Privacy-Preserving Location-Dependent Query Processing

Mikhail J. Atallah and Keith B. Frikken
Purdue University
CERIAS and Computer Science Department
West Lafayette, IN 47907
{mja,kbf}@cs.purdue.edu

Abstract

A mobile portable device will often make queries, to a remote database, that depend on its location: It may ask for the nearest coffee shop, restaurant, pharmacy, etc. For privacy reasons, the mobile unit may not wish to disclose its precise location to the remote database – while it is unavoidable that the cell phone company already knows the rough location of the customer ("somewhere in Lafayette"), it is quite another matter if the customer's precise location can be tracked over time through his pattern of location-dependent queries to the remote database. This paper describes an efficient protocol, between the client and database, through which a client can learn the answer to its location-dependent query without revealing to the remote database anything about his location, other than what the database can infer from the answer it gives to the query (which is unavoidable). We also analyze the performance of some other, simpler solutions, that do not require the database to run a protocol with the client, but that can reveal more information about the private location and also introduce inaccuracies in the answer – we quantify how much error these simpler schemes introduce in the answer.

1. Introduction

One of the drawbacks of pervasive computing is that it can mean invasive, detailed tracking of individuals. The tracking information could possibly be used for unintended or unauthorized purposes. For a mobile device to ask for a location-dependent information from a remote database would seem to necessitate revealing the location of that mobile device: How can the remote database return the address of the post-office that is nearest to the mobile unit, without knowing precisely the location of that mobile unit? This paper describes techniques for doing this. The main reason for not wanting to reveal one's location information is privacy: Whereas the very act of using a wireless device inherently reveals (e.g., to the cell phone company) the general area of the mobile unit's location at a coarse level of granularity, the database query may be formulated using finer-grain location information, and the user may be reluctant to unnecessarily reveal such detailed information to the many remote databases that he uses. This paper deals with how to answer such location-dependent queries without revealing to the remote database one's fine-grain location information. The solutions in this paper are efficient in terms of communication in that they have much less communication than the trivial private solution of sending the entire database to the client.

There has been much previous work on location-dependent query processing (see, for example, [2, 7, 8, 9, 11, 15, 16, 17], to name just a few – they contain a more exhaustive list of references to related work). Our work differs from the existing literature in that we put most emphasis on privacy, and in such a first step do not consider many of the other issues that the previous location-dependent query-processing literature concerned itself with (such as continuous motion over time, the use of caching and related performance issues, updating strategies for handling rapidly changing information, . . .etc). For future work, it would be interesting to combine privacy-preservation with the elegant previous techniques already documented in the existing literature on location-dependent query processing. Our contribution, for now, is mainly in efficiently achieving privacy-preservation in the processing of queries, and quantifying the tradeoffs involved therein.

The next few sections (Sections 2 to 4) give the simpler solutions, that typically have lower overhead, but at the costs of a lower-quality answer to the query and of more potential for leakage of information about the private location. The protocol-based solution is given in Section 5, and shows how the client can obtain exactly the same answer to the query *as if the database knew the private location, yet without revealing anything about that private location to the database* (other than, of course, what the database can infer from the answer it gives to the query, e.g., from the location

of the nearest post-office).

To make the discussion that follows specific, we focus on proximity queries of the "nearest-neighbour" type, such as "give me the address of the post-office that is nearest to my current position". Our protocol-based approach of Section 5 can handle more general queries, as will be pointed out.

2. Random Perturbation of the Client's Position

This section presents a solution that is simple in the sense that it does not require any modification to the way the database does its query-processing. It suffers from drawbacks that will be pointed out below.

Let $\vec{P}$ denote the position vector of the client, given as a pair of geographic coordinates. The database contains *sites* of the kind specified in the query (e.g., "gas station", or "post-office"), and the answer to the query is the database site nearest to $\vec{P}$. The results of this section hold for any distance metric, not only the Euclidean one.

A simple solution consists of the following steps:

1. The client selects a distance δ large enough that the client deems it acceptable if his location was known by the remote database with an error of δ. Note that δ is not known to the database, and may vary from one query to the next even for the same client (because the privacy/accuracy tradeoff for that client may change over time, or from one query to the next).

2. The client generates a random vector $\vec{Q}$ of length $||\vec{Q}|| = \delta$, and sends as query the "fake" position $\vec{P} + \vec{Q}$.

3. The database responds with a position vector $\vec{R}$ of the database site nearest to $\vec{P} + \vec{Q}$.

The privacy parameter δ changes from one query to the next, depending on how important privacy is to the client at that moment of time, relative to the importance of an accurate answer (if he is very low on gas and wants the nearest gas station then he may choose $\delta = 0$).

We now quantify how much "damage" is done, to the quality of the answer, by the perturbation distance of δ. Let $\vec{S}$ be the true answer, the one that would be returned by the database if δ had been zero. The *damage* is the difference between the two distances $\vec{P}$-to-$\vec{R}$ and $\vec{P}$-to-$\vec{S}$ where $\vec{R}$ is the answer returned by the database in Step 3. In other words, the damage is

$$||\vec{P} - \vec{R}|| - ||\vec{P} - \vec{S}||.$$

2.1. Worst-case analysis

This subsection deals with the worst-case value of the damage to the query's answer, that results from perturbing the query's location by δ.

Theorem 1 *If, in a nearest-neighbour proximity query, the client position is randomized by adding to it a random vector of length δ, then the damage to the answer is no greater than 2δ:*

$$||\vec{P} - \vec{R}|| - ||\vec{P} - \vec{S}|| \leq 2\delta.$$

Furthermore, this bound of 2δ is tight, i.e., there is an example where it is achieved.

Proof. We begin with the proof of the upper bound of 2δ on the damage. The vectors $\vec{P}$, $\vec{Q}$, $\vec{R}$, $\vec{S}$ are as defined in the above query-processing algorithm. If $\vec{R} = \vec{S}$ then the damage is zero, hence smaller than 2δ and the proof is done. So assume henceforth that $\vec{R} \neq \vec{S}$.
First, observe that:

$$||\vec{P}-\vec{R}|| = ||\vec{P}+\vec{Q}-\vec{Q}-\vec{R}|| \leq ||\vec{Q}||+||\vec{P}+\vec{Q}-\vec{R}|| = \delta + ||\vec{P} + \vec{Q} - \vec{R}|| \quad (1)$$

where the triangle inequality was used.
Now, the fact that the answer returned is $\vec{R}$ rather than $\vec{S}$ implies the following

$$||\vec{P} + \vec{Q} - \vec{R}|| \leq ||\vec{P} + \vec{Q} - \vec{S}||$$

which, using the triangle inequality, gives

$$||\vec{P} + \vec{Q} - \vec{R}|| \leq ||\vec{Q}|| + ||\vec{P} - \vec{S}|| = \delta + ||\vec{P} - \vec{S}|| \quad (2)$$

Combining (1) and (2) gives

$$||\vec{P} - \vec{R}|| \leq \delta + \delta + ||\vec{P} - \vec{S}||$$

which gives

$$||\vec{P} - \vec{R}|| - ||\vec{P} - \vec{S}|| \leq 2\delta.$$

This completes the proof of the upper bound.

We now give an example that achieves the bound of 2δ just proved. It suffices to give a one-dimensional example (in which case the vectors are scalars). Choose $S = 0$, $Q = +\delta$, $R = 6\delta$, and $P = 2\delta + \epsilon$ where ϵ is arbitrarily small (much smaller than δ). The database gets queried for the site nearest to $P + Q = 3\delta + \epsilon$, so it returns the site $R = 6\delta$. The site nearest to P is $S = 0$. The damage in that case is

$$|P - R| - |P - S| = |2\delta + \epsilon - 6\delta| - (2\delta + \epsilon) = (4\delta - \epsilon) - 2\delta - \epsilon = 2\delta - 2\epsilon$$

which goes to 2δ as ϵ is made arbitrarily small. QED

2.2. Average-case analysis

The case of worst-case damage is unlikely to occur, in fact it has measure zero in a probabilistic model of random (understood henceforth to mean independent and uniformly distributed) sites, queries, and "perturbation vectors" $\vec{Q}$ (with a fixed modulus δ for the perturbation, so only its direction is random). In such a probabilistic model, what is the expected value of the damage? This section provides an answer, under one assumption we need to make so as to handle the case of the perturbation vector $\vec{Q}$ taking the query "outside the map", i.e., outside the region in which the n sites lie; to avoid this troublesome situation, and for the sake of making the analysis tractable, we henceforth assume that the map wraps around as on the surface of a sphere (hence no perturbation can result in crossing the map's boundary, as there is no boundary). If the map in reality does have a boundary, then our analysis still has some validity as an approximation because what it means then is that we are only concerning ourselves with points that are "well within" the bounding box containing the n sites, that we exclude from the analysis "boundary effects" caused by perturbations that cause boundary-crossing, or due to sites close to the boundary of the bounding box. In such a case, the justification for excluding the boundary sites from the analysis is that they are a negligible fraction of the total number n of sites – in fact they are zero percent for an infinitely large n (the ratio of boundary to total has a $\sqrt{n}$ in its denominator and hence goes to zero as n goes to infinity).

Theorem 2 *If, in a nearest-neighbour proximity query, the client position is randomized by adding to it a random vector of length δ, then the expected damage is no larger than δ, that is,*

$$E(||\vec{P} - \vec{R}|| - ||\vec{P} - \vec{S}||) \leq \delta$$

.

Proof. First, observe that the "nearest-neighbour" distance statistics are location-independent. This implies the following

$$E(||\vec{P} - \vec{S}||) = E(||\vec{P} + \vec{Q} - \vec{R}||) \quad (3)$$

because $\vec{S}$ is to $\vec{P}$ what $\vec{R}$ is to $\vec{P} + \vec{Q}$ (namely, the site nearest to it). Now observe that

$$||\vec{P} - \vec{R}|| = ||\vec{P} + \vec{Q} - \vec{Q} - \vec{R}|| \leq ||\vec{Q}|| + ||\vec{P} + \vec{Q} - \vec{R}|| \quad (4)$$

where the triangle inequality was used. Taking expectations in Equation (4) and then using Equation (3) gives

$$E(||\vec{P} - \vec{R}||) \leq \delta + E(||\vec{P} - \vec{S}||)$$

and we therefore have:

$$\text{Expected Damage} = E(||\vec{P} - \vec{R}|| - ||\vec{P} - \vec{S}||) \leq \delta$$

which completes the proof. QED

2.3. Discussion of location perturbation

The above approach, of randomly perturbing the client's position, becomes unsatisfactory when one realizes that it is possible (as we show in Section 5) to achieve privacy without any compromise of the quality of the answer: Zero damage, yet without revealing anything about one's position $\vec{P}$ (other that what can be inferred about it from the answer to the query). This will come at the expense of both sides having to run a special protocol for the purpose of achieving this goal, whereas the above simple scheme did not require any modification to the way the database handles queries. But before we present the powerful approach of Section 5, we briefly describe in the next two sections other possible approaches to the problem: The approach in Section 3 is related to (and suffers from similar drawbacks) the above perturbation method, whereas the approach in Section 4 results in zero damage but requires the client to trust that one (or more) intermediaries will not collude with the database.

3. Grid Method

In this section, we present a variation on the scheme presented in the previous section: Unlike the previous section, this variant does not result in any loss of accuracy, but it potentially requires more communication. The idea behind this scheme is to "grid" the plane, covering it with tiles of dimensions $\lambda \times \lambda$; after this gridding of the plane, the client queries the database with the tile that contains the client's location. The database answers the query with all sites that are closest to at least one point in the query tile; that is, if v is any point of the query tile (not necessarily a site) and site w is the closest site to v, then w is a part of the answer that the database will return to the client (note that w could be inside the query tile, or outside of it, and that a site inside the query tile is always chosen as a part of the answer). Upon receiving these sites the client determines which of them is closest to his actual location. The disadvantage of this scheme is that the client has potential to receive many sites in response to the query – the expected number received depends on λ but also on the average density ρ of sites per unit area (the two determine the expected number of sites per tile, which is $\lambda^2 d$). Note that, if the n points are inside a $\Delta \times \Delta$ bounding box, then $\rho = n/\Delta^2$. It would be interesting to determine precisely the expected number of sites returned (with the correct constant factors), assuming a randomly selected query tile and (as usual) uniformly distributed sites. Since we already know that points inside the query tile are always included in the answer, their expected number $\lambda^2\rho$ ($= n\lambda^2/\Delta^2$) is a lower bound on the expected number of sites returned.

A refinement of the above scheme is to have the database treat the answers that would be returned by the above

scheme merely as "candidates" for the one site that is returned as answer: The site that has the largest number of "votes" from within the tile. In other words, if v and w are as above, then the winning candidate w is the one with the largest number of v's in the tile that "choose it" as the nearest site to them. This variant, which we call *one-answer variant* of the grid method, does not have the increased communication because a single site is returned as answer, but it does have an accuracy tradeoff that is quantified below.

3.1. Worst-case analysis of one-answer variant

Theorem 3 *In the one-answer variant of the grid method, the worst-case damage to a query's answer is no greater than the tile diameter D. Furthermore, this bound is tight, i.e., there is an example where it is achieved.*

Proof. In what follows we use $\vec{P}$ to denote the client's location vector, $\vec{C}$ to denote the position vector of the tile's centroid, $\vec{R}$ to denote the database's answer vector, and $\vec{S}$ to denote the true answer to the client's query. If D denotes the diameter of a tile, then we must prove that

$$||\vec{P} - \vec{R}|| - ||\vec{P} - \vec{S}|| \leq D.$$

Since the above holds trivially if $\vec{R} = \vec{S}$, we henceforth assume that $\vec{R} \neq \vec{S}$.

We claim that the answer $\vec{R}$ that is returned is the site that is closest to the centroid of the tile, i.e., it is the centroid that determines that answer. We prove the claim by contradiction. Suppose that, contrary to the claim, the site closest to position $\vec{C}$ is $\vec{R}'$ where

$$||\vec{R}' - \vec{C}|| < ||\vec{R} - \vec{C}||.$$

Let line L be the perpendicular bisector of the line segment that joins positions $\vec{R}'$ and $\vec{R}$. The centroid is on the same side of that line L as position $\vec{R}'$ (because the centroid is closer to position $\vec{R}'$ than to position $\vec{R}$); without loss of generality, assume that both the centroid and $\vec{R}'$ are below line L (hence position $\vec{R}$ is above that line L). The fact that $\vec{R}$ rather than $\vec{R}'$ was returned as the answer implies that the area of the query tile that is above L is larger than the area of the query tile that is below L (otherwise the answer would have been $\vec{R}'$), which requires that the centroid is above L, contradicting the fact that the centroid is below L. This proves the claim, and implies that the centroid's distance to site $\vec{R}$ is no greater than the centroid's distance to site $\vec{S}$:

$$||\vec{R} - \vec{C}|| \leq ||\vec{S} - \vec{C}|| = ||\vec{S} - \vec{P} + \vec{P} - \vec{C}||$$

which, using the triangle inequality, gives

$$||\vec{R} - \vec{C}|| \leq ||\vec{S} - \vec{P}|| + ||\vec{P} - \vec{C}|| \quad (5)$$

Now observe that:

$$||\vec{P} - \vec{R}|| = ||\vec{P} + \vec{C} - \vec{C} - \vec{R}|| \leq ||\vec{P} - \vec{C}|| + ||\vec{C} - \vec{R}|| \quad (6)$$

where the triangle inequality was used.
Combining (5) and (6) gives

$$||\vec{P} - \vec{R}|| \leq 2||\vec{P} - \vec{C}|| + ||\vec{S} - \vec{P}|| \leq D + ||\vec{S} - \vec{P}||$$

where we used the fact that the distance from the centroid to any point inside the tile is at most half the tile diameter, i.e., that $||\vec{P} - \vec{C}|| \leq D/2$. Re-arranging the above gives

$$||\vec{P} - \vec{R}|| - ||\vec{S} - \vec{P}|| \leq D$$

which completes the proof of the upper bound.

We now give an example that achieves the bound of D that was just proved. We give the example for the Euclidean distance, so that $D = \lambda\sqrt{2}$, but it is easy to create similar examples for other distance metrics. Consider a query tile whose centroid is the origin of coordinates $(0, 0)$. Let the client position be at $(\epsilon - \lambda/2, \epsilon - \lambda/2)$, i.e., near the bottom-left corner of the tile. Let the only two sites be at respective positions $(-\lambda, -\lambda)$ and $(\lambda - \epsilon, \lambda - \epsilon)$ where ϵ is arbitrarily small. In this example the correct answer is $\vec{S} = (-\lambda, -\lambda)$ but the answer that is actually returned is $\vec{R} = (\lambda - \epsilon, \lambda - \epsilon)$. The damage for this specific example is

$$||\vec{P} - \vec{R}|| - ||\vec{P} - \vec{S}|| = (1.5D - 2\epsilon\sqrt{2}) - (0.5D + \epsilon\sqrt{2}) = D - 3\epsilon\sqrt{2}$$

which goes to D as ϵ is made arbitrarily small. QED

Corollary 1 *In the one-answer variant of the grid method, using the Euclidean distance metric, the worst-case damage to a query's answer is no greater than $\sqrt{2}\lambda$. Using the Manhattan distance L_1 the worst-case damage is no more than 2λ. Using the L_∞ distance it is no more than λ.*

3.2. Average-case analysis of one-answer variant

The probabilistic model we use is of random (uniformly distributed) sites and random client locations. Tile size is fixed at λ. We no longer need the "wraparound" assumption which was made in the random-perturbation section (we cannot "fall outside the map" in the grid model).

Theorem 4 *In the one-answer variant of the grid method, the expected damage to a query's answer is no larger than the expected distance between a random point in a tile, and the center of that tile. For the Euclidean metric, this bound is*

$$(\lambda/6)(\sqrt{2} - 0.5\ln((\sqrt{2} - 1)/(\sqrt{2} + 1))$$

which is approximately $0.3826\lambda = 0.27D$.

Proof. As before, we use $\vec{P}$ to denote the client's location vector, $\vec{C}$ to denote the (position vector of) the query tile's centroid, $\vec{R}$ to denote the database's answer vector, and $\vec{S}$ to denote the true answer to the client's query. Let $T(\vec{U}, \vec{W})$

denote the perpendicular bisector of the line segment between site positions $\vec{U}$ and $\vec{W}$. Since site and query positions are random (independent and uniformly distributed), the probability that $\vec{C}$ and $\vec{P}$ lie on the same side of any such line $T(\vec{U}, \vec{W})$ is the same as the probability that they are on opposite sides of it. This implies that the probability that $\vec{R} = \vec{S}$ is 0.5, in which case the damage is zero. The other case, when $\vec{C}$ and $\vec{P}$ are on opposite sides of $T(\vec{R}, \vec{S})$, has probability 0.5 and is analyzed next.

Following the same geometric-inequality steps as in the proof of Theorem 3, up until the place where that proof combines (5) and (6), gives

$||\vec{P} - \vec{R}|| \leq 2||\vec{P} - \vec{C}|| + ||\vec{S} - \vec{P}||.$

Re-arranging the above gives gives

Damage $= ||\vec{P} - \vec{R}|| - ||\vec{P} - \vec{S}|| \leq 2||\vec{P} - \vec{C}||.$

When computing the contribution of the above case to the overall expected damage, we must multiply by 0.5 to account for the probability of that case. Doing so gives us an overall expected damage of

Overall Expected Damage $\leq$

$0.5 * 2*$ (Expected Distance Between $\vec{P}$ and $\vec{C}$)

which proves the claimed bound. The exact value of the expected distance between a random point in a $\lambda \times \lambda$ tile and the center of that tile, varies depending on the distance metric considered. For the Euclidean distance, a straightforward analysis shows that it is as claimed in the theorem's statement. QED

3.3. Discussion of the grid method

Comparing the grid method to the random perturbation method of the previous section, we note that the λ for the grid method has to be known to the database (otherwise it cannot process the query), whereas the δ of the perturbation method was not known to the database.

The grid method is also more rigid for the following practical reasons. If the tiling is rigid and λ fixed, then the database can pre-compute, off-line, the site-proximity set of each tile, making it possible to subsequently perform on-line query-processing very fast (because processing a query tile is then a simple lookup operation). If, however, the tiling is not fixed, then it can be expensive (in terms of query-processing computations at the database's end) to allow the client to dynamically change the value of λ from one query to the next – this would mean more computational overhead that cannot be pre-computed by the database. (As mentioned earlier, the client may wish to adjust the λ parameter from one query to another, as his privacy/accuracy tradeoff valuation changes.) One compromise solution would be to have a fixed menu of k available λ values that the client could choose from, which would give some flexibility to the client while the database retains the ability to do off-line pre-processing of each tiling's site-proximity information; of course the database now has k separate tilings to maintain.

4. Anonymization

Another approach consists of accurately revealing to the database the client's location but hiding from the database the identity of the client. This can be done by interposing an anonymizing intermediary – a mix [3] – between the clients and the database, so the database knows the exact client location for each query but does not know who asked the query (thus there is no damage to the quality of the answer to the query). The intermediary knows the client is asking *some* query from the database but does not learn anything about the nature of the query or its parameters because the client-to-database traffic is encrypted [3] in both directions after the establishment of a session key through the usual cryptographic techniques [14]. Achieving the necessary "hiding in a crowd" effect, that is needed for foiling traffic analysis, requires that queries from many clients are handled by the same intermediary. Making such an intermediary act like a mix [3] provides resistance to traffic analysis: An eavesdropper could not make the connection between the traffic incoming from the many clients and the queries outgoing to the (possibly many) database(s).

The main problem with this kind of approach is its vulnerability to misbehavior by an intermediary: Even though the use of encryption easily hides from the intermediary the query coming from each client (only the database can decrypt it), all is lost if the intermediary were to collude with the database (the database could then associate the client's query and location with the client's identity). A cascade of mixes increases the security (collusion by all intermediaries would be needed for the system to fail), but also the cost and complexity.

Another drawback is that implementing a mix is a complicated business. To achieve a sort of "hiding in a crowd" without a mix, a simple solution would be for the client to send several locations to the database. All that the database learns is that the client is at one of several locations. This could be modified to use the techniques in Section 2 (i.e., the client sends several perturbed points to the database). The advantages of this scheme are its simplicity in that the database does not need to do anything other than answer queries. However, the disadvantages are that the level of privacy-protection is minimal and the communication complexity is larger than previous solutions.

Contrast the above approach (with or without using a mix) with the powerful approach of the next section, where there are only 2 parties involved (the client and the database) and there is therefore no vulnerability to collusion: The client's private location cannot become known to

the database, yet he gets the same quality of answer to his location-dependent query as if the database knew his location.

5. The Protocol-Based Solution

As before, we illustrate the technique using nearest-neighbour proximity queries, but it can be used for other query criteria as long as they involve a partitioning of the map into regions, and processing a query point requires finding the region containing that point. One of the lessons of this section is that, at least for this problem, speed and efficiency inherently give better security; this is somewhat surprising, in view of our experience with so many other situations where there is a security-performance tradeoff.

5.1. Pre-processing done by the database

This subsection describes how the database processes the information at its end, so that it can later run the privacy-preserving query-processing protocol with the remote client. The following steps construct the data structure once, and then update it (additions/deletions of sites) incrementally later on. Therefore the initial $O(n \log n)$ construction time, and the subsequent polylogarithmic updates as site insertions and deletions occur, can be amortized later on over a large number of queries coming from many clients. Each query takes $O(\log n)$ amount of communication for its processing by the protocol.

The data structure used is a hierarchical-search directed acyclic graph (DAG) for query point location in a planar subdivision [10], where the planar subdivision is a Voronoi Diagram [6, 13, 12] of the sites at the database. Other location-dependent query processing papers [17] also make use of Voronoi Diagrams; our use of these has a different emphasis – as stated earlier, here we put most emphasis on privacy, whereas other considerations were more central in the previous location-dependent query-processing literature. We are constrained in our use of this DAG search structure by the strict privacy requirement, namely, that the database should not learn anything other than what it can infer from the query's answer; this rules out revealing such things as whether the query point is closer to one non-answer site than to another, or revealing the specific reason for which the query point is outside of a Voronoi cell (only yes/no is allowed), etc.

The search DAG is constructed as follows.

1. The set S of n sites in the database are processed so as to produce a Voronoi Diagram [6, 13, 12], a subdivision of the plane into cells such that each cell is associated with one of the sites p of S and consists of that portion of the plane that is closer to p than to any of the sites in $S - \{p\}$. Each such cell is therefore a closed convex region that has a polygonal boundary, because it is (by definition) the intersection of $n - 1$ closed half-planes. The Voronoi Diagram of S is the union of all these cell boundaries, which is an embedded planar graph with n faces, one for each site. Hence the Voronoi Diagram has $O(n)$ edges and vertices. Detailed discussions of Voronoi Diagrams, and of the $O(n \log n)$ time algorithm for their construction, abound in the literature (see [6, 13] for a textbook-style presentation and an extensive list of references). There are also techniques for dynamically updating a Voronoi diagram as sites are added and removed.

2. The planar subdivision (representing the Voronoi Diagram) is triangulated.

3. A planar-point-location search data structure DAG is built on the Voronoi Diagram of S, a structure of logarithmic depth and linear size, that supports point location queries: Given any query point p, the structure can in logarithmic time find the Voronoi cell that contains that query point p. The structure DAG is a hierarchical directed acyclic graph, that is, a graph whose vertex set can be partitioned into $h = O(\log n)$ levels, $L_0, \ldots, L_h$, such that every directed edge is from some L_i to L_{i+1}, $|L_0| = 1$, and $c_1\mu^i \leq |L_i| \leq c_2\mu^i$, for some $\mu > 1$ and positive constants c_1 and c_2. The search starts at L_0 and proceeds to a vertex in L_1, then to one in L_2, etc. It ends at L_h, in the Voronoi cell that contains the query point p and, because of the definition of a Voronoi Diagram, that Voronoi cell containing p corresponds to the site that is nearest to p: That site is the answer to the nearest-neighbor proximity query. For the details of this data structure DAG and of how it can do all of the above, see the paper by Kirkpatrick [10] (see also [6, 13]). Each of the comparisons during the search consists of comparing the point p to $O(1)$ triangles to determine in which triangle it lies (is is guaranteed to lie in one of these triangles): The triangle in which it lies determines to which node of DAG the search moves down for the next comparison.

The following details in the operation of DAG [10, 6] will have consequences for the security of the scheme.

- During the processing of a query, not every edge taken from an L_i to L_{i+1} necessitates point-to-triangle inclusion comparisons: Some do not require such a comparison, because the triangle of L_{i+1} that contains p can be the same as the triangle of L_i that contains p. So the number of levels where such comparisons are done, during the processing of a query, may be less than h.

- Even when moving from L_i to L_{i+1}, during the processing of a query does necessitate point-to-triangle inclusion comparisons, the number of such comparisons is variable (although upper-bounded by a constant d).

The reason for the above two items is that DAG is constructed "bottom up" starting with L_h and ending with L_0, and the construction of L_i from L_{i+1} is done by

1. removing an independent set of $O(1)$-degree vertices from L_{i+1} (vertices that do not share a common edge), and then

2. re-triangulating.

The fact that h is logarithmic is because the independent set removed in the above Step 1 has a number of vertices linear in the size of L_{i+1}, that is, a fixed fraction of L_{i+1} is removed. The fact that processing a query p at a level L_i involves $O(1)$ point-to-triangle inclusion comparisons, is because each vertex v of the independent set removed in Step 1 has degree $O(1)$, i.e., it is upper-bounded by a constant d because each new triangle, created in the re-triangulation of Step 2 after the removal of v, does not overlap with more than $degree(v)$ old triangles. Now, clearly not all of the triangles of L_{i+1} are impacted by the removal of that independent set in Step 1, and therefore some triangles of L_{i+1} will remain, unchanged, in L_i; if, during a top-down search for a query point p, that point is in such a triangle common to L_i and L_{i+1}, then moving the query from L_i to L_{i+1} does not require any new point-to-triangle inclusion comparison.

The implication of the above is that the total number of point-to-triangle inclusion comparisons, that are involved in processing a particular query, is not entirely predictable a priori to the client, except for the knowledge that it is upper-bounded by hd where d is the maximum degree of a vertex of the independent set mentioned in the above Step 1. This should be kept in mind for the query-processing discussion in the next section.

5.2. Processing a query

As should be clear from the above, we need a way for the client and database to cooperatively allow the database to determine whether a query point p that is known to the client but not to the database, is inside a triangle T that is known to the database but not to the client. A protocol was given in [1] for doing this without revealing to the database anything other than the Yes/No answer to the question of whether the client's query point is in T or not. Here we use this protocol up to hd times, at most d times for each step of the h-step path through DAG, starting at L_0 and ending at L_h.

5.3. Security Analysis

The above approach of building and searching the hierarchy DAG works well in the *honest-but-curious* model of the participants, which assumes that the participants will follow the protocols exactly, but may try to use the data the protocol makes available to them to compute things they are not supposed to know. The honest-but-curious model may be a reasonable one for this kind of application: Not following the protocol (i.e., not being "honest") is a more detectable activity than merely being "curious" (which would involve harder-to-detect computations done "on the side" for the purpose of learning what the protocol foolishly reveals). Dishonesty leaves more of a trail (audit records, or employees who could whistle-blow, etc) than curiosity. A database operated by a public company cannot afford to be caught deviating from the agreed-upon protocol, for a variety of reasons (regulatory, fear of embarrassment and of damage to the corporate image, fear of lawsuits, etc).

But there is a problem with this whole approach if more powerful models of the adversary are assumed, e.g., if the database can be malicious rather than merely honest-but-curious. A malicious database can *continue running the query protocol with the client even though it already knows the answer to the query*, thereby refining its knowledge of the client's private location; we call this the *continuous refinement attack*. Note that we do not, on the other hand, worry about the database not following the structure of DAG to guide the search because that would be self-defeating – DAG is an asymptotically optimal way of doing point location. As will be pointed out below, the efficiency of DAG is a form of protection, increasing the security of the scheme by making it difficult for the adversary to deviate from the agreed-upon use of DAG (more on this below).

To prevent the above continuous refinement attack by a malicious adversary, the client has to store hd, and must refuse to engage in more than hd rounds of point-triangle inclusion comparisons. But this still has drawbacks:

- It requires the client to store the hd bound for the database. This is burdensome as it must be done securely (not just by the database "informing" all clients of a new hd value), but it is mitigated by the fact that such changes are rare because d is fixed (it is a characteristic of Kirkpatrick's algorithm [10]), and although h can change over time as the database evolves (it increases as sites are added to S, decreases as sites are removed from S), these changes in h occur rarely (e.g., n must roughly double before h increases by 1).

- Because the number of point-to-triangle inclusion comparisons for processing a query can be less than hd, a malicious database can continue (even after it learns the answer) until it uses up the full hd quota of

comparisons – it uses the extra point-to-triangle inclusion comparisons to refine its knowledge of the client's location to a smaller portion of the Voronoi cell that contains it (i.e., it cheats).

- If the malicious database discovered a new data structure, better than DAG, for processing a query with t point-to-triangle inclusion comparisons, where t is much smaller than hd, then it could use that data structure rather than DAG (without telling the client that it is doing so), thereby obtaining $t - hd$ additional point-to-triangle inclusion comparisons as extra "ammunition" in the above-mentioned continuous refinement attack. It is therefore fortunate that DAG is asymptotically optimal (although there are in fact documented constant-factor improvements in its performance). An optimal DAG, and the correspondingly low hd count stored at the client, would leave no room for the adversary to obtain such $t - hd$ extra comparisons for its attack. This is a situation where better efficiency in DAG implies higher security for the overall scheme.

We are not aware of any data structure for point location queries in a planar subdivision (Voronoi or otherwise) that processes every query in precisely the same number of comparison operations, rather than with a variable number that is upper-bounded by $O(\log n)$. This was not an issue in the previous planar point location literature, because there was no reason for it to be an issue: optimality "to within a constant factor" was the goal. It is the need for privacy-preservation that, as we pointed out, brings to the fore such a new requirement, both for planar point location and for other problems. But even if we were somehow able to design a hypothetical data structure $\mathcal{T}$ that does manage to process every query in precisely the same predictable number of comparison operations, there is no way to know that the database is actually using $\mathcal{T}$ and not DAG if both involve point-to-triangle inclusion comparisons, *unless* $\mathcal{T}$ is much better than DAG and results in a lower count being stored in the client (in that case the database would have to use $\mathcal{T}$).

6. Concluding Remarks

This paper described schemes through which a client can obtain an answer to a location-dependent query (one whose answer depends on the client's location), without revealing the client's location to the remote database that processes the query. The more elaborate of our schemes requires the remote database to run a special protocol with the client, as a result of which the client gets the correct answer, yet the remote database learns nothing about the client's location (other than what it can deduce from the answer to the query, which is unavoidable). We also give simpler solutions, that do not require the database to run a special protocol with the client. These involve obfuscation (through random perturbation, or tiling, etc) of the client's exact location, and introduce inaccuracies in the answer; we analyze and quantify the relation between these inaccuracies and the amount of obfuscation.

References

[1] M. J. Atallah and W. Du, "Secure Multi-Party Computational Geometry," *Proc. 2001 Workshop on Algorithms and Data Structures (WADS)*, Providence, Rhode Island, August 2001. Lecture Notes in Computer Science, Vol. 2125, Springer Verlag, pp. 165–179.

[2] H. Cao, S. Wang, and L. Li, "Location Dependent Query in a Mobile Environment," *Information Sciences: An International Journal* (special issue on multimedia and mobile agents), Elsevier Science, New York, Volume 154, 2003, pp. 71–83.

[3] D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Communications of the ACM*, Vol. 24, No. 2, February 1981, pp. 84–88.

[4] N. Dadoun and D. Kirkpatrick, "Parallel Processing for Efficient Subdivision Search," *Proc. 3rd ACM Symp. Computational Geom.*, 1987, pp. 205–214.

[5] N. Dadoun and D. Kirkpatrick, "Parallel Construction of subdivision hierarchies," *J. of Computer and System Sciences*, Vol. 39, 1989, pp. 153–165.

[6] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag (New York: 1987).

[7] S. Ilarri , E. Mena , A. Illarramendi, "A System based on Mobile Agents for Tracking Objects in a Location-dependent Query Processing Environment," *Proceedings of the 12th International Workshop on Database and Expert Systems Applications*, September 2001, pp. 577-581.

[8] S. Ilarri , E. Mena , A. Illarramendi, "Dealing with Continuous Location-Dependent Queries: Just-in-Time Data Refreshment," *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications 2003 (PerCom 2003)*, March 2003, pp. 23–26.

[9] S. Ilarri , E. Mena , A. Illarramendi, "A System Based on Mobile Agents for Tracking Objects in a Location-Dependent Query Processing Environment," *Proceedings 12th International Workshop on Database and Expert Systems Applications*, Sept. 2001, pp. 577–581.

[10] D. G. Kirkpatrick, "Optimal Search in Planar Subdivisions," *SIAM J. Comput.*, Vol. 12, 1983, pp. 28–35.

[11] G. Kollios, D. Gunopulos, V. J. Tsotras, "Nearest Neighbor Queries in a Mobile Environment," *Proceedings of the International Workshop on Spatio-Temporal Database Management*, September 1999, pp. 119-134.

[12] D. T. Lee, R. L. Drysdale III, "Generalization of Voronoi Diagrams in the Plane," *SIAM J. Computing*, 10(1), 1981, pp. 73–87.

[13] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer Verlag, 1993.

[14] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C (Second Edition)*, Wiley.

[15] A. Y. Seydim, M. H. Dunham, and V. Kumar, "Location Dependent Query Processing," in S. Banerjee, P. Chrysanthis, and E. Pitoura, editors, *Proceedings Second ACM International Workshop on Data Engineering for Mobile and Wireless Access (MobiDE'01)*, Santa Barbara, California, May 2001, pp. 47–53.

[16] A. Y. Seydim, M. H. Dunham, and V. Kumar, "An Architecture for Location Dependent Query Processing," *Proceedings of the 12th International Workshop on Database and Expert Systems Applications*, September 03-07, 2001, pp. 549-555.

[17] B. Zheng , D. L. Lee, "Processing Location-Dependent Queries in a Multi-Cell Wireless Enviroment, *Proceedings of the 2nd ACM international workshop on Data engineering for Wireless and Mobile Access*, Santa Barbara, California, May 2001, pp. 54–65.