Mobile Local Search with Noisy Locations

Rinku Dewri and Ramakrishna Thurimella
Colorado Research Institute for Security and Privacy
Department of Computer Science
2280 S Vine St., University of Denver, CO 80210, USA
{rdewri,ramki}@cs.du.edu
Ph: +1-303-871-4189

Abstract

The deep penetration of mobile devices have led to the emergence of multiple mobile applications that seek to harness the positioning capabilities embedded in such devices. One of the most functional of these applications is local search. Local search is similar to a regular web search, yet is more powerful in a mobile setting since results are ranked both by prominence and locality. Undoubtedly popular, the current design of local search applications fails to cater equally to a privacy-aware user who desires finer controls in her location disclosure. Towards this end, we propose the design for a private local search (PriLS) application that enables a user to first learn the geographic variation in local search results and then use it to determine a noisy location that has little or no affect on the search results. Parametric studies and real-world evaluations show that PriLS can help identify geographic locations that would produce similar search results (compared to when the user's location is used) with no noticeable delays in user experience. They also reveal that large areas typically exist where there is no change in the result of a local search query, and can be exploited to provide spatial privacy guarantees to a user.

Keywords: location privacy, local search, mobile device

1 Introduction

Wide adoption of GPS-enabled mobile devices and advances in cellular telecommunication networks now allow content providers to deliver search results to a mobile user such that they are relevant semantically, as well as, geographically. Local search applications such as Google Places, Yelp, Loopt, Where, AroundMe, and many others, have spawned in the past decade to cater to this user base, and at the same time, tap into the revenue potential of the market. Despite the excellent functionality and growth potential, the collection of user location information raises multiple privacy concerns. Since the legal restrictions remain to be clearly defined, it is difficult to know if the collected information can later be used to encroach on user privacy. Large fraction of users indeed demand protection of their location privacy, yet still consider local search applications to be an invaluable tool in their devices [4]. Unfortunately, existing applications take a permission-based approach to resolve this problem, which either prevents a user from harnessing the power of local search, or requires the user to forgo any privacy expectation.

Table 1: Starbucks locations returned by Google search in Los Angeles, CA, USA.

starbucks near Los Angeles, CA	**starbucks near *Point A***	**starbucks near *Point B***
120 S Los Angeles St	206 N Larchmont Blvd	3461 W 3rd St
217 N Hill St	3461 W 3rd St	3150 Wilshire Blvd
138 S Central Ave	727 Vine St	206 N Larchmont Blvd
800 N Alameda St	3680 Wilshire Blvd	3680 Wilshire Blvd
350 S Grand Ave	3150 Wilshire Blvd	3450 Wilshire Blvd
639 N Broadway	3450 Wilshire Blvd	727 Vine St
333 S Hope St	859 Highland Ave	2134 Sunset Blvd
555 W 5th St	7122 Beverly Blvd	1601 Wilshire Blvd
444 S Flower St	6102 Sunset Blvd	1700 N Vermont Ave
523 W 6th St	5545 Sunset Blvd	859 Highland Ave

Consider the data shown in Table 1. The user Alice in this example is located at *Point A*, somewhere in Los Angeles, CA, USA, and wants to find neighboring Starbucks coffee locations. She uses a local search platform such as Google Places to perform the search on her mobile device. However, Alice is unwilling to reveal her true coordinates. Hence, she denies permission to access her GPS coordinates, and instead performs a generic search for "starbucks in Los Angeles, CA." The first column lists the top ten Starbucks locations retrieved by Google based on their prominence and distance from the city center. The second column shows the top ten locations that would have been retrieved if Alice agreed to share her location. Note that no result is common in the two lists. Therefore, all results returned in this search are inaccurate. In addition, from the standpoint of service quality, Alice is unaware of the level of degradation in the result. The third column in the table lists the results of a similar search, but performed with respect to *Point B*. This list has a 70% match with the list in the second column. Interestingly, *Point B* is a public storage site (256 N Westmoreland Ave, Los Angeles, CA 90004) that is 2 km away from *Point A* (4650 Rosewood Avenue, Los Angeles, CA), a residential address. If the existence of *Point B* (or other such locations) was known to Alice, she could have used it as her coordinates, and retrieved a much useful set of results from the local search. The private local search application we design in this work is directed towards facilitating such informed decision making.

We begin this work with a brief review of existing proposals in location privacy preservation in Section 2. In Section 3, we describe a *privacy-supportive* architecture for local search. Such an architecture involves intermediate metadata exchange between the user device and the provider before retrieving the result set. The metadata helps the client side of the application infer the geographic variability of the results with respect to multiple locations on a broad area. Combining this information with user-driven privacy settings, the client then determines a suitable location for the local search. The primary contribution in this work is the fast generation of this metadata at the server side, and its subsequent utilization for balancing privacy and accuracy. Towards this end, we develop a branch and bound algorithm in Section 4 that operates on an augmented kd-tree data structure, and quickly generates the top query matches for multiple query points. The generated data is then compactly packaged and sent to the client. Section 5 describes the decoding of the packaged data in the resource limited environment of a mobile device, and the subsequent selection of a location that generates similar results as that generated for the user's location. We discuss how privacy is evaluated for this location selection process. Section 6 discusses the performance results from an empirical evaluation on real-world data.

2 Related Work

2.1 Anonymity sets

Location privacy has earlier been achieved through the use of obfuscation and dummy queries. A user can hide her actual query in a set of dummy queries and achieve location privacy [16]. Gruteser and Grunwald [13] proposed the use of spatial and temporal cloaking to obfuscate user locations. The cloaking is performed at a trusted third party site. Individual preferences in terms of temporal and spatial tolerances can also be incorporated during such cloaking [10]. Enforcing properties such as k-anonymity ensures that users will not be uniquely located inside a region in a given period of time. Multiple other suggestions are available on how the cloaking region should be formed. Bamba et al. enforced a location l-diversity requirement where the number of still-object counts must also be above a user-specified threshold [2]; Liu et al. proposed that a minimum level of entropy should also be maintained in the queries originating from the cloaking region [17]; Mokbel et al. presented how different query formats can be supported using cloaking regions [19]; Dewri et al. proposed enforcing query m-invariance in cloaking regions [7]; Shin et al. anonymized the profile of the user using k-anonymity [24]; Riboni et al. proposed smoothing out differences in the distribution of query parameters [5]. Ghinita et al. proposed a decentralized architecture where mobile nodes utilize a distributed protocol to self-organize into a fault-tolerant overlay network, from which a k-anonymous cloaking set of users can be determined [11]. Kalnis et al. proposed that all obfuscation methods should satisfy the reciprocity property [14] in order to prevent inversion attacks where knowledge of the underlying anonymizing algorithm can be used to identify the actual user [12].

2.2 Beyond anonymity sets

Moving beyond anonymity sets, Khoshgozaran et al. proposed a protocol where K-nearest neighbor queries are reduced to a set of private block retrieval operations on a database [15, 23]. These retrievals can be performed using a tamper-resistant processor located at the server so that the content provider is oblivious of the retrieved blocks. Computational inefficiency or the dependence on additional hardware makes this approach currently unsuitable for mainstream adoption.

Most anonymity set based proposals suffer from the requirement to specify privacy parameters that are not intuitive or difficult to gauge. Xu and Cai explored this problem by treating privacy as a feeling-based property and proposed using the popularity of a public region as the privacy level [32]. Soriano et al. showed that the privacy assurances of this model do not hold when the adversary possesses footprint knowledge on the spatial regions over time [29]. Niu et al. recently revisited the use of dummy queries with the objective of addressing side information that the adversary may have on query probabilities from different locations [20]. In a subsequent work, the authors demonstrated that caching of query results can help improve the privacy in dummy query models [21]. These works are driven by an entropy-based privacy metric. Much like the result reuse approach in caching, Shokri et al. proposed a collaborative model where users can retrieve search results from their mobile peers, whenever possible, thereby not requiring location disclosures to the service provider [26].

Shokri et al. argued that location privacy should be quantified based on the expected estimation error of an adversary [25]. They provided a method to arrive at different types of inferences regarding a user's location based on a known mobility profile of the user. Using methods of likelihood estimations, the authors showed

that above measures such as the anonymity set size or entropy do not correctly quantify the privacy enforced by the method [28]. Moreover, all these approaches treat privacy as an immutable property that must be strictly enforced. In reality, most of us like to weigh location privacy with the prospective gains in service before skewing our preferences for one.

2.3 Differential privacy

Dewri introduced the idea of merging a well-known form of privacy in databases, namely differential privacy, and k-anonymity [6]. Under this model, an anonymity set of size k is first formed and then an obfuscated location is generated such that the probabilities of reporting this location from any of the k locations are close to each other. Andrés et al. improved this approach by proposing a new model called geo-indistinguishability, where the dependence on the anonymity set is removed, and a privacy radius is introduced as a parameter [1]. This integration of location privacy and differential privacy remains the state-of-the-art in privacy models for location privacy protection. The primary drawback of these models is that the choice of the obfuscated location is driven only by privacy requirements, and no attempt is made to accommodate its impact on the query results.

2.4 Privacy-accuracy trade-off

Examination of the privacy/accuracy trade-off in location-based applications is rare. Shokri et al. explored an optimal location obfuscation method that can hinder privacy attacks and provide the best service quality, essentially targeting an equilibrium solution in a Stackelberg Bayesian game [27]. They compute quality loss as the average dissimilarity in service quality between the user's true location and a pseudo-location. Privacy is computed as the expected error of the adversary in an inference attack. Along similar lines, Bordenabe et al. provided a mechanism to minimize the service quality loss for a given degree of geo-indistinguishability [3]. Similar to most of the earlier works, both of these works assume that service quality in an application is directly proportional to the distance between the pseudo-location and the true location; however, this assumption hardly holds in a local search application (and others as well) where the quality of search results depends on multiple factors other than distance.

To the best of our knowledge, our prior work is the only known attempt to consider arbitrary ranking functions for local search results, instead of the commonly assumed K-nearest neighbors [8]. We proposed the use of multi-dimensional scaling to do a lossy compression of the result set similarity information into a RGB image. A client then uses the image to decide an obfuscated location with a tolerable loss in service quality. This work presents an alternative method to communicate the similarity information in a lossless format, and also assesses the inferencing possibilities surrounding the choice of the obfuscated location. We acknowledge that a separate series of work also exists for privacy protection in location trajectories. We do not discuss them here due to their limited relevance to this work.

3 Private Local Search (PriLS)

A typical local search process is initiated by the user using an application interface. Some applications may also require the user to authenticate herself. The search query includes the keyword(s) specifying the interests of the user, and is accompanied by a location tag. The location tag helps define a broad geographic area wherein

objects relevant to the search are identified. The location tag could be generic area names, postal codes, street addresses or a precise latitude/longitude pair. In the absence of such tags, this search area may be inferred using other techniques; for example, applications that include a map (e.g. Google Places) can utilize the visible map area as the search region. A ranking algorithm is next applied on the identified objects, the results of which are communicated to the application client on the user device. We refer to the K highest ranked objects as the top-K result set. Content providers typically limit the number of results in the initial communication to the top-10 or top-20 result set. Subsequent results are returned upon request, although the maximum number of ranked results that can be obtained is also limited (60 in Google Places search). Ranking methods may differ from one provider to another, and remain well-guarded business secrets.

3.1 Ranking results

A local search is driven by multiple factors while ranking the objects. Consider the methodology adopted in Google Places [22]. The first considered factor is relevance, which involves the semantic matching of search keywords and object descriptions. The second considered factor is prominence, which is an evaluation of the relative importance of the relevant objects irrespective of their location. Calculation of prominence scores for an object include other sub-factors such as reference counts, highest score of objects that refer to this object, and number of user reviews, among others. The third factor is the distance from a reference point. If the search query explicitly specifies a precise location, then those coordinates can be used as the reference point; otherwise, a geographic center of the broad area, or coordinates of public places may be chosen as the reference point. Numerical scores corresponding to prominence and distance evaluations are normalized and combined into a single score for the object. The exact method of this aggregation is not known, although the available documentation hints at a linear weighted combination. Extensive surveys have been performed to gather public opinions on what (sub-)factors receive the highest weight [18]. The top-K results from the ranking may be displayed locally to the user based on the overall scores, or in order of increasing distance. Note that although the option to view the top-K result set according to distance is locally available to the user, the ranking algorithm dominates which objects get selected in this set. The inclusion of all these factors, as well as the degree to which one factor is given importance over others, makes the top search results different from one application to another. Every local search application seeks to provide recommendations that are useful in the context of the search.

3.2 PriLS process

The precise location of a user can be automatically obtained by the application by communicating with an on-board GPS unit. However, if the user does not grant the permission to do so, then a manual entry of the location tag must be performed. However, as seen in Section 1, the use of a generic tag often defeats the purpose of local search. Few content providers can also estimate user locations using cell-towers and wifi access point databases. The legal standards surrounding this approach is currently being litigated in a number of cases [9].

The private local search (PriLS) application we seek to develop operates within a privacy-supportive architecture. Unlike in a typical location-based services model where the content provider does not aid the user while making privacy decisions, privacy-supportive providers design their applications such that users can make informed decisions after weighing the impact of location inaccuracies on result set accuracy. This is facilitated by exchanging metadata content between the server and the client prior to actual processing of the request.

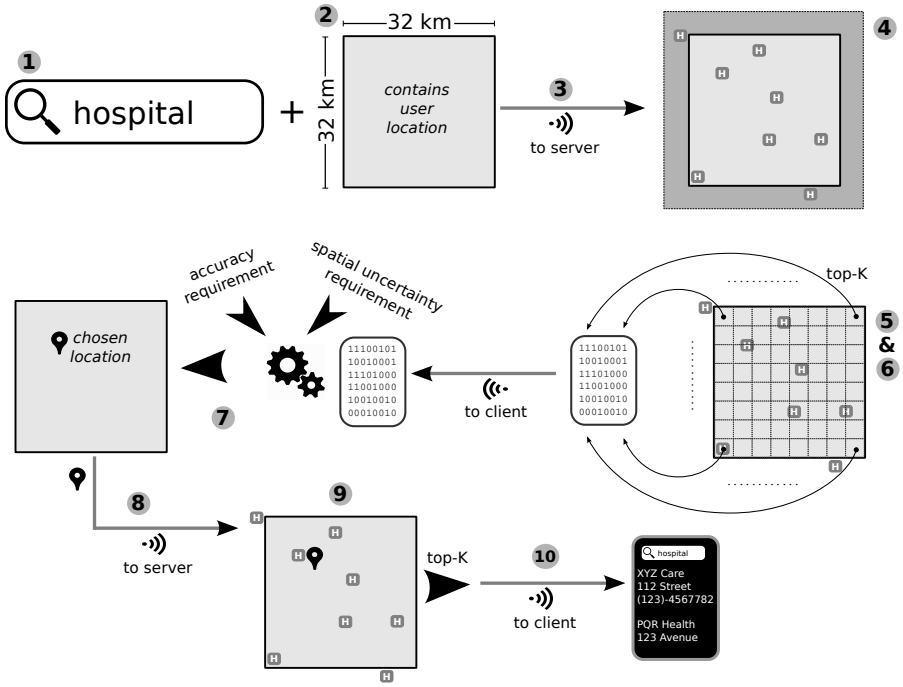

Figure 1: Local search process in PriLS.

The exact content of this metadata may vary from domain to domain, albeit the objective is to provide a quick overview of the location versus service interactions.

Local search in PriLS involves a sequence of client-server communications, with the objective of determining a private location capable of retrieving an acceptably accurate result set. The following provides a description of this sequence (Fig. 1). Client and server processing are indicated by 'C' and 'S' respectively. Bold face headings signify steps that impose additional overhead due to the privacy-supportive architecture.

[1-C] *Accept search keywords through user interface.*

[2-C] *Determine a broad area based on user's location.* PriLS performs this step by randomly generating a rectangular region of significant area (say 1000 km^2) such that the user's location (retrieved from the on-board GPS unit) is contained within it. We use such a large area to prevent the inference of any reasonably precise location for the user during this step.

[3-C] *Send search query (keywords and broad area) to the server.*

[4-S] *Find relevant objects in broad area.* The broad area used to find relevant objects is expanded by a fixed distance in each direction. This is done to ensure that objects close to locations on the edges of the broad area, yet outside, get included in the set of relevant objects.

[5-S] *Determine top-K objects with respect to multiple locations in the broad area.* PriLS overlays the broad area with a $r \times c$ grid, and uses the center of each cell in the grid as a candidate location.

[6-S] *Encode and send the top-K result sets information to the client.* The encoding is performed to succinctly represent which objects get added and removed from the top-K result set as one moves from one location

to the next in the grid. We do not communicate any feature data (e.g. object names, phone numbers, addresses, etc.) in this phase. The encoded metadata is also compressed prior to communication.

[7-C] *Decompress and decode metadata, and pick a random (noisy) location that best meets user set privacy and accuracy requirements.* Given the limited resources available in a mobile device, special attention is given in this step to memory and processing requirements. Keeping the processing time low is also crucial for fast user experience. The accuracy requirement is described in Section 5.

[8-C] *Send coordinates of chosen location to the server.*

[9-S] *Determine the top-K result set with respect to chosen location.*

[10-S] *Send result set (including feature data) to client.*

We emphasize that the data communicated to the client in step 6 contains only POI identifiers, and no other details. As such, it does not contain any information directly useful to the user. It is in steps 8, 9 and 10 that useful information (POI details) is retrieved by making a regular query, albeit with a noisy location. Steps 8, 9 and 10 can be rephrased to say that POI details are retrieved using the identifiers corresponding to the top-K result set for a noisy location. In either case, the noisy location has to decided in step 7. Section 4 focusses on the efficient computation of steps 5 and 6, and Section 5 elaborates on step 7. The remainder of the steps are either trivial, or there exists efficient implementations that are already in use.

3.3 Threat model

The PriLS process assumes a semi-honest content provider (or adversary). Hence, the provider follows the defined protocol, yet may try to infer the location from the data that is exchanged during the protocol. We assume that this inference process is performed at a later time i.e. no real-time verification of a conclusion derived from the inference is performed. For example, the inference may indicate that the user was present at certain coordinates with high probability, but verifying if the user was truly present at those coordinates is not possible. We refer to this as the *offline localization* threat, as opposed to the *real-time tracking* threat. Further, the scenario we consider applies to sporadic queries, i.e. the time difference between two successive queries is significant. Note that the client side of PriLS requires access to the true location of the user (needed in steps 2 and 7 of the process). However, no exchange of these raw coordinates is required with the server. It is also possible to transfer all computations involving raw coordinates from PriLS to the trusted computing base of the system.

4 Server Processing

The PriLS server, upon receipt of a search query, first expands the broad area and then identifies the relevant objects within this expanded area. Spatial data structures such as R-trees and their variants are effective in quickly identifying objects that fall within a bounding rectangle. All local search applications must perform this object search. We begin our discussion from the point where relevant objects matching the search keywords have been identified. Let N be the total number of such objects.

A {1,2,3}	B {1,2,3}	C {1,2,5}
D {1,2,3}	E {1,4,5}	F {1,4,5}
G {1,2,7}	H {1,2,7}	I {1,2,7}

cell	ETP	TDL
A	0	1,2,3
B	01	1,2,3
C	010	1,2,3,5,3
D	0101	1,2,3,5,3
E	01010	1,2,3,5,3,4,5,1
F	010101	1,2,3,5,3,4,5,1
G	0101010	1,2,3,5,3,4,5,1,7,3
H	01010101	1,2,3,5,3,4,5,1,7,3
I	010101011	1,2,3,5,3,4,5,1,7,3

Figure 2: ETP and TDL example for a hypothetical 3×3 grid.

4.1 Encoded data format

PriLS computes the top-K result set corresponding to multiple locations in the broad area. These locations are chosen as the centers of cells belonging to a $r \times c$ grid overlaid on the broad area. We refer to these locations as *evaluation points.* Cells are indexed using a zero-based numbering: $(0,0)$ to $(c-1, r-1)$. We assume that the top-K result set is static for all points inside a cell.

Assuming that the objects are named using integers from 0 to $N-1$, the top-K result set corresponding to cell (x,y) is denoted as $topK(x,y) = \{p_1, p_2, ..., p_K\}_{x,y}$ where each $p_i \in \{0,...,N-1\}$. Note that this representation of a top-K set is sufficient to infer if two sets are referring to the same objects, and to compute the number of common objects. PriLS uses two data structures to encode these sets corresponding to the $r \times c$ cells.

1) *ETP (equal-to-previous) bit vector.* This vector is of size $r \times c$ bits, where the bit at index $x + yr$, $x \in \{0,...,c-1\}$ and $y \in \{0,...,r-1\}$, denotes if the top-K result set corresponding to cell (x,y) is same as that of the previous cell. The previous cell for a cell in the first column is the one at the same column of the preceding row; otherwise, it is the one in the preceding column of the same row. Cell $(0,0)$ always has its bit set to 0. For all other cells, the bit is set as:

$$ETP[x+yr] = \begin{cases} 0 & ,topK(x,y) \neq topK(prev(x,y)) \\ 1 & ,topK(x,y) = topK(prev(x,y)) \end{cases},$$

where $prev(x,y) = (x, y-1)$ if $x = 0$; otherwise $(x-1, y)$.

2) *TDL (top-K delta log) vector.* This vector contains integers in $\{0,...,N-1\}$, from which the top-K result set of all cells can be obtained. The first K integers in the vector are the same as in $topK(0,0)$. Subsequent integers are added as follows. Consider a cell (x_{cur}, y_{cur}) other than $(0,0)$ such that $ETP[x_{cur} + y_{cur}r] = 0$. Let $(x_{pre}, y_{pre}) = prev(x_{cur}, y_{cur})$. Corresponding to (x_{cur}, y_{cur}), we append two sets of integers to TDL: first, the integers that are in the top-K set of (x_{cur}, y_{cur}) but not in that of (x_{pre}, y_{pre}), commonly denoted as $topK(x_{cur}, y_{cur}) - topK(x_{pre}, y_{pre})$, and then, the integers that are in the top-K set of (x_{pre}, y_{pre}) but not in that of (x_{cur}, y_{cur}), i.e. $topK(x_{pre}, y_{pre}) - topK(x_{cur}, y_{cur})$. As an exception to this method, in case the total number of integers to be appended in these two operations is more or equal to K, then $topK\ (x_{cur}, y_{cur})$ is instead appended, with new objects $(topK\ (x_{cur}, y_{cur}) - topK(x_{pre}, y_{pre}))$ appended first. In other words, TDL is a log of what new objects are added and old objects are removed as the top-K result set undergoes a change. The exception handles situations where the top-K set is cheaper to store than the add/remove information.

Example: Consider a 3×3 grid with the hypothetical top-3 sets in Fig. 2. The construction of the TDL and ETP vectors progress as show in the right table. We append the bit 1 to ETP when processing cell D since the previous of cell D is considered to be cell A, and they have the same top-3 set. For cell E, objects 4 and 5 replace objects 2 and 3 in the previous cell's result. However, appending 4, 5, 1 (the top-K of cell E) is cheaper than appending 4, 5, 2 and 3.

The choice of this particular encoding format is motivated by two observations–(i) local search results tend to stay similar for locations in close proximity, and (ii) changes in the result set are incremental. Therefore, result sets can be concisely represented by only tracking their equality with previous sets, and the difference, in case the sets change.

4.2 Augmented kd-tree

We use a kd-tree data structure to store the N objects prior to top-K querying. Given a set of N locations (object positions) of the form (x,y), the location having the median x-value is used as the root node. All remaining locations with x-values less than that of the root is found on a left subtree, and the remainder on a right subtree. The process is repeated recursively on the two subsets; however, the medians are obtained by alternating between the x- and y-dimensions, generically called the splitting dimension. We implement a construction algorithm that pre-sorts all objects and operates in $O(KN\log N)$ time and $O(N)$ storage [31]. During the construction, we augment each node of the tree with object prominence bounds. Each object is assumed to have an associated prominence score in $[0,1]$. Every node of the tree stores the maximum prominence value p_{max} in the subtree rooted at that node (including the node).

4.3 Top-K search

Given a reference point (x_{ref}, y_{ref}), the top-K result set is found based on a ranking function. We use a weighted combination of the prominence score and the distance (normalized) to the reference point as the rank value of an object o at (x,y) and with prominence score p.

$$rank(o) = \alpha \frac{\mathsf{dist}\left[(x,y),(x_{ref},y_{ref})\right]}{\mathscr{N}} + (1-\alpha)(1-p),$$

where dist is a distance metric, $\mathscr{N}$ is a normalizing factor, and $0 \leq \alpha \leq 1$. Under this function, lower rank values signify closer and prominent objects; hence, they are better choices.

4.3.1 Lower bound on rank value

The lower bound on the rank values in a subtree rooted at node o can be estimated by using the rank formula with the maximum prominence value in the subtree and the minimum possible distance between the reference point and any node in the subtree. The maximum prominence score in the subtree is computed for every node of the kd-tree during construction. The computation of the minimum possible distance is performed as follows. Let $d_x(o)$ and $d_y(o)$ be some already known estimate of the minimum absolute distance (along the x- and y-dimensions respectively) of the reference point to nodes in a subtree rooted at node o (inclusive). These values are zero if o is the root node. Assuming an Euclidean distance metric, an estimate of the minimum possible distance for the subtree rooted at o is then $\sqrt{d_x(o)^2 + d_y(o)^2}$. Let o_l and o_r be the left and right child nodes

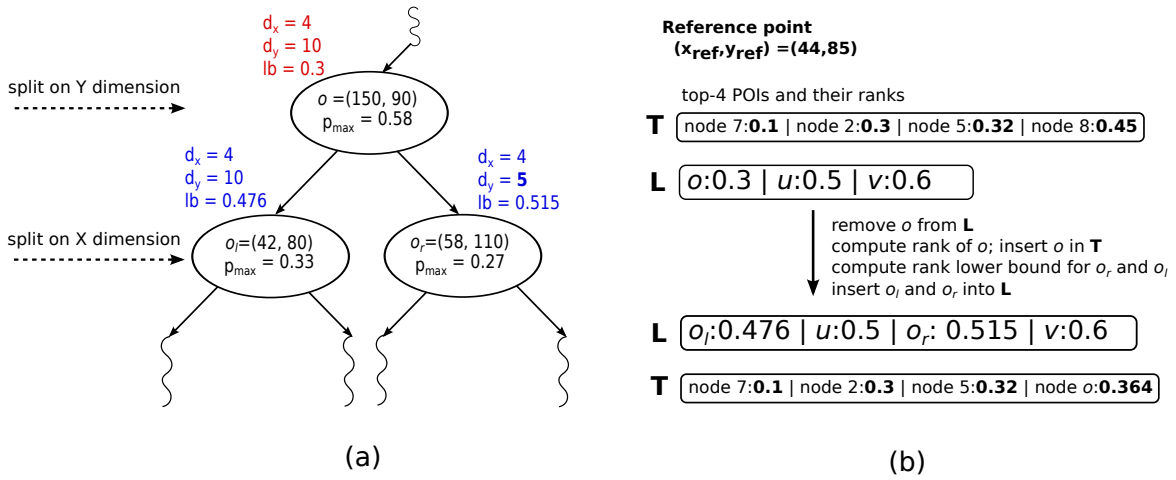

Figure 3: (a) Example kd-tree for lower bound computation and top-4 search; assumed: $\alpha = 0.3$ and $\mathcal{N} = 320\sqrt{2}$. (b) Example contents of data structures used in top-4 search.

respectively. Without loss of generality, let us assume that the x-dimension was used as the splitting dimension at o, and $x_{ref} < x$. Since all nodes in the right subtree of o will have x-dimension values greater or equal to x, we can use $|x - x_{ref}|$ as the minimum absolute distance (in the x-dimension) between the reference point and any node in the subtree rooted at o_r, or $d_x(o_r) = |x - x_{ref}|$. However, we cannot make such a claim for the left subtree, and hence the parent estimate is maintained, i.e. $d_x(o_l) = d_x(o)$. No new information is available for the y-dimension at this level; hence, $d_y(o_r) = d_y(o_l) = d_y(o)$. The analysis is similar if $x_{ref} \geq x$, with the change that $d_x(o_l) = |x - x_{ref}|$ and the other quantities use the parent estimates. Similarly, when the y-dimension is the splitting dimension, parent estimates in the x-dimension are used, while improved ones in the y-dimension are possible.

Example: Refer to the partial kd-tree in Fig. 3(a). Assume that the reference point is $(x_{ref}, y_{ref}) = (44, 85)$ and distance estimates d_x and d_y have already been computed for node o. Recall that these estimates are zero for the root node. For the purpose of demonstration, say $d_x(o) = 4$ and $d_y(o) = 10$. We will now compute these estimates for the left and right subtrees of o. Observe that $y_{ref} = 85$ is less than the y-coordinate of o ($= 90$). Given that the y-dimension is used for splitting nodes at o, all nodes in the right subtree of o have y-coordinate values greater than 90. Hence, the minimum y-distance we will see for nodes in the right subtree of o is $d_y(o_r) = |85 - 90| = 5$. A similar argument is not possible for the x-distance; however, we can still retain the estimate from the parent of o_r, effectively giving us $d_x(o_r) = d_x(o) = 4$. For the left subtree, we can only retain the values from the parent, i.e. $d_x(o_l) = d_x(o) = 4$, $d_y(o_l) = d_y(o) = 10$. Assuming $\alpha = 0.3$ and $\mathcal{N} = 320\sqrt{2}$, the rank lower bound for the subtree rooted at o_r is then $0.3 \times \frac{\sqrt{4^2+5^2}}{320\sqrt{2}} + 0.7 \times (1 - 0.27) = 0.515$, and that at o_l is $0.3 \times \frac{\sqrt{4^2+10^2}}{320\sqrt{2}} + 0.7 \times (1 - 0.33) = 0.476$.

4.3.2 Best first search

The top-K search is performed using a best first strategy on the augmented kd-tree. The nodes to explore are maintained in a list L, sorted in increasing order of the rank lower bound obtainable in the subtree rooted at the node (inclusive). We denote the lower bound by the suffix *.lb*. References to the current best K nodes are

maintained in another ordered list T (first best to K^{th} best). The exploration starts with only the root node in L. A single step during the search proceeds as follows. The head node o in L is removed from the list. If the list T has less than K node references, or $o.lb$ is less than the rank value of one or more nodes referred to in T, then the node o and/or its children nodes could *potentially* be included in the top-K result. Otherwise, the search is terminated as K objects have already been identified and lower rank values are no longer possible. In the former case, we insert o in T according to its rank if necessary. Next we compute the lower bound for the left and right subtrees of o, and insert the left and right child nodes into L (positioned according to the computed bounds). This finishes one step of the search. The process is repeated until the list L is empty, or the search is terminated.

Example: Consider the L list shown in Fig. 3(b) during a top-4 search for the reference point $(x_{ref}, y_{ref}) = (44, 85)$. We delete the first node in the list and consider the subtree rooted at that node (o at $(150, 90)$ in the example). Using $\alpha = 0.3$ and $\mathscr{N} = 320\sqrt{2}$, and assuming that the prominence value of the POI corresponding to node o is 0.58, the rank of o is then $0.3 \times \frac{\sqrt{(150-44)^2+(90-85)^2}}{320\sqrt{2}} + 0.7 \times (1-0.58) = 0.364$. Therefore, the POI corresponding to o replaces the fourth element in the T list. The figure also shows where o_r and o_l are inserted in L. Note that in the next iteration, we can terminate the search since the first node in the updated L (i.e. o_l) has a lower bound that is larger than the rank of the last node in the list T.

4.4 Pruning evaluation points

Procedure 1 lists the abridged pseudo-code to populate the ETP and TDL vectors for encoding the top-K results (covers steps 5 and 6 of PriLS). A top-$(K+1)$ search is performed and the result is stored in a list T ordered by increasing value of the object ranks. Recall that the top-K search process also utilizes a similar list T. In fact, the same list is used in both modules, i.e. the list T in the top-K search is always initialized to the result of the previous search, if any. The *skip* flag indicates whether the search should be performed for an evaluation point. Lines 6 and 7 store the rank value of the $(K+1)^{th}$ best object and the index of the evaluation point whenever a search is performed. Lines 8 and 9 update the ETP vector and append data to the TDL vector, if necessary, as explained in Section 4.1.

The pruning logic is implemented in Lines 10 through 14. First, using the next evaluation point $(x+yr+1)$ as the reference point, we recompute the rank of the objects in T. The objects are then reordered within T in increasing order of these new rank values. An insertion sort is effective in this step since most objects are already likely to be in their correct positions. Line 12 computes a lower bound for the K-th rank value if the search for the next evaluation point is performed. Let $\delta/\mathscr{N}$ be the smallest normalized distance between two evaluation points. Hence, as we move from one evaluation point to the next, the distance to any object can at best reduce by δ. If the last search was performed for evaluation point lea, then the rank value of an object corresponding to evaluation point $x+yr+1$ can at best decrease by $\alpha(x+yr+1-lea)\frac{\delta}{\mathscr{N}}$. The $(K+1)^{th}$ best object in the last search had rank value τ, which can at best become $\mathscr{R} = \tau - \alpha(x+yr+1-lea)\frac{\delta}{\mathscr{N}}$. Any other object that had rank values higher than τ during the last search will have a new rank value at least as large as $\mathscr{R}$. Therefore, if $\mathscr{R}$ is larger than or equal to the rank value of the current K^{th} best object (in the reordered list T), then no object can replace the first K existing objects in T. The evaluation point $x+yr+1$ can then be skipped. We always perform the search for the cells in the first column. Note that the true user location is not explicitly used in the procedure. However, it does get implicitly used through the top-$(K+1)$ search (Line 5).

We use 2 bytes to represent each integer in the TDL vector, thereby limiting the number of relevant objects N to at most 65536. The ETP and TDL data, along with the object count N, are compressed using the zlib

Procedure 1: PriLS computation of TDL and ETP data

Input : r : number of rows in broad area, c : number of columns, K : number of top matches desired
Output: ETP: equal-to-previous vector, TDL: top-K delta log vector

```
1  skip ← false
2  for y ← 0 to r − 1 do
3      for x ← 0 to c − 1 do
4          if skip = false then
5              T ← top-(K + 1) search result for (x, y)
6              τ ← rank(T[K + 1])
7              lea ← x + yr //last evaluation at
8          compute ETP bit
9          if ETP[x + yr] = 0 then add data to TDL vector
10         if x + yr < rc − 1 then
11             reorder objects in T in increasing order of ranks computed w.r.t. next evaluation point
12             ℛ ← τ − α(x + yr + 1 − lea) δ/𝒩
13             if ℛ ≥ rank(T[K]) and x < c − 1 then skip ← true
14             else skip ← false
15
```

compression library (www.zlib.net) and then sent to the client.

In this work, we have not explored the possibility of sending the coordinates of the N objects to the client, and performing the top-K computations in the client device. Few issues have to be addressed before we can adopt this alternative method. First, it will require that the ranking function (used in determining the top-K results for a cell) is implemented on the client side of the application. If a ranking function is hard coded into the client application, then a modification to it would imply an update to the application. Clients not opting to update the application will therefore continue to use the old function. This can generate result rankings that are not consistent across user devices, and prevent the service provider from enforcing a uniform business model. Second, implementation of a parametric ranking function at the client side would require that parameter values (such as weights and prominence values) are transmitted to the client application. We are unaware of any service that provides access to such proprietary data, thereby limiting the development of third-party client applications that leverage the proposed architecture. More exploration is needed on how ranking-specific data can be released without revealing the business logic of the service provider. Third, since transferring the complete details of the N objects will lead to unnecessary overhead, a client will (at some point) have to finally request specific details on a chosen set of objects. The exact method used here will dictate if the user's location can be compromised by analyzing the chosen set. Finally, the search algorithm may or may not be suitable for execution on a client device.

5 Client Processing

The user device, upon receipt of the compressed ETP and TDL data, first decompresses it in memory. Procedure 2 highlights how the top-K result set of each cell can be obtained from the ETP and TDL data structures. The

Procedure 2: Decode top-K result sets

Input : r : number of rows in broad area, c : number of columns, K : number of top matches desired, ETP: equal-to-previous vector, TDL: top-K delta log vector

Output: as determined in Line 19

```
1  topK ← φ; fc.topK ← φ
2  for y ← 0 to r − 1 do
3      for x ← 0 to c − 1 do
4          adds ← φ; to.delete ← 0
5          if ETP[x + yr] = 0 then
6              repeat
7                  o ← TDL.next()
8                  if o ∉ topK or |adds| ≥ K/2 then
9                      adds ← adds ∪ {o}
10                     to.delete ← to.delete + 1
11                 else
12                     topK ← topK − {o}
13                     to.delete ← to.delete − 1
14             until |adds| = K or to.delete = 0
15             if |adds| = K then topK ← adds
16
17             else topK ← topK ∪ adds
18
19         Note:topK(x,y) is topK; use as necessary
20         if x = 0 then fc.topK ← topK
21
22         if x = c − 1 then topK ← fc.topK
23
```

algorithm assumes the existence of a function $TDL.next()$ that returns the next unread integer in the TDL vector. Since K is a constant, the number of additions and removals to a top-K set will always be equal. The $to.delete$ variable is incremented for every new object addition, and then an equal number of objects are removed from $topK$. If the number of additions becomes greater or equal to $K/2$, then no removal data is present; as a consequence of the exception, all K objects of the result set are instead included. This condition is tested in Line 8. Lines 15 and 17 update $topK$ with the newly added objects, or populate it from scratch if the exception method was used. Line 20 overwrites $topK$ with data from the first column ($fc.topK$) as the ETP bit for the first column is computed based on this set. All logic surrounding the top-K result of a cell can be placed in Line 19. Executing Procedure 2 on the TDL and ETP vectors generated in the example in Section 4.1 will recreate the top-3 result sets.

In order to avoid decoding each and every top-K result set in memory, PriLS uses a two pass process to select a cell as the user's location. The first pass determines the top-K result set corresponding to the user's current location (cell). This is performed by executing Procedure 2, however breaking out of the loops (in Line 19) after the user's cell has been processed. Let (x_u, y_u) be the user's cell; $topK(x_u, y_u)$ is the top-K result set obtained from this pass. In the second pass, the top-K result set corresponding to each cell is matched with $topK(x_u, y_u)$. For each possible value $t = 0, ..., K$, we maintain a count $cm(t)$ of the number of cells that has t

objects same as in $topK(x_u, y_u)$.

Ideally we wish to be able to randomly select a cell from the set of cells corresponding to a given number of matching objects. However, this would require storage of all cell indices along with the matching counts. We avoid this storage by adopting the method of reservoir sampling (a reservoir of size one for each value of t). Under this method, we record the locations of $K+1$ cells, one for each possible value of t, denoted as $rloc(t)$. Let us assume that the number of matches between $topK(x,y)$ and $topK(x_u, y_u)$ is t. As a result, $cm(t)$ is incremented by one. We then set $rloc(t) = (x,y)$ with probability $1/cm(t)$. Therefore, the $rloc$ data gets probabilistically updated as we progress through all the cells.

5.1 Privacy evaluation

Recall that the TDL vectors only encode POI identifiers, and has no details on the POIs. Therefore, a regular query with a location tag has to be made to retrieve the usable results (steps 8, 9 and 10). A simple method to select a location for use in the query would be to randomly pick a cell whose top-K set has at least a specified number of matches with the top-K set of the user's cell. We denote this threshold parameter by $\mathscr{M}$, an integer between 0 (any result set) and K (no accuracy loss). Let us assume that (x^*, y^*) is the cell chosen under this method, and communicated to the server in step 8 of PriLS.

The semi-honest server can ask the following question: *which cells (x,y) in the grid could the user be located such as (x^*, y^*) can potentially be a chosen location?* The answer depends on the value of $\mathscr{M}$. Without knowledge of this parameter, all cells are candidate user locations, thereby assuring the required privacy. For subsequent analysis, we assume that the adversary's background knowledge consists of the value used for $\mathscr{M}$.

5.1.1 Attacking the simple selection method

We assume that the adversary has some prior knowledge on where the user was located at the time of the query. We represent this knowledge by a probability distribution Φ on the cells, where $\Phi(x,y)$ is the user's probability of being at the cell (x,y) according to the adversary's prior knowledge. Once the adversary obtains the false location (x^*, y^*), he can create a posterior distribution Φ' by conditioning on the requirement that $topK(x^*, y^*)$ must have at least $\mathscr{M}$ matches with the top-K of a potential user cell. Let $match((x_1,y_1),(x_2,y_2)) = |topK(x_1,y_1) \cap topK(x_2,y_2)|$. Only cells with at least $\mathscr{M}$ matches are potential user cells. The posterior distribution is then given as

$$\Phi'(x,y) = \begin{cases} \frac{\Phi(x,y)}{\sum_{match((x',y'),(x^*,y^*)) \geq \mathscr{M}} \Phi(x',y')} & , match((x,y),(x^*,y^*)) \geq \mathscr{M} \\ 0 & , otherwise \end{cases}. \quad (1)$$

5.1.2 Ensuring privacy

The posterior distribution will immediately help eliminate a number of cells in the grid, specifically those with $match(\cdot,(x^*,y^*))$ less than $\mathscr{M}$. For others, the posterior probability is a constant factor of the prior probability. Under the offline localization threat, we consider such cells to contribute towards the spatial privacy area provided by the algorithm.

$$Privacy_{\text{offline localization}} = |\{(x,y) | match((x,y),(x^*,y^*)) \geq \mathscr{M}\}| \quad (2)$$

Note that some of these cells may have a higher posterior probability than others; some may also be zero. However, as can be seen from Eq. 1, this skewness is due to the prior distribution Φ and is not an artifact of the privacy algorithm. In fact, within the spatial privacy area, the inferences that the adversary can make using Φ' can simply be made using Φ.

5.2 Location selection

For privacy, the cell (x^*, y^*) is to be chosen such that we can easily compute the minimum number of cells that has at least $\mathscr{M}$ matches with it. Let us consider the scenario where our selection method picks one of $rloc(K), ..., rloc(z)$ as the point (x^*, y^*), for some given z. We wish to compute a lower bound on the spatial privacy level provided by this selection. Our derivation is based on the following observation.

Observation: *Given three top-K sets, namely* $topK\ (x_u, y_u)$, $topK(x_1, y_1)$ *and* $topK(x_2, y_2)$, *such that*

$$\begin{aligned} match((x_u, y_u), (x_1, y_1)) = |topK(x_u, y_u) \cap topK(x_1, y_1)| &= n_1 \\ match((x_u, y_u), (x_2, y_2)) = |topK(x_u, y_u) \cap topK(x_2, y_2)| &= n_2, \end{aligned}$$

at least $\max(0, n_1 + n_2 - K)$ *objects must be common between* $topK(x_1, y_1)$ *and* $topK(x_2, y_2)$, *i.e.* $match((x_1, y_1), (x_2, y_2)) \geq \max(0, n_1 + n_2 - K)$.

Let us consider the subset C of cells such that $(x_u, y_u) \in C$ and, for any two cells (x_1, y_1),$(x_2, y_2) \in C$, $match((x_1, y_1), (x_2, y_2)) \geq \mathscr{M}$. Let z be the minimum number of matches between $topK(x_u, y_u)$ and the top-K result of a cell in C. Based on the above observation, any two cells that has $z\ (\geq K/2)$ matches with $topK(x_u, y_u)$, will have, in the worst case, $(z + z - K)$ matches between their top-K results. Therefore, in the set C, we must have $2z - K \geq \mathscr{M}$ or $z \geq \lceil \frac{1}{2}(K + \mathscr{M}) \rceil$. As a result, if the selection method uses $z = \lceil \frac{1}{2}(K + \mathscr{M}) \rceil$, then at most $(K - z)$ mismatches can occur and the spatial privacy area is bounded as

$$Privacy_{\text{offline localization}} \geq \sum_{t=z}^{K} cm(t). \tag{3}$$

This lower bound on the spatial privacy facilitates privacy-accuracy trade-off feasibility checks. The user provides privacy expectations in terms of the minimum required obfuscation area $\mathscr{S}$ and an accuracy expectation in terms of the number (or percentage) of result matches $\mathscr{M}$ that must exist. Using the supplied $\mathscr{M}$, we calculate the smallest spatial privacy area as per our location selection method and verify that it is at least $\mathscr{S}$; otherwise the user is notified that the provided settings are infeasible and requirements need to be relaxed. In an alternative interface, the user may simply be asked to provide the privacy expectation $\mathscr{S}$. The application can compute z assuming $\mathscr{M} = K$, and verify if the spatial privacy is satisfied. If not, the value of $\mathscr{M}$ is decremented and the process is repeated. This proceeds until the specified spatial privacy requirement is met or $\mathscr{M}$ becomes zero. In such an interface, the user should be notified that a certain number of results $(K - z)$ may be incorrect. The latter interface may also be more suitable since users are likely to always set $\mathscr{M}$ to the highest possible accuracy $(= K)$. In our empirical study, we show what minimum spatial privacy levels are possible when no mismatch is tolerated $(z = K)$ and when one mismatch can be tolerated $(z = K - 1)$.

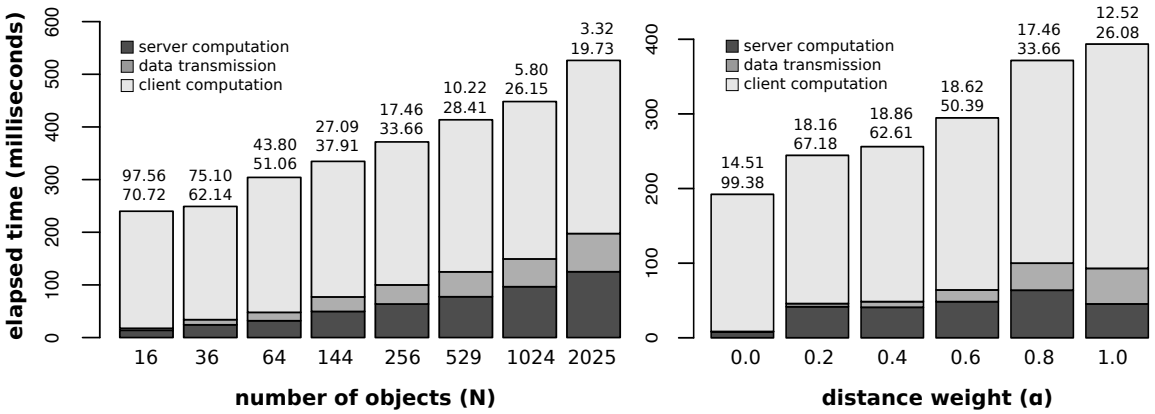

Figure 4: Overhead of computation time at server and client, and data transmission time (over 3G–320KB/s) for varying object counts N and weight α on the distance to objects. The two numeric quantities for each bar represent—top: the average percentage of nodes explored in the augmented kd-tree during a top-10 search; bottom: percentage of skipped evaluation points on the grid.

6 Empirical Evaluation

For performance evaluation, we assume a 160×160 grid (25600 evaluation points) over a 32×32 km^2 broad area ($ONE.CELL.AREA = 200 \times 200$m^2). Local business listings in Los Angeles, CA, USA are obtained from the SimpleGeo Places database, and used as object locations. K is set at 10, and the distance weight α is set to 0.8. The distance normalizer $\mathcal{N}$ is set to the length of the diagonal of the broad area. Object prominence scores are assigned from $\{0.95, 0.90, ..., 0.2, 0.25\}$ using a Zipf distribution with exponent 0.8. Lower scores are more frequent under this distribution. Experiments are performed on a 2.8 GHz quad-core Intel Xeon system running Mac OS X 10.6 with 8GB memory. Server side algorithms are implemented in C and executed using four threads (evaluation points are equally divided). Client side of PriLS is executed on an Android emulator running a virtual device with a ARM Cortex-8 processor ($\sim$800MHz) and 512MB memory. We assume that a 3G connection with 320 KB/s speed [30] is used to transmit the compressed ETP and TDL data.

6.1 Parametric study

Fig. 4 shows a breakdown of the additional wall-clock time that would be incurred by PriLS, compared to other local search applications, as a result of varying N and α. Computation time at the server includes the steps of constructing the kd-tree, generating the ETP and TDL data, and then compressing them. The client computation time includes data decompression and user location selection using the two pass method. We assume a worst-case scenario where the user is located in the last cell, thereby forcing both passes to traverse the entire ETP and TDL data.

6.1.1 Number of objects (N)

Majority of the overhead is incurred at the client device, although the total time is within half a second. The generation and transmission of the ETP and TDL data incurs nominal overhead. This can be attributed to the

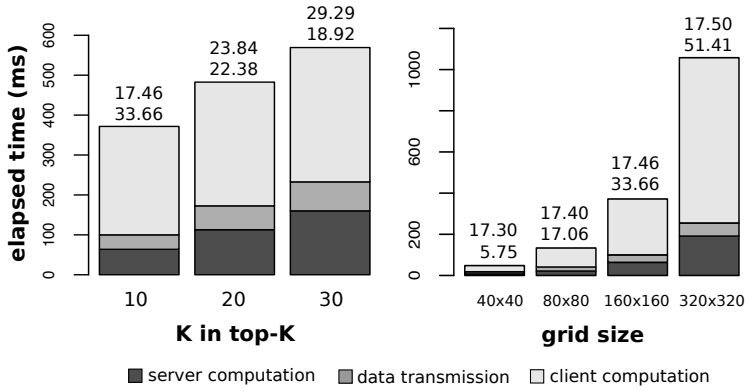

Figure 5: Overhead of computation time at server and client, and data transmission time (over 3G–320KB/s) for varying K and grid size. The two numeric quantities for each bar represent—top: the average percentage of nodes explored in the augmented kd-tree during a top-10 search; bottom: percentage of skipped evaluation points on the grid.

pruning heuristics in the server-side algorithms. The two numeric quantities corresponding to each parameter value signify the average percentage of nodes (out of N) explored when a top-K search is performed and the percentage of evaluation points that is skipped. The tree exploration is close to exhaustive for small number of objects, but reduces significantly for moderate to higher values of N. However, the timing is not significantly affected since the top-K result set seldom changes when the number of objects is fewer. Note that the number of evaluations skipped is much higher for lower values of N. Higher number of objects tend to result in a longer TDL vector. Nonetheless, we observe a data compression ratio of at least $1:2$ in all cases.

6.1.2 Weight on distance (α)

Not surprisingly, when no weight is given to distance ($\alpha = 0$), the same result set applies to all cells and most evaluation points are skipped. However, there does not seem to be any specific trend in server computation time as α is increased. The result set undergoes more changes as distance receives higher weight, resulting in longer TDL length, data transmission time and client processing time. Irrespective of these trends, the total overhead of the process is within half a second.

6.1.3 Result size (K)

A higher number of objects in the result set allows us to compute result similarity over a larger set. The top-K searches require slightly more time (Fig. 5) since more number of objects are being sought (more empty slots in T). Ideally, result similarity assessment using high K values is not necessary if the focus is on retrieving most of the top few objects. We use the value of 10 since most applications initially display the top-10 results. Too high values such as $K = 50$ are not typical since user's are likely to rephrase search keywords if useful results are not available in the top ten or twenty results.

Table 2: Worst case performance statistics of PriLS for search terms with varying densities. Evaluation performed using $K = 10$ and a 160x160 grid (25600 evaluations) over a 32x32 km^2 area centered at Los Angeles, CA, USA (34.053691^o N, 118.243126^{o}W).

search term	N	% nodes explored	% evaluations skipped	server computation time (ms)	data transmission time (ms)	client computation time (ms)	total overhead time (ms)
bowling	12	99.37	80.42	8.97	1.69	222.58	233.24
bus station	32	67.95	41.17	26.75	8.44	227.63	262.82
farmers market	50	58.06	33.43	44.08	9.72	240.49	294.29
police	84	48.05	33.64	52.36	12.16	225.03	289.55
starbucks coffee	92	35.63	26.16	48.56	16.19	246.02	310.77
grocery	95	35.42	33.15	45.86	14.84	229.41	290.11
restaurant italian	124	28.61	29.96	46.93	15.28	242.46	304.67
liquor store	125	29.20	32.37	52.06	19.00	234.14	305.20
bookstore	126	28.98	29.18	49.86	17.12	239.26	306.24
library	141	27.86	30.65	54.19	18.12	241.39	313.70
night club	149	23.74	34.75	47.67	19.66	238.96	306.29
clothing store	169	28.63	25.96	71.59	18.38	241.56	331.53
car rental	196	23.87	24.63	64.82	18.38	233.09	316.29
department store	208	20.95	26.08	61.68	22.41	247.81	331.90
parking	281	18.83	18.93	84.53	18.28	241.39	344.20
atm	297	15.66	27.76	66.80	25.81	253.86	346.47
gas station	347	13.52	27.74	72.20	28.66	254.14	355.00
pharmacy	369	13.48	23.84	75.65	28.72	257.84	362.21
hospital	476	15.49	16.89	118.81	27.94	255.77	402.52
café	608	9.79	20.68	92.08	30.34	260.72	383.14
bakery	834	7.15	22.36	98.27	37.31	272.78	408.36

6.1.4 Grid size

Smaller number of cells imply larger cell sizes and less evaluation points. Hence, the total computation time is relatively much less (Fig. 5). However, when larger cell sizes are used, the assumption that all points within a cell generate the same top-K result set starts to fail. On the other hand, using an overly fine grid results in more evaluation points, with most points being skipped during computation (more than 50% for a 320×320 grid). The total computation time can also be comparatively higher in such cases.

6.2 Performance evaluation

Table 2 shows the performance statistics for different search keywords on the Los Angeles data. The search terms produce object counts varying from 12 for low density businesses to 834 for high density businesses. The observed statistics are very much similar to those in the parametric study – low object counts result in more exploration of the tree, but comparatively more pruning of evaluation points; higher object counts result in more data being transmitted. Computation time at the server includes the steps of constructing the kd-tree, generating the ETP and TDL data, and then compressing them. The client computation time includes data decompression and user location selection using the two pass method. We assume a worst-case scenario where the user is located in the last cell, thereby forcing both passes to traverse the entire ETP and TDL data. The total overhead

Table 3: Spatial privacy area in PriLS for different search terms in Los Angeles. Each cell shows the value for no mismatch and at most one mismatch, separated by '/' . Legends—**OCA**: area of one cell (0.04 km^2), ***CB***: City Block (0.05 km^2), ***GM***: Giant Mall (0.3 km^2), ***SE***: Section (2.5 km^2), ***PK***: Park (7 km^2), ***NH***: Neighborhood (20 km^2), ***ST***: Settlement (60 km^2), ***TW***: Township (120 km^2), ***CT***: City (500 km^2).

search term	spatial privacy area				
	0^{th} percentile	1^{th} percentile	10^{th} percentile	25^{th} percentile	50^{th} percentile
atm	OCA/GM	CB/SE	GM/PK	GM/PK	SE/NH
bakery	OCA/GM	OCA/GM	CB/SE	GM/PK	GM/NH
bookstore	OCA/SE	CB/PK	GM/PK	GM/NH	PK/ST
bowling	PK/CT	PK/CT	ST/CT	TW/CT	TW/CT
bus station	OCA/PK	GM/NH	SE/ST	PK/TW	NH/TW
café	OCA/GM	CB/SE	GM/PK	GM/NH	SE/NH
car rental	OCA/GM	CB/SE	GM/PK	GM/NH	SE/ST
clothing store	OCA/SE	CB/PK	GM/NH	GM/NH	SE/ST
department store	OCA/SE	CB/PK	GM/PK	GM/NH	SE/NH
electronics store	GM/TW	PK/TW	NH/TW	TW/CT	TW/CT
farmers market	OCA/PK	GM/PK	SE/NH	PK/ST	NH/TW
gas station	OCA/GM	CB/SE	GM/PK	GM/PK	SE/NH
grocery	OCA/SE	CB/PK	GM/NH	SE/NH	PK/ST
hardware store	OCA/SE	CB/PK	GM/NH	SE/NH	PK/ST
hospital	OCA/GM	CB/SE	GM/PK	GM/PK	SE/NH
library	OCA/SE	CB/PK	GM/NH	GM/NH	SE/ST
liquor store	OCA/SE	CB/SE	GM/PK	GM/NH	SE/ST
night club	OCA/GM	CB/SE	GM/PK	GM/NH	SE/ST
parking	OCA/GM	CB/SE	GM/NH	SE/NH	PK/ST
pharmacy	OCA/GM	CB/SE	GM/PK	GM/PK	SE/NH
police	OCA/PK	CB/PK	GM/NH	SE/ST	PK/ST
restaurant italian	OCA/GM	CB/PK	GM/NH	SE/NH	PK/ST
starbucks coffee	OCA/SE	CB/PK	GM/NH	SE/NH	PK/ST

time added by PriLS does not contribute any visible difference to user experience. Based on the data in the table, an average of 60 milliseconds is spent by the server to generate the ETP and TDL structures for a query. Hence, such a processing server would be able to serve about 16 queries per second. As a rough estimate, 5 such servers will be necessary to address a workload of 3 million queries over a period of 12 hours. The number will be slightly higher during peak request periods. While not an inordinate resource requirement, any improvement in the data structures and the search algorithms will add to the scalability of PriLS.

6.3 Privacy evaluation

Table 3 lists the spatial privacy area generated in PriLS for the Los Angeles area. The area is obtained by multiplying the privacy level given by Eq. 2 with the area of one cell ($0.04km^2$). The first value in a cell corresponds to the case of zero mismatch ($z = 10$), and the second value corresponds to the case of at most one mismatch ($z = 9$).

The resulting spatial privacy level for a user depends on where the user is located, the cell (x_u, y_u), when making the query. We summarize this statistic using percentiles. The p^{th} percentile value v indicates that, when cells are sorted in increasing size of their induced spatial privacy area for the user, each of the first $p\%$ of the

cells generate an area smaller or equal to v. In other words, the remaining $(100-p)\%$ of the cells will have a spatial privacy area at least as large as v. Consider the values for "starbucks coffee" in the 1^{th} percentile column: CB/PK. The first value (CB) indicates that queries originating from 99% of the cells can retrieve the correct top-10 Starbucks and have a spatial privacy area of a "City Block" (0.05 km^2). However, when at most one result mismatch is allowed, the privacy area increases to that of a large "Park." By looking at the other values in the row, we can conclude that for most places (say the 10^{th} percentile) in Los Angeles from where a user can search for Starbucks coffee shops, PriLS can help retrieve the exact top-10 shops by hiding the user in an area the size of four shopping malls (GM), and increase this obfuscation to a significantly large area (NH) if at most one incorrect result can be tolerated. Based on the results across the variety of search terms, it is interesting to see that for a user (in Los Angeles) who can tolerate one mismatch in the result set, there is no reason for a local search application to collect precise location coordinates; in fact, the precision required can be much coarse in certain areas. We want to emphasize that, even after setting a tolerance of at most one mismatch, the retrieved results may still have no mismatch.

Note that even for the user who has no privacy concerns ($\mathscr{S} = OCA$), PriLS still provides a default level of privacy as is implicit due to the distribution of the searched objects. For example, search terms like "bowling" and "electronics store" still generate spatial privacy areas larger than one cell. In general, PriLS improves the privacy level of the user beyond specified expectations, whenever possible. For instance, a user may have set the spatial threshold to the size of a "Giant Mall". When searching for "starbucks coffee," her privacy obfuscation will be (i) at the level of a "Park" in 50% of the cells (no mismatch), (ii) at the level of a "Section" in 25% of the cells (no mismatch), (iii) at the level of a "Giant Mall" in 15% of the cells (no mismatch), (iv) at the level of a "Park" in 9% of the cells (one mismatch), and (v) at the level of a "Section" in 1% of the cells (one mismatch).

7 Conclusions

In this paper, we proposed the design of a mobile local search application (PriLS) that performs a privacy-utility analysis prior to issuing a location based query. By exchanging succinct data structures between the server and the client, PriLS makes a best effort to hide the user location within a specified spatial uncertainty area, with minimal impact on the result accuracy. We implemented multiple heuristics to make the process efficient and suitable for real-world adoption. The empirical evaluation indicates that the proposed algorithms are able to prune a large section of the search tree during a top-K search, as well as, reduce the number of searches to be performed. The overhead associated with the execution of the PriLS steps has been found to be less than 500 milliseconds. In addition, we demonstrate the possibility that accurate local search may be feasible with much coarser knowledge of the user location. The distance metric we use here is Euclidean, although adapting the computations to use network distances will not be difficult.

Other forms of adversary models can also be considered. One can consider a malicious server who manipulates the ETP and TDL data structures in an attempt to learn the true location of the user. It remains to be analyzed if an intelligent modification is indeed possible. In addition, the server must also keep the user motivated to use the service. This in itself is much more difficult once the user observes discrepancies in the final query answers and the physical realities. We have also not considered the case of a user making multiple queries in a relatively small amount of time. Each query will impose some constraint on where the user could be located, thereby making it possible to correlate such short burst queries for a finer approximation of the user's location.

The biggest hindrance to immediate real world deployment of our approach is that local search services are not designed to be privacy-supportive, and they do not directly provide the similarity information necessary in our method. Nonetheless, at the expense of a slight latency, a middleware service can be established that will issue top-K queries to a local search provider and generate the necessary similarity information. We expect that, in the long run, such information will be directly obtainable from the service provider.

References

[1] M. E. Andrés, N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi. Geo-indistinguishability: Differential Privacy for Location-Based Systems. In *Proceedings of the 20th ACM Conference on Computer and Communications Security*, pages 901–914, 2013.

[2] B. Bamba, L. Liu, P. Pesti, and T. Wang. Supporting Anonymous Location Queries in Mobile Environments with Privacy Grid. In *Proceedings of the 17th International World Wide Web Conference*, pages 237–246, 2008.

[3] N. E. Bordenabe, K. Chatzikokolakis, and C. Palamidessi. Optimal Geo-Indistinguishable Mechanisms for Location Privacy. In *Proceedings of the 21th ACM Conference on Computer and Communications Security*, pages 251–262, 2014.

[4] J. L. Boyles, A. Smith, and M. Madden. Privacy and Data Management on Mobile Devices. Technical report, Pew Research Center, 2012.

[5] C. Bettini D. Riboni, L. Pareschi and S. Jajodia. Preserving Anonymity of Recurrent Location-Based Queries. In *Proceedings of the 16th International Symposium on Temporal Representation and Reasoning*, 2009.

[6] R. Dewri. Local Differential Perturbations: Location Privacy Under Approximate Knowledge Attackers. *IEEE Transactions on Mobile Computing*, 12(12):2360–2372, 2013.

[7] R. Dewri, I. Ray, I. Ray, and D. Whitley. Query m-Invariance: Preventing Query Disclosures in Continuous Location-Based Services. In *Proceedings of the 11th International Conference on Mobile Data Management*, pages 95–104, 2010.

[8] R. Dewri and R. Thurimella. Exploiting Service Similarity for Privacy in Location-Based Search Queries. *IEEE Transactions on Parallel and Distributed Systems*, 25(2):374–383, 2014.

[9] Electronic Privacy Information Center. Locational Privacy. `https://epic.org/privacy/location_privacy/`, 2016. [Online; accessed 12-March-2016].

[10] B. Gedik and L. Liu. Protecting Location Privacy with Personalized k-Anonymity: Architecture and Algorithms. *IEEE Transactions on Mobile Computing*, 7(1):1–18, 2008.

[11] G. Ghinita, P. Kalnis, and S. Skiadopoulos. PRIVE: Anonymous Location-Based Queries in Distributed Mobile Systems. In *Proceedings of the 16th International Conference on World Wide Web*, pages 371–380, 2007.

[12] G. Ghinita, K. Zhao, D. Papadias, and P. Kalnis. A Reciprocal Framework for Spatial k-Anonymity. *Journal of Information Systems*, 35(3):299–314, 2010.

[13] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *Proceedings of the 1st International Conference on Mobile Systems, Applications, and Services*, pages 31–42, 2003.

[14] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preventing Location-Based Identity Inference in Anonymous Spatial Queries. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1719–1733, 2007.

[15] A. Khoshgozaran, C. Shahabi, and H. Shirani-Mehr. Location Privacy: Going beyond k-Anonymity, Cloaking and Anonymizers. *Journal of Knowledge and Information Systems*, 26(3):435–465, 2011.

[16] H. Kido, Y. Yanagisawa, and T. Satoh. An Anonymous Communication Technique Using Dummies for Location-Based Services. In *Proceedings of the IEEE International Conference on Pervasive Services*, pages 88–97, 2005.

[17] F. Liu, K. A. Hua, and Y. Cai. Query l-Diversity in Location-Based Services. In *Proceedings of the 10th International Conference on Mobile Data Management: Systems, Services and Middleware*, pages 436–442, 2009.

[18] D. Mihm. Local Search Ranking Factors. `www.davidmihm.com/local-search-ranking-factors.shtml`, 2012. [Online; accessed 12-March-2016].

[19] M. F. Mokbel, C. Chow, and W. G. Aref. The New Casper: Query Processing for Location Services Without Compromising Privacy. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 763–774, 2006.

[20] B. Niu, Q. Li, X. Zhu, G. Cao, and H. Li. Achieving K-anonymity in Privacy-aware Location-based Services. In *Proceedings of the 33rd Annual IEEE International Conference on Computer Communications*, pages 754–762, 2014.

[21] B. Niu, Q. Li, X. Zhu, G. Cao, and H. Li. Enhancing Privacy through Caching in Location-based Services. In *Proceedings of the 34th Annual IEEE International Conference on Computer Communications*, pages 1017–1025, 2015.

[22] B. O'Clair, D. Egnor, and L. E. Greenfield. Scoring local search results based on location prominence. *US Patent 8046371 B2*, 2011.

[23] S. Papadopoulos, S. Bakiras, and D. Papadias. Nearest Neighbor Search with Strong Location Privacy. *VLDB Endowment*, 3(1-2):619–629, 2010.

[24] H. Shin, J. Vaidya, and V. Atluri. A Profile Anonymization Model for Location Based Services. *Journal of Computer Security*, 19(5):795–833, 2011.

[25] R. Shokri, G. Theodorakopoulos, J.-Y. L. Boudec, and J.-P. Hubaux. Quantifying Location Privacy. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy*, pages 247–262, 2011.

[26] R. Shokri, G. Theodorakopoulos, P. Papadimitratos, E. Kazemi, and J.-P. Hubaux. Hiding in the Mobile Crowd: LocationPrivacy through Collaboration. *IEEE Transactions on Dependable and Secure Computing*, 11(3):266–279, 2014.

[27] R. Shokri, G. Theodorakopoulos, C. Troncoso, J.-P. Hubaux, and J.-Y. L. Boudec. Protecting Location Privacy: Optimal Strategy Against Localization Attacks. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, pages 617–627, 2012.

[28] R. Shokri, C. Troncoso, C. Diaz, J. Freudiger, and J.-P. Hubaux. Unraveling an Old Cloak: k-Anonymity for Location Privacy. In *Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society*, pages 115–118, 2010.

[29] M. Soriano, S. Qing, and J. Lopez. Time Warp: How Time Affects Privacy in LBSs. In *Proceedings of the 12th International Conference on Information and Communications Security*, pages 325–339, 2010.

[30] M. Sullivan. 3G and 4G Wireless Speed Showdown: Which Networks Are Fastest? *PC World*, April 2012.

[31] I. Wald and V. Havran. On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N). In *IEEE Symposium on Interactive Ray Tracing*, pages 61–70, 2006.

[32] T. Xu and Y. Cai. Feeling-Based Location Privacy Protection for Location-Based Services. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 348–357, 2009.