algorithm 2.

Algorithm 2 LCF^2:Local Cell Footprints Finding Algorithm

```
1: locate the tightest $m$-rect $rc$ using $u$'s location ($x$,$y$);
2: $rc$ is extended to $ur$ by applying extending rules;
3: insert $ur$ into a stack $st$;
4: while $st$ is not empty do
5:     $ur$=get the top item from $st$;
6:     if $ur.area \geq min_u$ then
7:         return users related to $ur$ as the footprints;
8:     if all four borders have been opened then
9:         pop the top item from $st$;
10:    else
11:        open the boundary that hasn't been opened;
12:        extend the opened rectangle to a new user rectangle $ur$;
13:        insert $ur$ into $st$;
```

We continue to illustrate the details of LCF^2 through answering four questions: (1) Which m-rect is tightest when a user locates in more than one m-rect? (2) What information of a user rectangle is saved in a stack? (3) What are the extending rules for finding a user rectangle? (4) If min_u is not satisfied, which boundary is selected to be opened?

1) The Tightest m-rect and The Stack: Since more users are expected to be returned as the safe footprints, we define the priority order of m-rect as 2-rect>1-rect>0-rect. When a user locates in more than one m-rects, a m-rect with the highest priority is selected.

A user rectangle ur is inserted into a stack if $ur.area < min_u$. The stack is in a decreasing order with the number of users related to ur as the key. Each item ur in the stack includes: (1) the bottom-left and top-right coordinates of ur; (2) users related to ur; (3) a bitstring representing whether the left, right, bottom or top boundary has been opened. When four boundaries have been opened, ur is popped from the stack.

2) Extending Rules: With the aim of locating new boundaries, the extending rules are used in two cases: one is to find a user rectangle ur for a m-rect; the other is to find a new user rectangle ur when some borders are opened. The extending rules in the second case can be revised from the ones in the first case. Thus, we show the extending rules of m-rects first.

Let rc be the tightest m-rect where a user locates. The basic idea of extending is to find the corresponding types of m-rects to locate a new left, right, bottom or top user boundaries in neighbor cells. Due to Theorem 3, not every m-rect in a cell needs to be checked. We only search the *proper* m-rects. Taking 0-rects as an example, to locate a right bound, r-rects, br-rects, and tr-rects are supposed to be checked. However, in our method, only r-rects are used to locate the new right bound (details see Rule 0). That is because neither br-rects nor tr-rects will occur before an r-rect is encountered. This point is assured by Theorem 3, which can be proved by contradiction.

Rule 0:(**0-rects**) Let the cell where the user locates be (c_{x_0}, c_{y_0}). Starting from $p = c_{x_0} + 1$ ($p = c_{x_0} - 1$), the first r-rect (l-rect) found in the cell (p, c_{y_0}) determines the right (left) border of ur when p increases (decreases) by 1. Symmetrically, starting from $p = c_{y_0} + 1$ ($p = c_{y_0} - 1$), the first t-rect (b-rect) found in the cell (c_{x_0}, p) determines the top (bottom) border of ur when p increases (decreases) by 1.

For an m-rect (m=1 or 2), m borders of ur have been fixed by the m-rect itself. For instance, the right of ur inherits the right of rc for r-rects (m=1). Thus, the rules for m-rects (m=1 or 2) are used to find the other 4-m unknown borders. The searching idea is similar, which can be revised easily from Rule 0. For space limit, the specific rules for m-rects (m=1 or 2) are omitted.

When a boundary of ur is opened, a new boundary is expected to be found on the opened direction. If more than one border is opened, a ur becomes a special m-rect. The corresponding extending rules for m-rects are applied, but the searching start cell changes to the ones where the opened boundaries located. If only one border is opened, related m-rects are checked on the opened direction. For example, if the right is opened, the first tr-rect, br-rect or r-rect encountered determines the new right. The start cell scanned is the cell where the old right locates.

3) Removing Users Selection: Recall FBuck has three options for removing users selection. However, Option 2 and Option 3 in FBuck cannot be applied in LCF^2, since LCF^2 has no global PRs information. In addition, FBuck removes users one by one. In order to improve the efficiency, we propose three strategies for users removing on the basis of the boundary. Each time a boundary is opened, all users on the boundary are removed together. We revise Option 1 in FBuck as follows on the basis of the boundary.

Option 1: Four borders are opened as the increasing border score. The border score is defined based on the inner distance and the number of boundary users.

Definition 10. *(Score) Let NUM_{ty} be the number of ty boundary users of ur, where $ty \in \{l, r, b, t\}$. The* border score *is defined as $score_{ty} = indist_{ty} * NUM_{ty}$.*

Option 1 aims to find a safe user rectangle earlier. Option 2 intends to obtain a large footprints set. However, it is unknown when min_u will be satisfied, and how many users will remain when different boundaries are opened. Thus, we estimate the number of remaining users in an safe obfuscated location with different boundaries being opened. $d = \frac{min_u - AREA(ur)}{MIN(wid,hgt)}$ is the farthest distance by which ur is extended to achieve min_u. The number of remaining users RM_d^{ty}, who locate at d far away from ur in the ty direction, is counted, where $ty \in \{l, r, b, t\}$. If RM_d^{ty} in any direction is 0, d is divided by 2 until $RM_d^{ty} \neq 0$. Based on this idea, we propose Option 2.

Option 2: The larger RM_d^{ty} is, the sooner ty is selected to be opened.

Option 3: The boundaries are opened in a random order, but the horizontal and vertical directions are selected alternately. For example, a random opening order is $\{r, b, l, t\}$.

VIII. Performance Evaluation

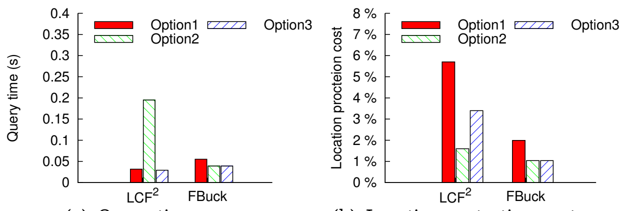
In this section, the effectiveness and efficiency of our proposed algorithms, FBuck and LCF^2, are experimentally evaluated under different system settings. The evaluation metrics include the downloaded PRs size, query time, location protection cost at the client's site, and index updating time at the server site. None of the existing algorithms considers a social discovery service with posterior privacy screening. Hence, we only compare FBuck and LCF^2 with the baseline. The algorithms are implemented in C++ and evaluated on a desktop running Windows 7 with an Intel Core 2.1GHz CPU and 4GB main memory.

To the best of our knowledge, no real large-scale social discovery datasets have been publicly released to date. Therefore, we use a real check-in data crawled from Foursquare [5], which has more than 20 million users and 2 billion check-ins records. We extract the most dense 64*64 km^2 subspace as a service space. Real venues simulate seeking users, and real check-in users are discovered users. A total of 20,238 simulated users are in the space. The cell size of the grid is set to 0.3% of the space width. By default, the proximity range R for each seeking user is 4% of the space width. For each discovered user, the width of location uncertain requirement is 200 *m*.

A. Different Options for Removing User Selection

Different selections on removing users affect the query time and the location protection cost. Query time is the time to find safe footprints at the client site. Location protection cost is how much QoS is sacrificed to protect privacy. The cost is evaluated by (1-$\frac{|FS_u|}{|U_{pt}|}$), where FS_u (U_{pt}) is the footprints set with (without) privacy protection. Based on the experimental results, a default option for removing users selection is set for LCF^2 and FBuck in the following experiments.

For LCF^2, Option 3 shows the best query time in Fig. 6(a). Option 1 is slighter higher than Option 3, whereas Option 2 is the worst. That is because more time is spent by Option 2 on finding the direction with the most remaining seeking users iteratively. In contrast, the location protection cost using Option 2 is minimum among the three options (see Fig. 6(b)). Though the location protection cost for Option 3 is higher than Option 2, less than 2% redundant cost is sacrificed. Considering the tradeoff between short query time and low query cost, we apply Option 3, a random selection, as the default setting for LCF^2.



(a) Query time (b) Location protection cost

Fig. 6. Different removing users selection options

For FBuck, we observe that Option 3 shows the best performance on both query time and location protection cost. That is because Option 3 considers both inner distances and outer distances. Thus, Option 3 is used as the default setting. We also observe that the performance using Option 2 and Option 3 looks similar. That implies the effect of outer distances dominates inner distances. Comparing the performances of Option 1 between FBuck and LCF^2, we can see that the query time of LCF^2 is better than FBuck, since several users are removed when a boundary is opened. In contrast, the location protection cost of LCF^2 is higher than FBuck.

B. Different Minimum Uncertain Requirements

Since Baseline and FBuck both use the HGrid, Baseline is not included in evaluation on size of downloaded PRs and index updating time. The PRs downloaded size indicates the communication cost between mobile devices and SPs. Recall that LCF^2 downloads a partial PRs. In our experiments, we evaluate the downloaded size in the densest sub-space. The size of the sub-space is 100 times to the most constraint uncertain requirement. Fig. 7(a) shows the effect of min_u on the size of downloaded PRs. The downloaded PRs size doesn't change for the two methods. Since the global PRs are downloaded, the downloaded size of FBuck is about two times larger than LCF^2. More sophisticated pruning techniques can be done on the SP site to reduce the download size, which due to space limitation is beyond the scope of this paper.

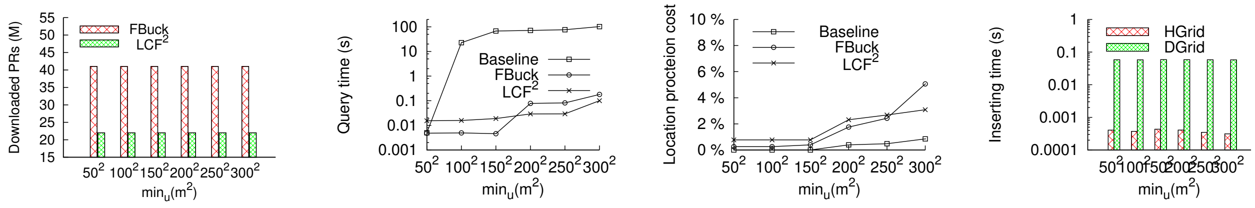
Fig. 7(b) shows the trend of query time at various min_u settings. The actual query time of Baseline is too worse to be shown. In order to improve the efficiency of Baseline, we apply an approximate computation that an anchor is used to represent the users with a group of close PRs. The query time of three methods increases with min_u, since a user requires a large obfuscated location. Among three algorithms, Baseline shows the worst query time even though an approximate computation is employed. When min_u is small (less than 150^2), FBuck is faster than LCF^2, since LCF^2 spends more time on searching proper m-rects in neighbor cells. When min_u continues to increase, LCF^2 becomes the best one. The reason is that, when min_u becomes large, more users are removed to enlarge the intersection area. FBuck iterates to delete the proper users one by one. By contrast, a boundary is opened by LCF^2, which removes several users once.

Fig. 7(c) shows that the location protection costs of three algorithms increase with min_u increasing. Since Baseline finds a maximum footprints set greedily, the location protection cost of Baseline is smallest among three algorithms. Global PRs is used in FBuck. Thus, the location protection cost of FBuck is smaller than LCF^2 at most cases. When min_u increases to 300^2, LCF^2 outperforms FBuck. Generally, global PRs information is helpful to find a safe footprints set with small privacy protection cost.

The index updating time are the insertion time and deletion time at the SP, when a user with PR is inserted into or deleted from the index. For GDrid, the insertion time dominates the deletion time. Recall in a DGrid, certain user's PRs are shifted to the cell boundary due to Theorem 3. As a result, an insertion may lead to several users deletions and re-insertions. Thus, we utilize the average insertion time to show the change trend of the index updating time. Fig. 7(d) shows increasing min_u has no effect on the insertion time. As expected, the insertion time of HGrid is much smaller than DGrid. Even so, the average insertion time of DGrid is about 0.06*s*, which is acceptable for SPs.

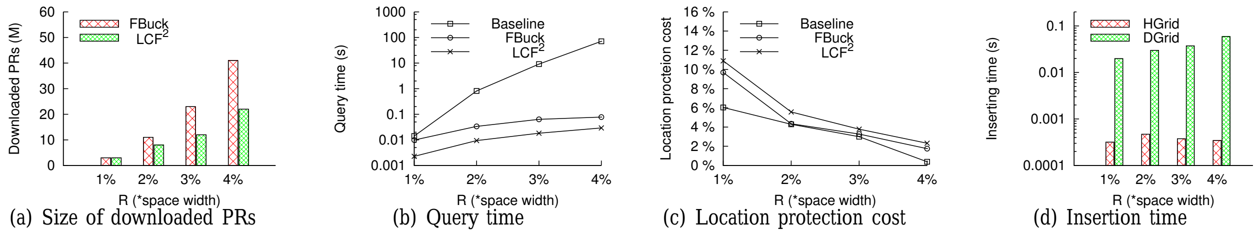
C. Different Proximity Ranges

In this section, the range (R) of PR varies in [1%, 4%] of the space width. Fig. 8(a) shows the changes of downloaded PRs size. In general, the downloaded size increases with R. Increasing R indicates more PRs intersect with each other, and more information needs to be saved in an index. Specifically, an enlarged PR covers more cells, thus the grid size increases for FBuck; the number of m-rects in a cell also increases for LCF^2. When R is 1% of the space width, the downloaded size of FBuck and LCF^2 is similar. When R continues to enlarge, the advantage of partly downloading is obvious. LCF^2 outperforms FBuck. Fig. 8(b) shows the query time of three algorithms increases with R enlarging, since a user locates in more seeking user's PRs. Since Baseline is a greedy algorithm, its query time grows exponentially when R increases. Though



(a) Size of downloaded PRs (b) Query time (c) Location protection cost (d) Insertion time

Fig. 7. Different uncertain location requirements



(a) Size of downloaded PRs (b) Query time (c) Location protection cost (d) Insertion time

Fig. 8. Different ranges of proximity regions R

the query time of FBuck and LCF^2 also increases, their rise rates are very small. When R increases to 4%, the query time of FBuck is 0.08s, while LCF^2 needs 0.03s.

From Fig. 8(c), we observe that the location protection cost of three algorithms decreases with the increase of R. That is because the intersection area of PRs becomes large on average. Then, fewer users are removed from a candidate users set to satisfy the personalized uncertain requirements. For location protection cost, FBuck and Baseline outperform LCF^2 at all settings. When R is small, the cost gap between LCF^2 and FBuck (Baseline) is obvious. When R is large, the gap shrinks. Similar to Fig. 7(d), the insertion time of HGrid is better than DGrid. From Fig. 8(d), enlarging R has little effect on the insertion time for HGrid. The insertion time of DGrid increases sharply. The number of m-rects in a cell increases with R. As a result, DGrid takes more time on affected users deletions and re-insertions.

IX. Conclusion

In this paper, we investigated two privacy enhanced social discovery algorithms with posterior screening. We observed that traditional social discoveries cannot provide secure services in standard security model. To address this problem, we propose a novel framework DistSD, where a discovered user's location privacy is ensured through posterior screening in mobile devices. Two heuristic algorithms (FBuck and LCF^2) are proposed under the new framework. Experimental results show that FBuck has the best efficiency at the server site, but the downloaded size is high. LCF^2 has a small downloaded size and the best query efficiency at client site, whereas the location protection cost is a little high but acceptable. Considering a limited network bandwidth, LCF^2 achieves a good tradeoff between the privacy protection and the query cost.

Acknowledgment

This research was partially supported by the grant from the Natural Science Foundation of China (No. 61303017), the Natural Science Foundation of Hebei Province (No. F2014210068), and the Fourth Outstanding Youth Foundation of Shijiazhuang Tiedao University.

This work is supported in part by NSF through grants IIS-1526499, and CNS-1626432. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X GPU used for this research.

References

[1] M. Andres, N.E. Bordenabe, K. Chatzikokolakis, and et al, *Geo-Indistinguishability: Differential Privacy for Location-based Systems*. In proc. of CCS, pages 901-914, 2013.

[2] A. Boldyreva, N. Chenette, Y. Lee, and et al, *Order-preserving symmetric encryption*. In proc. of EUROCRYPT, pages 224-241, 2009.

[3] J. Farnden, B. Martini and K-K R. Choo, *Privacy Risks in Mobile Dating Apps*. In proc. of AMCIS, pages 13-15, 2015.

[4] A. Fattori, A. Reina, A. Gerino, and S. Mascetti, *On the Privacy of Real-World Friend-Finder Services*. In proc. of MDM, pages 331-334, 2013.

[5] H. Gao, J. Tang, H. Liu, *gSCorr: Modeling geo-social correlations for new check-ins on location-based social networks*. In proc. of CIKM, pages 1582-1586, 2012.

[6] H. Hu, J. Xu, C. Ren, and B. Choi, *Processing private queries over untrusted data cloud through privacy homomorphism*. In proc. of ICDE, pages 601-612, 2011.

[7] S. Mascetti, L. Bertolaja, C. Bettini, *A Practical Location Privacy Attack in Proximity Services*. In proc. of MDM, pages 87 - 96, 2013.

[8] M. F. Mokbel, C. Y. Chow, and W. G. Aref, *The new casper: query processing for location services without compromising privacy*. In proc. of VLDB, pages 763-774, 2006.

[9] S. Mascetti, D. Freni, C. Bettini, et al, *Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies*. The VLDB Journal, 20(4): 541-566, 2011.

[10] K. Mouratidis, M. L. Yiu. *Shortest path computation with no information leakage*. In proc. of VLDB, pages 692-703, 2012

[11] I. Polakis, G. Argyros, T. Petsicos, and et al, *Where's Wally? Precise User Discovery Attacks in Location Proximity Services*. In proc. of CCS, pages 817-828, 2015.

[12] B. Palanisamy, L. Liu, *Attack-Resilient Mix-zones over Road Networks: Architecture and Algorithms*. Trans. Mob. Comput. 14(3): 495-508, 2015

[13] L. Siksnys, J. R. Thomsen, S. Salternis, et al, *Privacy and Flexible Proximity Detection in Mobile Social Networks*. In proc. of MDM, pages 75 - 84, 2010.

[14] W.K. Wong, D.W. Cheung, B. Kao, et al, *Secure knn computation on encrypted databases*. In proc. of SIGMOD, pages 2432-2441, 2009.

[15] G. Yang. *The Complexity of Mining Maximal Frequent Itemsets and Maximal Frequent Patterns*. In proc. of SIGKDD, pages 344-353 2004.

[16] B. Yao, F. Li, X. Xiao. *Secure Nearest Neighbor Revisted*. In proc. of ICDE, pages 733-744, 2013.