Anonymous Query Processing in Road Networks

Kyriakos Mouratidis, Man Lung Yiu

Abstract—The increasing availability of location-aware mobile devices has given rise to a flurry of location-based services (LBS). Due to the nature of spatial queries, an LBS needs the user position in order to process her requests. On the other hand, revealing exact user locations to a (potentially untrusted) LBS may pinpoint their identities and breach their privacy. To address this issue, spatial anonymity techniques obfuscate user locations, forwarding to the LBS a sufficiently large region instead. Existing methods explicitly target processing in the Euclidean space, and do not apply when proximity to the users is defined according to network distance (e.g., driving time through the roads of a city). In this paper, we propose a framework for anonymous query processing in road networks. We design location obfuscation techniques that (i) provide anonymous LBS access to the users, and (ii) allow efficient query processing at the LBS side. Our techniques exploit existing network database infrastructure, requiring no specialized storage schemes or functionalities. We experimentally compare alternative designs in real road networks and demonstrate the effectiveness of our techniques.

1 Introduction

The low cost and small size of positioning equipment (e.g., GPS receivers) have allowed their embedding into PDAs and mobile phones. The wide availability of these location-aware portable devices has given rise to a flourishing industry of location-based services (LBS). An LBS makes spatial data available to the users through one or more *location servers* (LS) that index and answer user queries on them. Examples of spatial queries could be "Where is the closest hospital to my current location?" or "Which pharmacies are open within a 1 km radius?". In order for the LS to be able to answer such questions, it needs to know the position of the querying user.

There exist many algorithms for efficient spatial query processing, but the main challenge in the LBS industry is of a different nature. In particular, users are reluctant to use LBSs, since revealing their position may link to their identity. Even though a user may create a fake ID to access the service, her location alone may disclose her actual identity. Linking a position to an individual is possible by various means, such as publicly available information (e.g., city maps and telephone directories), physical observation, cell-phone signal triangulation, etc. User privacy may be threatened because of the sensitive nature of accessed data; e.g., inquiring for pharmacies that offer medicines for diseases associated with a social stigma, or asking for nearby addiction recovery groups (Alcoholics/Narcotics Anonymous, etc). Another source of threats comes from less sensitive data (e.g., gas station locations, shops, restaurants, etc) that may reveal the user's interests and shopping needs, resulting in a flood of unsolicited advertisements through e-coupons and personal messages.

- *K. Mouratidis is with the School of Information Systems, Singapore Management University, Singapore 178902. E-mail: kyriakos@smu.edu.sg*
- *M.L. Yiu is with the Department of Computer Science, Aalborg University, DK-9220 Aalborg, Denmark. E-mail: mly@cs.aau.dk*

To solve this problem the following general approach is taken [28], [23]. When a user u wishes to pose a query, she sends her location to a trusted server, the *anonymizer* (AZ), through a secure connection (e.g., SSL). The latter obfuscates her location, replacing it with an *anonymizing spatial region* (ASR) that encloses u. The ASR is then forwarded to the LS. Ignoring where exactly u is, the LS retrieves (and reports to the AZ) a *candidate set* (CS) that is guaranteed to contain the query results for any possible user location inside the ASR. The AZ receives the CS and reports to u the subset of candidates that corresponds to her original query. In order for the AZ to produce valid ASRs, the users send location updates whenever they move (through their secure connection). The described model is shown in Figure 1.

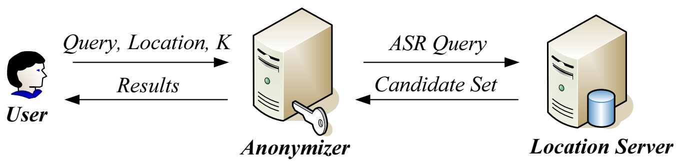


Fig. 1. System model

The ASR construction at the AZ (i.e., the anonymization process) abides by the user's privacy requirements. Particularly, specified an anonymity degree K by u, the ASR satisfies two properties: (i) it contains u and at least another $K-1$ users, and (ii) even if the LS knew the exact locations of all users in the system, it would not be able to infer with a probability higher than $1/K$ who among those included in the ASR is the querying one (i.e., u).

While the above ASR properties guarantee spatial K-anonymity to u, the model in Figure 1 also imposes requirements on CS computation. In particular, given an ASR, the LS must produce an *inclusive* and *minimal* CS. Inclusiveness demands that CS is a superset of u's query results; this property ensures that u receives accurate and complete answers. Minimality, on the other hand, requires that the CS contains the minimum number of data objects, without violating inclusiveness. Minimality ensures that CS transmission (from the LS to the AZ), and its filtering at the AZ do not incur unnecessary communication and processing overheads (and, thus, no further delay in reporting the results to u).

Existing methods construct truly anonymous ASRs and include ASR-based query processing mechanisms for the LS. However, they are targeted explicitly at the Euclidean space. The very concept of the ASR reflects this assumption, and the existing CS computation algorithms are tailored to the Euclidean distance. In practice, however, most LBS users are restricted to move in a road network in order to reach the queried spatial objects (hospitals, gas stations, shops, etc). Thus, they are typically interested in (and express their queries in terms of) the *network distance*, e.g., the traveling time through the roads of a city to the locations of interest. In this paper, we propose the first K-anonymity-based framework for location privacy in road networks, termed *network-based anonymization and processing* (NAP). Our contributions can be summarized as follows:

- We propose an edge ordering anonymization approach for users in road networks, which guarantees K-anonymity under the strict *reciprocity* requirement (described later).
- We identify the crucial concept of *border nodes*, an important indicator of the CS size and of the query processing cost at the LS.
- We consider various edge orderings, and qualitatively assess their query performance based on border nodes.
- We design efficient query processing mechanisms that exploit existing network database infrastructure, and guarantee CS inclusiveness and minimality. Furthermore, they apply to various network storage schemes.
- We devise batch execution techniques for anonymous queries that significantly reduce the overhead of the LS by computation sharing.

The rest of the paper is organized as follows: Section 2 overviews related work. Section 3 defines our system model and objectives. Section 4 elaborates on network-based anonymization (at the AZ), while Section 5 focuses on the processing of anonymous queries (at the LS). Section 6 empirically evaluates our methods on real road networks. Finally, Section 7 concludes the paper.

2 Related Work

Section 2.1 reviews related work on road network databases and Section 2.2 surveys the literature on spatial anonymity.

2.1 Query Processing in Road Networks

Basic Notation. In general, a road network can be modeled as a weighted graph $G = (N, E)$. N contains the network nodes, while E is the set of edges. Nodes n in N model road intersections, locations of road turns, or positions where traffic conditions change (e.g., a street gets narrower). On the other hand, every edge e connects two nodes and is associated with a non-negative weight $w(e)$. Weight $w(e)$ may represent, for instance, the traveling time from one node to the other. Figure 2 shows an example of a road network. Edge n_1n_2 has weight 3, and its endpoints are nodes n_1 and n_2. Let p be a point on an edge e with weight $w(e)$. The *partial weight* from p to an end-node of e is proportional to their (Euclidean) distance, while the sum of the two partial weights is equal to $w(e)$. For instance, object o_1 (shown as a solid point) lies on edge n_3n_4 and has partial weights 1 and 3 from nodes n_3 and n_4, respectively. Similarly, user u (the hollow point) falls on edge n_2n_3 and both of its partial weights are 2.

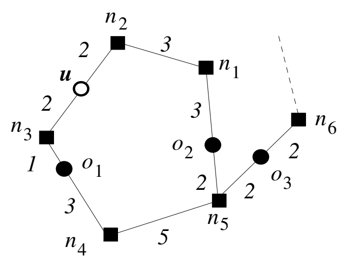


Fig. 2. Road network example

The *network distance* $d_N(u, o)$ between a user u and an object o is defined as the sum of edge weights along the shortest path (in the network) from u to o. In our example, the network distance $d_N(u, o_1)$ between user u and object o_1 equals to 2+1=3. Its derivation is strongly related to shortest path computation. In case of a small network, main memory shortest path algorithms (e.g., Dijkstra's algorithm [11]) can be applied to compute $d_N(u, o)$. Otherwise, disk-based data structures [31], [22] are utilized.

Query Processing by Network Expansion. Users are often interested in location-based queries such as r-range and kNN queries, in the context of a road network. Given a distance threshold r and a user location u, the r-range query returns all objects within (network) distance r from u. On the other hand, the kNN query retrieves the k objects that are closest to u. In the rest of the paper, the term distance refers to the network distance, and the r-range and kNN queries refer to their network versions (unless otherwise specified).

Papadias et al. [30] developed efficient indexing and processing methods for the above queries. They proposed the following disk-based structures for indexing the road network and the data objects: (i) the adjacency index packs adjacency lists of network nodes into disk blocks, (ii) the edge R-tree spatially indexes the network edges, and (iii) the object R-tree (ORT) organizes the locations of the data objects.

Network expansion [30] is a well-known technique for evaluating r-range and kNN queries. Starting from the user location u, it discovers objects on encountered edges while traversing the network like Dijkstra's algorithm, until the query results (i.e., data objects of interest) are found. Suppose that, in Figure 2, user u issues a range query with $r = 9$. First, we access the adjacency index to identify edges within the query range, following the steps in Table 1. A min-heap H is employed for organizing entries of the form $(n_i, d_N(u, n_i))$ (for encountered nodes n_i) in ascending order of distance $d_N(u, n_i)$. In our example, the edge n_2n_3 containing u is initially identified, and its end-nodes n_2 and n_3 (both with distance 2) are inserted into H. In each iteration, the node n_i with the minimum distance is de-heaped from H, its incident edges n_in_j are recorded, and its adjacent unvisited nodes n_j (having $d_N(u, n_j)$ within the range) are inserted into H. For example, at the first three steps of Table 1, edges n_2n_3, n_2n_1, and n_3n_4 fall completely within the query range. However, at step 4 the de-heaped node n_1 has distance 5 from u and only the partial edge $n_1n_5(4)$ lies within the range $r = 9$. The process continues until H becomes empty. Having discovered the relevant edges, we probe the ORT to retrieve the result objects, o_1 and o_2.

Step	De-heaped entry	H contents	Found edges
1	—	$(n_2, 2)$ $(n_3, 2)$	n_2n_3
2	$(n_2, 2)$	$(n_3, 2)$ $(n_1, 5)$	n_2n_1
3	$(n_3, 2)$	$(n_1, 5)$ $(n_4, 6)$	n_3n_4
4	$(n_1, 5)$	$(n_4, 6)$	$n_1n_5(4)$
5	$(n_4, 6)$	—	$n_4n_5(3)$

TABLE 1
Steps for range search, r=9

A kNN query can be evaluated by the above network expansion technique, subject to the following differences. In this case, there is no fixed distance threshold r. Thus, whenever we encounter an edge e, we need to search in the ORT to find the objects on e. During the process, a set W is used to keep track of the k objects found so far that are closest to u. Let γ be the largest distance in W. The search procedure terminates when the next de-heaped node has a distance (from u) larger than γ, since all remaining (non-encountered) objects are guaranteed to be further from u (than γ). To exemplify, we apply this method to answer the NN query (i.e., k=1) of u in Figure 2. The first two steps are the same as those shown in Table 1. After entry $(n_3, 2)$ is de-heaped, we discover edge n_3n_4 and access ORT to find the objects on it (i.e., o_1). Now, we insert o_1 into W and update γ to $d_N(u, o_1) = 2 + 1 = 3$. Observe that the adjacent node n_4 (of n_3) is not inserted into H because its distance is greater than γ. Next, entry $(n_1, 5)$ is de-heaped; its distance is greater than γ so it is ignored. Eventually, the heap H becomes empty and object o_1 is returned as the NN of u.

Alternative Query Evaluation Techniques. *Euclidean restriction* [30] is another approach for processing r-range and kNN queries. This technique, however, relies on the assumption that the weight of each edge is equal to its Euclidean distance. Based on this property, a set of candidate objects can be retrieved fast from the ORT. Then, the adjacency index is used to compute the network distances of the candidates from the query location u, in order to determine the actual result. Nevertheless, the assumption used in Euclidean restriction does not always hold in practice. For instance, depending on the application, the weight of an edge may be its toll fee or traveling time. In this paper, we adopt the network expansion technique due to its proliferation and its superior performance for r-range and kNN queries, as shown empirically in [30]. Finally, it is worth noticing that there exist pre-computation techniques that improve query performance [26], [19] at the cost of a preprocessing overhead and a higher update cost. Our approach may be utilized with these methods as well, since it relies only on primitive network-based search operations (see Section 5.3).

2.2 Anonymous Location-based Queries

Recently, considerable research interest has focused on preventing identity inference in location-based services. Studies in this area [17], [14], [28], [23] typically assume the model described in Section 1, proposing spatial cloaking (i.e., location obfuscation) techniques. In the following, we describe existing techniques for ASR computation (at the AZ) and query processing (at the LS). At the end, we cover alternative location privacy approaches and discuss why they are inappropriate to our problem setting.

Spatial Cloaking at the AZ. In general, the AZ applies the concept of K-anonymity [33] to hide the querying user location u. The idea is to compute an anonymizing spatial region (ASR), containing u and at least $K - 1$ other user locations. This offers privacy protection in the sense that the actual user position u cannot be distinguished from others in the ASR, even when a malicious LS is equipped/advanced enough to possess all user locations. This spatial K-anonymity model is most widely used in location privacy research/applications, even though alternative models are emerging.

Casper [28] is the first work on efficient and scalable AZ implementation for ASR computation. A quad-tree is utilized for indexing user locations and deriving ASRs. Suppose that the AZ needs to compute a 2-anonymous region (i.e., K=2) for querying user u_1 in Figure 3(a). The AZ first locates the leaf quad that contains u_1 and traverses the tree upwards until it identifies a region covering at least K users (including u_1). In this case, the AZ derives rectangle $R_{1,2,3}$ (containing three users) as the 2-anonymous region of u_1.

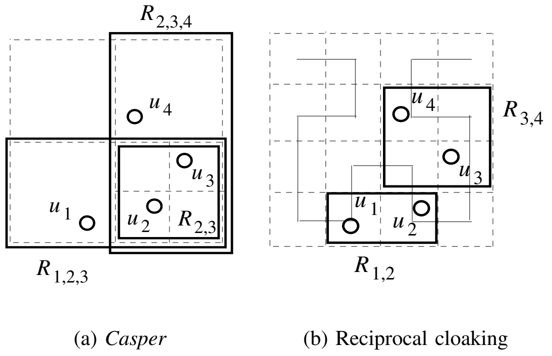


Fig. 3. Spatial K-anonymous cloaking, K=2

A closer look reveals that the ASR alone is not sufficient to guarantee privacy. The adversary may be able to infer u's location by correlating ASRs of different users. In Figure 3(a), u_2 and u_3 have the same 2-anonymous region $R_{2,3}$, whereas u_1 and u_4 have 2-anonymous regions $R_{1,2,3}$ and $R_{2,3,4}$, respectively. Observe that $R_{1,2,3}$ is the only ASR that contains u_1. If the adversary knows all user locations, it is possible to deduce that u_1 posed a query with ASR $R_{1,2,3}$. User u_4 encounters the same problem if $R_{2,3,4}$ is used.

To address this issue, *HilbASR* [23] requires every K-anonymous region to satisfy the *reciprocity property*, meaning that each ASR must be shared by at least K users. In the example of Figure 3(b), users u_1 and u_2 share the same 2-anonymous region $R_{1,2}$. This way, the adversary cannot infer who between them posed a query with ASR $R_{1,2}$. To enforce this constraint into the cloaking process, [23] proposed to organize user locations based on their Hilbert ordering [8], as shown in Figure 3(b). The ASRs can be computed fast in a dynamic manner (i.e., for on-demand K values). Specifically, consecutive users (in the Hilbert order) form disjoint buckets of K, and the ASR is computed as a bounding rectangle of the bucket that corresponds to the querying user u. For example, if $K = 2$, the ASR of u_1, u_2 is the depicted rectangle $R_{1,2}$, and that of u_3, u_4 is $R_{3,4}$.

We note that there exist distributed spatial cloaking methods that do not use an anonymizer [16], [10], but require that the users trust each other and collaboratively create ASRs. In general, it is easier/cheaper for an adversary to compromise (or even register as) a user of the system, than to take over a central (strongly secured) AZ. Furthermore, inter-client communication makes distributed cloaking particularly slow.

Query Processing at the LS. We continue by presenting ASR-based query evaluation at the LS. Upon receiving an ASR, the LS needs to compute a set of candidate result objects (the CS). To guarantee inclusiveness, the CS must contain the answer set of any possible query (i.e., user) location within the ASR, since the precise location of u is unknown to the LS. For the evaluation of an anonymous r-range query, the LS computes the Minkowski region of her ASR R_u, extending it by r (e.g., the gray region in Figure 4(a)). Then, the LS retrieves the objects in that region, i.e., o_1 and o_2.

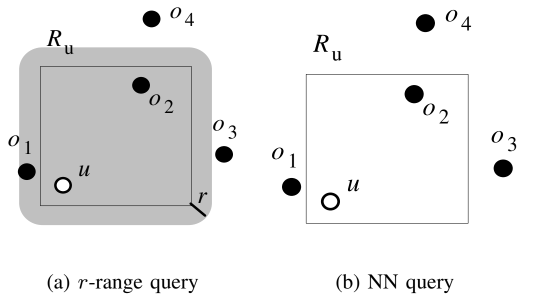


Fig. 4. Query processing at the LS (in Euclidean space)

The processing of an anonymous kNN query is more complex and relies on specific geometric properties in the Euclidean space [28], [23]. To compute the candidate set for the NN query (i.e., k=1) in Figure 4(b), the LS needs to retrieve (i) all objects located inside the ASR (i.e., o_2), and (ii) the kNN objects of any location along the ASR sides (i.e., o_1, o_2, o_3). Then, the LS eliminates duplicates (if any) and returns the candidate objects o_1, o_2, o_3 to the AZ.

We note that *Casper* [28] and *HilbASR* [23] are the only existing spatial anonymity methods that consider query processing at the LS. Their approaches are similar and follow the lines presented above; they explicitly target the Euclidean space and are inapplicable to road networks. For instance, the processing of an ASR-based kNN query (in both these works) requires the construction of perpendicular bisectors in order to derive the kNN objects along the ASR sides. However, perpendicular bisectors, as well as other Euclidean constructs involved, cannot be translated to the road network context.

Alternative Location Privacy Approaches. There exist location privacy approaches other than spatial anonymity. For instance, [20], [15] apply *private information retrieval* to process nearest neighbor queries. Based on cryptographic techniques, they guarantee that an adversary cannot infer the user's location u within polynomial time of a security parameter (e.g., key length). [20] is theoretical in nature, whereas [15] proposes a practical algorithm with $O(\sqrt{n})$ communication cost. However, query processing is particularly slow, and the method is inapplicable to range and kNN queries (with $k > 1$).

[24] proposes a location privacy method for NN processing. It uses a keyed function for mapping a 2D location onto a 1D value; the key is shared among the users and is unknown to the LS. The LS maintains the transformed (1D) data objects. A querying user u converts her location into its 1D value u' and forwards it to the LS. The latter returns the object that lies closest to u' (in 1D space). A drawback of this approach is that the reported NN is not always the actual one.

In [12], [25] the user u forwards to the LS a set of dummy locations in addition to hers, forming an *obfuscation set* of potential query points. If an adversary knows the real user locations, he may eliminate the dummies and identify u. In [35], the user sends only a fake location to the LS and incrementally retrieves its nearest neighbors. She stops when she is guaranteed to have obtained the NN of her actual location. The privacy region derived by [35] ignores other users' locations, and thus it cannot guarantee K-anonymity.

3 Problem Setting

We propose an anonymous query processing framework, targeted at road network databases. We adopt the trusted anonymizer model (i.e., the use of the AZ as a mediator between users and the LS), as illustrated in Figure 1. This choice is due to (i) the proliferation of this model in existing anonymous services (e.g., *Mailshell* [2] and *Spamgourmet* [3] provide anonymous use of Internet/email-based services without revealing the user's real email, the *Anonymizer* [1] provides anonymous web-surfing, etc), (ii) its suitability for time-critical applications [28], (iii) the existence of information security techniques and system architectures that support trusted third party services (e.g., [21]), and (iv) the availability of methods to ensure that a third party (i.e., the AZ) will honor the user privacy requirements [5], [4].

Our anonymization approach satisfies the reciprocity property (defined in Section 2.2). Specifically, it cloaks user locations using sets of line segments (i.e., road network edges, as we describe later), and ensures that each such set is shared among at least K users. Note that reciprocity guarantees K-anonymity even in the case where an adversary knows all user locations as well as the exact cloaking algorithm [17], [23], [16], [6].

An additional privacy requirement is that no user's position is revealed (regardless of whether she is the querying one or not). There are many potential attackers, whose knowledge and equipment may vary. It is possible that some of them ignore the user locations and, thus, it is essential not to transmit exact positions to the LS [17], [14], [28], [23]. Therefore, it is not an acceptable solution to simply append $K-1$ other user positions to that of u and collectively transmit them to the LS, even if this approach satisfies reciprocity.

We assume that the users $u \in U$ and the data objects $o \in O$ lie/move in a road network, as described in Section 2.1. We consider the generic network distance definition where the edge weights are non-negative and they do not have to be proportional to the Euclidean length of the corresponding road segment. For instance, they could be traveling times from endpoint to endpoint (i.e., depending on each road's congestion level, speed limit, number of lanes, etc). To simplify our discussion, we consider undirected (i.e., two-way) edges, but our techniques apply with trivial modifications to directed ones (e.g., one-way roads). We focus on static networks, with fixed edge weights. However, edge updates (e.g., weight/traveling time increase due to an accident) can be dealt with easily, provided that the AZ and the LS are informed of the network changes.

In this paper, we focus on two important location-based queries on road networks; the r-range and the kNN query described previously. For the latter type, we denote by $kNN\_dist(u)$ the network distance of the k-th NN. We consider *snapshot* queries (i.e., queries that are evaluated once and terminate), as opposed to queries that require constant update of their results (e.g., [9]).

The users log on to the system by first establishing a secure connection with the AZ. Through this connection, they update their locations to the AZ (whenever they move), they pose queries, and receive results. In addition to user locations, the AZ also stores the road network, as it needs to compute network distances to filter the candidate sets received by the LS. In particular, the edges are organized in a hash-table on edge ID, storing for each of them (i) its end-node IDs, (ii) its weight, and (iii) the IDs of edges incident to either end-node. The hash-table also stores the order and setting of the edge (introduced in Section 4). To map user/object coordinates to their containing edge, the edges are indexed by a PMR quadtree [18]. Each leaf quad stores the IDs of the edges that it (entirely or partially) covers. Given a user/object location, the tree is traversed down to the leaf quad that covers it, and the containing edge is identified among the edges stored in this leaf. Similar to previous spatial anonymity approaches, the AZ keeps all necessary information in main memory to cope with large numbers of users and high update rates.

Typically, the LS data are less dynamic and more voluminous than the users and are, thus, kept in secondary storage [28], [23]. We assume that the LS stores the data objects and the road network following the method of [30]. We choose this technique due to its proliferation in the road network literature. Note, however, that the theorems and general methodologies described below can be applied to different storage schemes as well as to memory-resident data. Specifically, the only requirement is that the employed network storage scheme provides the weights and connectivity of the edges and allows retrieval of the objects that fall on a given edge. Section 5.3 presents an example of NAP application to an alternative storage scheme.

In all our illustrations, we represent user locations as hollow points, data objects as solid points, and border nodes (to be defined later) as solid diamonds. To conclude this section, we list in Table 2 interpretations of primary symbols and acronyms used throughout the paper.

Term	Description
AZ	Anonymizer
LS	Location Server
AEL	Anonymizing Edge List
CS	Candidate Set
ORT	Object R-tree (at the LS)
K	Anonymity degree
$d_N(u,o)$	Network distance from user u to object o
$kNN\_dist(u)$	Distance from u to her k-th nearest object
$U/\lvert U\rvert$	The set/number of users registered with the AZ
ord_u	Order of user u
B	Total number of user buckets (for a given K)
b_i	The i-th bucket of users (for a given K)

TABLE 2
Interpretation of acronyms and symbols

4 Network-based Anonymization

In this section, we present the cloaking algorithm of our NAP framework. Our primary objective is to guarantee reciprocity-based anonymity. In NAP, the AZ anonymizes u with a set of line segments/edges instead of a spatial region (ASR).

The crux of our cloaking method is to utilize a global *edge ordering*; i.e., an ordered sequence that contains all network edges exactly once. The edge ordering is setting-sensitive, i.e., it specifies which end-node of the edge precedes the other. We refer to the position and setting of an edge in the ordering as the *edge order* and the *edge setting*, respectively. To avoid confusion, the setting of an edge depends solely on the ordered sequence, and has nothing to do with the direction (in the case of directed networks) of the road segment it models. Figure 5(a) shows a road network, and an ordering of its edges. The number next to each edge indicates its order and the arrow its setting.

The edge ordering defines an implicit linear order among the users themselves. In particular, a user u precedes another u' if the edge of u has smaller order than that of u'. If they fall on the same edge n_in_j (with setting from n_i to n_j), u precedes u' if it is closer to n_i. Ties among coinciding users are resolved arbitrarily. This precedence relationship defines the order ord_u of each user u. The position of a user in the defined sequence is referred to as the *user order*. The example in Figure 5(a) contains 10 users whose subscript indicates their order (i.e., user u_3 has order 3, etc).

Reciprocity in NAP is achieved by conceptually partitioning the user ordering into buckets of K users each, and forwarding to the LS the edges corresponding to the bucket of the querying user u. This set of edges is called the *Anonymizing Edge*

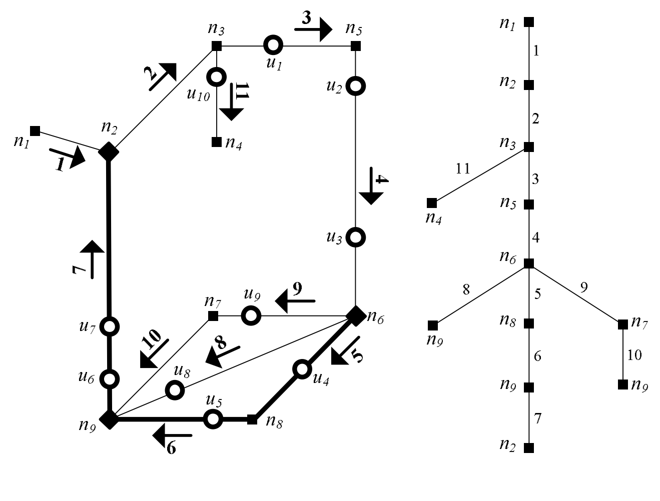


Fig. 5. Edge ordering

List (AEL) of u. Specifically, let U be the set of users registered with the AZ and assume that a querying user requires anonymity of degree K. Set U is partitioned into $B = \lfloor |U|/K \rfloor$ buckets, each containing K users, except the last one which may contain up to $2 \cdot K - 1$; the i-th bucket b_i (for $i < B$) consists of users with order from $(i-1) \cdot K + 1$ to $i \cdot K$, and the B-th is assigned the remaining ones.

Consider the network in Figure 5(a), where $|U| = 10$, and assume that u_6 poses a query with anonymity requirement $K = 3$. This results into 3 buckets, $b_1 = \{u_1, u_2, u_3\}$, $b_2 = \{u_4, u_5, u_6\}$, and $b_3 = \{u_7, u_8, u_9, u_{10}\}$. User u_6 belongs to b_2 and is anonymized together with the other users in it. The boundary users (i.e., first and last) of b_2 are u_4 and u_6, whose edges have orders 5 and 7. The AEL is formed by collecting all edges with orders between 5 and 7; i.e., it comprises edges n_6n_8, n_8n_9, and n_9n_2, shown bold.

We follow the above global edge/user ordering approach, because a local one could lead to privacy breach in a way similar to Figure 3(a) for *Casper*. Specifically, upon interception of an AEL generated by our method, one cannot infer who among the users in the corresponding bucket was the querying one. In other words, the AEL for any user in the same bucket is identical, and therefore an adversary cannot pinpoint the query originator with a probability higher than $1/K$ (recall that each bucket contains K or more users). Hence, our cloaking method satisfies reciprocity. Reciprocity, in turn, is a sufficient condition for anonymity [23] and, thus, *NAP guarantees K-anonymity to the querying users*.

In the rest of this section, we describe edge ordering strategies (Section 4.1). Then, we present particular techniques for the anonymization procedure (Section 4.2). Finally, we analyze the properties of the proposed edge orderings (Section 4.3).

4.1 Border Nodes and Edge Ordering

To ensure reciprocity we can use any global edge ordering; e.g., we could use a random permutation of network edges. However, for our method to be practical, we need to additionally take into account its performance. Consider again the model of Figure 1. Observe that the overall result latency experienced by the users depends on (i) the processing time spent for anonymization (at the AZ), CS computation (at the LS), CS filtering (at the AZ), and (ii) the communication cost spent for AEL forwarding (from the AZ to the LS), and CS transmission (from the LS to the AZ). We select an edge ordering taking the above into account, and based on the concept of *border nodes*.

Definition 1: **Border nodes:** We call border nodes of a given AEL those of its edges' end-nodes that are incident to some edge outside the AEL. That is, $BN(AEL) = \{n_i | (\exists n_i n_j \in E \text{ s.t. } n_i n_j \in AEL) \bigwedge (\exists n_i n_k \in E \text{ s.t. } n_i n_k \notin AEL)\}$, where E is the set of edges in the network.

To exemplify, consider Figure 5(a) where the AEL of u_6 (for $K = 3$) comprises the 3 edges shown bold. The end-nodes of AEL edges are n_6, n_8, n_9, n_2. Among them, n_6, n_9, and n_2 are border nodes (thus shown as solid diamonds), since each one is incident to some edge outside the AEL. On the other hand, end-node n_8 is incident to edges n_6n_8 and n_8n_9, both of which are inside the AEL. Hence, n_8 is not a border node.

Border nodes have an important property. It follows from Definition 1 that *any shortest path from a point inside the AEL to some object outside the AEL passes through some border node*. Based on this property, Theorem 1 below constitutes the foundation of CS computation in NAP.

Theorem 1: Assume that the LS receives an AEL for a kNN (r-range) query with parameter k (r, respectively). An inclusive *and* minimal *CS is formed as the union of*

- *the objects that fall on the AEL edges*
- *the kNN objects of all border nodes (the objects within distance r from any border node, respectively).*

Proof: We focus on the kNN case, as the proof for r-range queries follows the same lines. We prove inclusiveness by contradiction. Suppose that there is some object o among the k NNs of u that is not in the CS. The CS includes all objects on AEL edges, so o must fall outside the AEL. User u lies inside the AEL and, thus, the shortest path from u to o passes through some border node n; i.e., it holds that $d_N(u, o) = d_N(u, n) + d_N(n, o)$ (**A**). The CS includes the k NNs of n, so (i) o must lie further from n than its k-th NN, i.e., $d_N(n, o) > kNN\_dist(n)$ (**B**), and (ii) there are at least k objects o' (where $o' \neq o$) within distance $d_N(u, n) + kNN\_dist(n)$ from u in the CS, i.e., $kNN\_dist(u) \leq d_N(u, n) + kNN\_dist(n)$ (**C**). By adding factor $d_N(u, n)$ to both sides of inequality (**B**), we derive $d_N(u, n) + d_N(n, o) > d_N(u, n) + kNN\_dist(n)$, which according to (**A**) and (**C**) (applied to the left and right side of the inequality, respectively) leads to $d_N(u, o) > kNN\_dist(u)$. The latter contradicts the original hypothesis that o is one of the k NNs of u. Thus, CS is inclusive.

The querying user u could coincide with (and thus have zero distance from) any object that falls on an AEL edge. Thus, all such objects should be in the CS. On the other hand, the k NNs of the border nodes should also be in the CS, since u

may be located at any one of them. Hence, the derived CS contains no unnecessary objects, i.e., it is minimal. □

From Theorem 1 it follows that the CS size and the (AEL-based) processing cost at the LS increase with the number of border nodes in the AEL. To compute an optimal ordering (i.e., an ordering that achieves the minimum number of border nodes for any user) is impossible, since the same ordering must serve different user-specific anonymity degrees K. Therefore, we use heuristic-based algorithms.[1]

Random orderings. We describe two naïve orderings, RE and RN, used as baseline approaches. RE (*random edge ordering*) is a random permutation of the network edges; their settings are also decided at random. On the other hand, RN (*random node ordering*) first forms a random permutation of network nodes. Then, it scans the permutation and for each considered node n_i, it includes its incident edges $n_i n_j$ (with setting from n_i to n_j) into the produced ordering; previously encountered edges are ignored to avoid duplicates.

Traversal-based orderings. This family of orderings adapts the standard breadth-first and depth-first traversal algorithms (resulting in orderings termed BF and DF, respectively). The intuition behind traversal-based algorithms is that connected edges are likely to exhibit locality, and share common end-nodes (thus reducing the number of AEL border nodes). The traditional depth-first/breadth-first algorithms visit every node once (by marking them as visited). We adapt them so that they visit every edge once (by marking edges as visited instead), and allowing passing through a node multiple times. The settings are set according to the direction of the traversal. The ordering in Figure 5(a) is produced by DF, whose traversal follows (top-down) the search tree shown in Figure 5(b); the number next to each edge represents its visiting (and, thus, assigned) order.

Hilbert-based orderings. The rationale behind this approach is that nodes/edges close in Euclidean space are likely to be close in terms of network distance. In turn, this means that consecutive edges in the ordering are likely to share end-nodes. HE (*Hilbert edge ordering*) and HN (*Hilbert node ordering*), work in the same way as RE and RN, respectively, the difference being that edges and nodes are considered in Hilbert order (in HE, the Hilbert values of the edge midpoints are used as their sorting keys). We establish the convention that the settings are from left to right and, in case of vertical edges, from down to up. Intuitively, HE/HN are adaptations of the *HilbASR* (Euclidean) cloaking to road networks[2]. Hilbert-based orderings (as opposed to the previous two categories) are inapplicable to environments where the Cartesian coordinates of nodes/users are unknown or undefined (see Section 5.3).

Edge ordering is performed once when the AZ is set up for the first time. Upon termination, the computed orders/settings are stored in the edge hash-table described in Section 3.

4.2 Anonymization Procedure

Given an edge ordering, the next question is how AEL computation can be implemented efficiently at the AZ. Parameter K is not known in advance and varies, since different users have different anonymity requirements, and even queries by the same user may specify different K, depending on the nature of the queried data. As buckets are defined according to K, they cannot be explicitly materialized. Instead, the AZ employs an index that keeps the users sorted on their order and allows efficient AEL computation for arbitrary K. The index is an aggregate B-tree (similar to an aggregate R-tree [29]), whose internal nodes keep for each child the number of users in the corresponding sub-tree. Figure 6 shows this tree in the example of Figure 5(a). For each user (e.g., u_6) we store the ID of the edge it falls on ($n_9 n_2$), the edge's order (7), and its distance from the edge's first end-node ($|n_9 u_6|$). The latter two values are used (primarily the edge order and secondarily the distance from the first end-node) as the sorting key of the tree. In Figure 6 the numbers in the shaded boxes correspond to the aggregate information maintained, i.e., the cardinalities of the sub-trees rooted thereof. Note that we use a B-tree instead of a B$^+$-tree (i.e., user information is also stored in internal nodes), because it is faster for in-memory indexing [27].

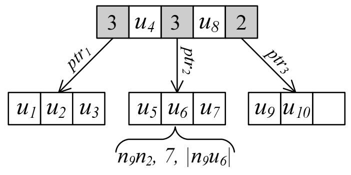


Fig. 6. Aggregate B-tree

When a user u poses a query requiring K-anonymity, the first task of the AZ is to acquire the order ord_u of u. To achieve this, it initially identifies the edge u falls on (using the edge quadtree) and the order of this edge. Next, it computes ord_u by traversing the aggregate B-tree according to the edge order and the distance of u from its first end-node, down to the leaf that contains u; ord_u is equal to the sum of all sub-tree cardinalities and users encountered on the left of u. In Figure 6, if the querying user is u_6, we first visit the B-tree root and infer that there are 3 users with smaller order under pointer ptr_1 plus user u_4. Then, we visit u_6's leaf node, which adds u_5 in the set of users preceding u_6. In total, there are 5 users before u_6 and, thus, its order is 6.

The next task of the AZ is to retrieve the users falling in u's bucket. Having computed ord_u, its bucket is the $\min(B, \lceil ord_u/K \rceil)$-th. The remaining users that correspond to this bucket are retrieved by moving to the left and to the right of u in the B-tree. In our example, where u_6 requires 3-anonymity, its bucket additionally includes users with orders 4 and 5, i.e., u_4 and u_5. Finally, the AZ forms the AEL by collecting all edges between the boundary users of b_2, as explained previously.

To conclude the discussion about the AZ, we need to clarify the issue of user movement. When the users move, they send

1. To provide an intuition, even if the optimization criterion were simplified to computing an Eulerian path [13], [34] (i.e., a continuous path that traverses every edge exactly once), finding such a path (if any) is an NP-complete problem [32].

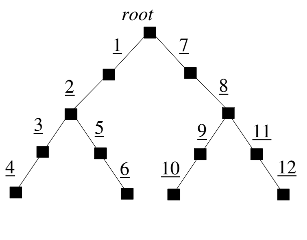
2. We experimented on different traversals, based on more complicated (and expensive) heuristics, but none performed better than the best of the above, as also discussed in Section 6. We also tried different space-filling curves, but Hilbert-based orderings were superior.

an update to the AZ reporting their old and new position. These updates are efficiently handled by deleting the old user location from the aggregate B-tree and then inserting the new one. Prior to the deletion (insertion), the AZ probes the edge quadtree with the old (new) position to find the user's old (new) edge and offset from its first end-node. While traversing the B-tree to perform the deletion (insertion), we update its aggregate information by decrementing (incrementing) by one the cardinality of each sub-tree visited. If updates and queries arrive at the same time, updates are processed first so that anonymization is correct. A final remark is that if the AZ information is stored in secondary storage, the adaptation of the above cloaking technique is easy, using a disk-based aggregate B$^+$-tree to index the users.

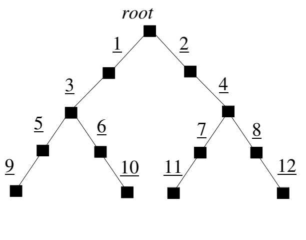
4.3 Analysis of Edge Orderings

The number of border nodes is an important indicator of the CS size and of the LS processing cost. To provide an insight on the behavior of our proposed edge orderings, we analyze their numbers of border nodes in a simple road network. A typical road network branches from the city center (e.g., the root) and exhibits a self-similar structure. We decompose the network into *junctions* and *road sections*. The junctions are nodes of degree higher than 2. The sections are paths (i.e., sets of connected edges) between two consecutive junctions that pass strictly through non-junction nodes. We treat the decomposed network as a tree with parameters: the fanout f, and the average number of edges per road section l. Figure 7 depicts a road network with $f = 2$ and $l = 2$.

An edge is said to be at level m if its path to the root passes through $m-1$ junctions; there are $l \cdot f^m$ edges at level m. To simplify our analysis, we assume that there is exactly one user per edge (the general case is obtained by scaling K accordingly). Thus, the AEL of a query contains K edges for any edge ordering.



(a) DF ordering



(b) BF ordering

Fig. 7. Typical road network topology, $f = 2$, $l = 2$

For RE, the probability that an edge is included in the AEL is $K/|E|$, where $|E|$ is the number of edges in the network. A $(2/l)$ fraction of the edges have a junction as an end-node (incident to another f edges), while the remaining have strictly non-junction end-nodes (i.e., each end-node is adjacent to one other edge). It follows that the number of border nodes for RE is:

$$\Phi_{RE} = K \cdot \left(\frac{2}{l}(1 - (\frac{K}{|E|})^f) + \frac{2l-2}{l}(1 - \frac{K}{|E|}) \right) \quad (1)$$

This number converges to $2 \cdot K$ when $K \ll |E|$.

An example of the depth-first ordering (DF) is shown in Figure 7(a). Assuming that the traversal begins from the root node, the label of each edge indicates its order. The AEL covers at most $\lceil K/l \rceil + 1$ connected road sections. Thus, the number of border nodes for DF is:

$$\Phi_{DF} = \lceil K/l \rceil + 1 \quad (2)$$

Clearly, DF leads to much fewer border nodes than RE.

The breadth-first ordering (BF) is illustrated in Figure 7(b). Our analysis involves investigation of different cases, depending on: (i) the level of the user in the tree, and (ii) the value of K. Suppose that the user is located on an edge at level m. When $K \leq f^m$, the AEL edges are distributed across K different road sections, i.e., those edges are incident to some edge outside the AEL. In case K is between f^m and $l \cdot f^m$, the AEL covers f^m road sections and there are 2 border nodes per road section. If K is larger than $l \cdot f^m$, then the number is defined recursively by considering also the road sections in the next level $m+1$. In summary, the number of border nodes for BF is given by:

$$\Phi_{BF}(K,m) = \begin{cases} 2K & \text{if } K \leq f^m \\ 2f^m & \text{if } K \in (f^m, l \cdot f^m] \\ 2f^m + \Phi_{BF}(K - l \cdot f^m, m+1) & \text{if } K > l \cdot f^m \end{cases} \quad (3)$$

Note that BF leads to fewer border nodes than RE, but more than DF.

The analysis of Hilbert-based orderings is non-trivial because they heavily rely on the actual coordinates of the nodes in the map. We conjecture that the Hilbert curve roughly preserves the connectivity of the road network and thus its number of border nodes is close to that of DF.

5 Anonymous Query Processing

In this section we describe AEL query processing at the LS; in Section 5.1 we present algorithms for minimal and inclusive CS computation for a single query, while in Section 5.2 we propose additional optimizations for the case where multiple AEL queries are processed in a batch. In Section 5.3 we demonstrate the generality of NAP with respect to the network storage scheme used at the LS.

5.1 Single Query Processing

Processing is based on Theorem 1. A direct implementation of the theorem uses (network-based) search operations as off-the-shelf building blocks. Thus, the NAP query evaluation methodology is readily deployable on existing systems, and can be easily adapted to different network storage schemes, as we discuss in Section 5.3. As a case study, in this section we focus on the storage scheme and the network expansion framework of [30], in order to provide a concrete NAP prototype.

Consider first the scenario where the AZ sends a single AEL query to the LS. CS computation follows Algorithm 1. Step 1 computes the border nodes of the AEL (using the edge R-tree and the adjacency index). Step 2 queries the ORT and places into the CS all objects falling on some AEL edge. Then, steps 3 and 4 expand all border nodes to include in the CS their kNN objects (or, for r-range query type, the objects within distance r from them). Depending on the query type, some optimizations are possible to reduce the LS processing cost.

Algorithm 1 Candidate Set Computation in NAP

```
Search(AEL $L$)
1. Identify the border nodes of $L$
2. Query the ORT and collect all objects falling on $L$ edges
3. For every border node $n$
4.     Perform range (or NN) search with parameter $r$ ($k$) at $n$
5. Form CS as the union of objects retrieved in steps 2 and 4
```

Range query optimizations. If the query type is r-range, border node expansion (in step 4) does not need to proceed to AEL edges, because the corresponding objects are either on some AEL edge (and, thus, retrieved by step 2) or, if they are outside the AEL, they are discovered by the expansion of another border node. An additional optimization particular to the range query type is to combine steps 2 and 4 so that CS objects are collected by querying the ORT only once. Specifically, after step 1, we expand the network (using only the adjacency index) for every border node up to distance r, and then query the ORT to retrieve the objects that fall in some of the acquired edges or inside the AEL. An additional optimization is that when the expansion of a border node n visits (i.e., de-heaps) a previously expanded node n', then expansion needs not proceed to (i.e., en-heap) the adjacent nodes of n', since the objects reachable through n' are inserted into the CS by n''s expansion.

kNN query optimizations. If the query type is kNN, in step 4 the LS retrieves the k NNs of each border node using network expansion in all directions, i.e., it also proceeds on the AEL edges. The reason is that, even if some NNs fall inside the AEL or belong to the k NN set of other border nodes, they lead to earlier termination of the expansion. However, kNN processing allows for an optimization on ORT accesses; if a border node expansion needs to process objects that fall inside the AEL or lie on an edge encountered in a previous expansion, we need not query the ORT, but may directly use the data objects already fetched into the memory-resident CS. Another optimization is to reuse kNN results of previously expanded border nodes; if during the expansion we de-heap some of these nodes, we directly insert their kNN objects into the temporary result (denoted by W in Section 2.1) and do not en-heap their adjacent nodes.

An important point in CS formation concerns the acceleration of AZ filtering. When the AZ receives the CS, it maps all objects onto network edges (using the edge quadtree) and performs expansion (using the edge hash-table) around u to retrieve the actual result. To minimize the cost of mapping, the LS organizes the CS by grouping together objects that fall on the same edge, and delimiting the groups with a special character. This way, upon receipt of the CS, the AZ identifies the containing edge of a group by probing the edge quadtree only once (for one of the objects)[3].

5.2 Batch Query Processing

In general, the LS processes queries in discrete timestamps, and multiple AEL-based queries may be arriving in the same timestamp. In this case, the queries are evaluated in a batch. Below we propose strategies aiming at maximizing computation sharing among different queries.

Sharing among expansions from a common border node. The nature of NAP enables computation sharing for common border nodes. In particular, the border nodes are not at arbitrary positions in the network; instead, they are network nodes and they are likely to be border nodes of multiple AELs. Let n be such a node and let Q_r and Q_{NN} be the sets of range and kNN queries, respectively, that have n as a border node. We share computations by performing a single expansion at n, based on the following two rules:

Rule 1. Among queries in Q_r (Q_{NN}) only the one with the largest parameter r_{max} (k_{max}, respectively) needs to be processed, since its result is by definition a superset of that for any other query in Q_r (Q_{NN}).

Rule 2. The r_{max} range query is evaluated before the k_{max}NN one (recall that range search is simpler/faster than kNN). Let S_r be the set of objects retrieved by the r_{max} range search and $|S_r|$ be their number. If $|S_r| \geq k_{max}$, then no further processing is necessary for Q_{NN}, as S_r is a superset of any kNN result for Q_{NN}. Otherwise (i.e., $|S_r| < k_{max}$), we need to additionally retrieve $k_{max} - |S_r|$ more NNs; the expansion continues from the point where the r_{max} range search stopped, starting with S_r as the set of NNs found so far and reusing the search heap of the range search.

Sharing among different border node expansions. Computation sharing is possible not only among queries with common border nodes, but also among different border node expansions. Stated this fact, before proceeding with the corresponding techniques, we point out that the range queries posed at the LS are processed before kNN ones, due to the relative complexity of the latter; i.e., we prefer having the kNN queries benefiting from available range results rather than the other way around. Different expansions share computations as follows.

When the processing of an r_{max} range query from border node n reaches another node n' for which r'_{max} range expansion has been previously performed, then if $r_{max} \leq r'_{max}$ the expansion of n needs not en-heap n''s adjacent nodes. The reasoning is similar to the corresponding optimization discussed at the end of Section 5.1. To maximize the effectiveness of this enhancement, we expand border nodes of range queries in descending r_{max} order. Computation sharing is possible in the case of k_{max}NN expansions too, exploiting formerly retrieved kNN and range results. Specifically, when a k_{max}NN expansion from a border node n reaches (i.e., de-heaps) another border node n' for which a result is computed

3. In general, the LS cannot directly include the edge ID into the CS, since the AZ does not necessarily use the same ID to refer to the same edge.

with $|S'_r| \geq k_{max}$ or $k'_{max} \geq k_{max}$ (where $|S'_r|$ and k'_{max} refer to n''s results), we insert objects retrieved for n' into the the temporary result W of n, and do not proceed to (i.e., en-heap) n''s adjacent nodes. To fully exploit this enhancement, we process different NN expansions in descending k_{max} order.

Algorithm 2 integrates the above and presents batch query processing at the LS. Given a set Q of AEL-based queries, step 1 extracts the border nodes for each of them. Step 2 expands the network (using only the adjacency index at this stage) for all range queries, applying the aforementioned order among them and following Rule 1. Step 3 inserts into the CS all objects that satisfy some range query or fall on some AEL edge; this is performed in a single probe of the ORT, thus avoiding multiple traversals. Steps 4 and 5 process the border nodes of kNN queries (i.e., nodes with non-empty Q_{NN}) in the order described above and following Rule 2. Note that in step 5 we establish the convention that $|S_r| = 0$ if no range query has been performed for n in steps 2, 3 (i.e., if $Q_r = \emptyset$). A point worth stressing is that Algorithm 2 produces a unified CS for all queries. It can be easily seen that this CS is the union of the individual candidate sets for every query in the batch, and is thus inclusive. It is also minimal, in the sense that it does not contain duplicate objects. To assist AZ filtering, the CS is organized in the way described in Section 5.1.

Algorithm 2 CS Computation for Multiple Queries in NAP

```
Search(Set of AEL-based queries $Q$)
1. Identify the border nodes of every AEL in $Q$
2. Expand each border node to $r_{max}$ range
3. Find objects on AEL edges or in step 2 ranges (using ORT)
4. For every border node $n$ with non-empty $Q_{NN}$
5.     If $|S_r| < k_{max}$, perform $k_{max}$NN search at $n$
6. Form CS as the union of objects retrieved in steps 3 and 5
```

5.3 Alternative Storage Schemes

NAP processing can be applied to any network storage scheme, as long as it provides the weights and the connectivity of the edges, and allows retrieval of the objects that fall on a given edge. To exemplify this, we use as a case study an alternative, widely used storage scheme [36], illustrated in Figure 8. Given a node id (e.g., n_i), the *adjacency tree* links to the location of a flat file (the *adjacency file*) that contains the node's adjacency list. The first entry of this list implies that n_i is adjacent to n_j and provides the weight of edge n_in_j; it also maintains a pointer into the *object file* that stores the data objects lying on n_in_j. For each object (e.g., p_m), the object file includes its partial weight from the first end-node of the edge (i.e., $w(n_ip_m)$).

Let us apply Algorithm 1 to the above storage scheme. In step 1, for each end-node n of the AEL edges we query the adjacency tree and determine whether n is a border node by examining its adjacency list (from the adjacency file). By definition, the adjacency list of n includes an entry for the corresponding AEL edge; we follow this entry's pointer to retrieve the objects falling on the edge (step 2). Regarding steps 3 and 4, network expansion can be easily applied to this

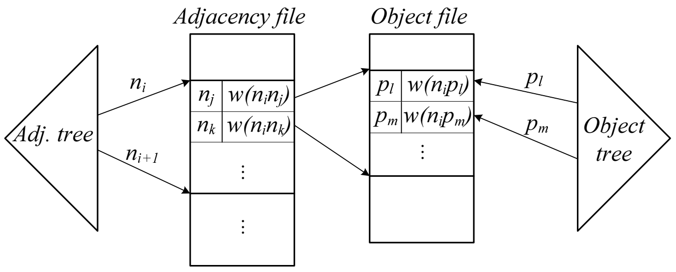


Fig. 8. Network storage scheme of [36]

scheme [36]. The adaptation of Algorithm 2 follows the same lines.

Note that NAP does not require the coordinates of nodes, users and objects (e.g., the above scheme does not include them). This property is essential, since Cartesian coordinates may be unknown or undefined (i.e., when mapping to a coordinate system is not meaningful). For instance, in underground spatial networks (e.g., metro systems) where GPS fails, RFID technology is used instead. Objects/users (e.g., trains) are tracked using the id of the closest RFID sensor, which may be dealt with as a network node (without requiring its coordinates). Anonymization in this case is similar to Section 4, but Hilbert-based orderings are inapplicable due to the lack of Cartesian coordinates.

6 Experimental Evaluation

In this section, we evaluate the robustness and scalability of our proposed methods on a real road network. Our algorithms were implemented in C++ and experiments were executed on a Pentium D 2.8GHz PC. We measured the average of the following performance values over a query workload of 100 queries: (i) anonymization time and refinement time at the anonymizer AZ, (ii) I/O time and CPU time for query processing at the location server LS, and (iii) the communication cost (in terms of transmitted points) for the anonymizing edge list AEL and the candidate set CS. Note that each edge in AEL is counted as two points.

6.1 Experiment Setup

We use the real road network of San Francisco, obtained from [7]. By default, our experiments use a subnetwork with 50000 edges. Weights of the edges are set to their lengths. We generate $|U|$ users and $|O|$ objects. The locations of users and objects follow either uniform distribution (by default) or Gaussian distribution[4]. At the LS, the disk page is 4KBytes and each index structure (edge R-tree, ORT, etc) is associated with a memory buffer with capacity set to 5% of its disk size. Table 3 summarizes the investigated parameters and their examined values, with the defaults shown bold. Parameter r is expressed as a multiple of the average edge weight.

4. The mean of the Gaussian distribution is at the network center and its standard deviation is 10% of the maximum network distance.

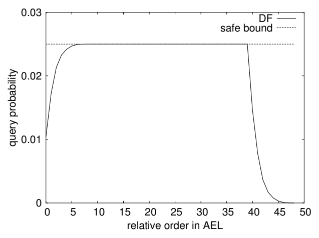
Parameter	Values
Number of users $\|U\|$ ($\times$ 1000)	10, 50, **100**, 150, 200
Number of objects $\|O\|$ ($\times$ 1000)	64, 128, **256**, 512, 1024
User distribution	**Uniform**, Gaussian
Object distribution	**Uniform**, Gaussian
Anonymity degree K	10, 20, **40**, 80, 160
NN query parameter k	1, 5, **10**, 15, 20
Range query parameter r	1, 1.5, **2**, 2.5, 3

TABLE 3
Summary of parameters and their values

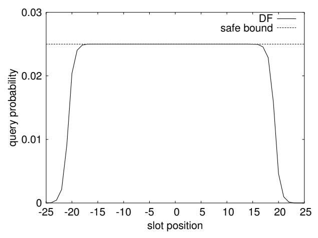
6.2 Robustness Experiments

In this section, we illustrate the achieved anonymity and study the performance of our methods for different orderings, location privacy models, and user/object distributions.

Anonymity strength. NAP is theoretically guaranteed to honor reciprocity and provide K-anonymity. Figure 9 empirically demonstrates this fact, i.e., that no user in the AEL is more than $1/K$ likely to have issued the query. We generate 1000 random queries with $K = 40$ and record the position of the querying user within the AEL according to ord_u (we include results only for DF as those for other orderings are almost identical). Figure 9(a) plots the querying frequency per user position in the AEL. The dashed line, labeled "safe bound", corresponds to probability $1/K = 0.025$. There are more than $K = 40$ positions (up to 48) because the AEL may contain over K users. Figure 9(b) provides another viewpoint, considering the median AEL position as slot 0. No frequency in Figure 9 exceeds the safe bound, i.e., an adversary with knowledge of all user locations and of the anonymization algorithm cannot pinpoint the querying u with a probability higher than $1/K$.



(a) w.r.t. first position



(b) w.r.t. median position

Fig. 9. Querying frequency across AEL positions

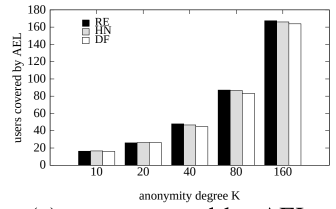
Effect of the proposed orderings. Table 4 shows the cost of the orderings presented in Section 4.1 for kNN queries, by fixing the parameters to their default values. Observe that the AEL size accounts for only a small fraction of the communication cost and it is insensitive to the particular ordering used. In RE, the AEL contains edges that are widely distributed in the network, leading to high processing cost (at LS) and communication cost due to the large CS size. RN places close to each other edges that share a common end-node and, thus, its cost is much lower than RE. Between the traversal-based orderings, DF outperforms BF because edges along a path are arranged next to each other, so the AEL contains many connected edges. The Hilbert-based orderings (HE, HN) perform similarly and are slightly worse than DF. The results for range queries are similar to those in Table 4. In subsequent experiments, we only compare the representative orderings: RE, DF, and HN.

	LS processing cost (sec)		*Comm. cost (points)*	
Ordering	**I/O time**	**CPU time**	**AEL**	**CS**
RE	0.8563	0.3092	44.75	390.01
RN	0.5236	0.2093	44.28	287.36
BF	0.3850	0.2078	45.72	292.94
DF	0.0989	0.0798	44.12	156.68
HE	0.0975	0.0959	44.93	177.11
HN	0.0925	0.0921	44.19	169.55

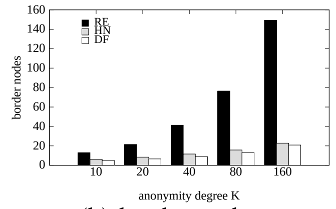
TABLE 4
Cost of kNN queries, default parameter values

Statistics of edge orderings. To understand the characteristics of NAP anonymization, we measure two statistics that are indicative of the query processing performance (at LS): (i) the actual number of users covered by AEL, and (ii) the number of border nodes in AEL.

Figure 10(a) shows the total number of users covered by AEL, as a function of the anonymity degree K. This number must be at least K by definition, but we wish to keep it as small as possible to enhance performance. We observe that the number of covered users stays close to K for all orderings, confirming that our edge-based anonymization approach does not produce unnecessarily large AELs.



(a) users covered by AEL

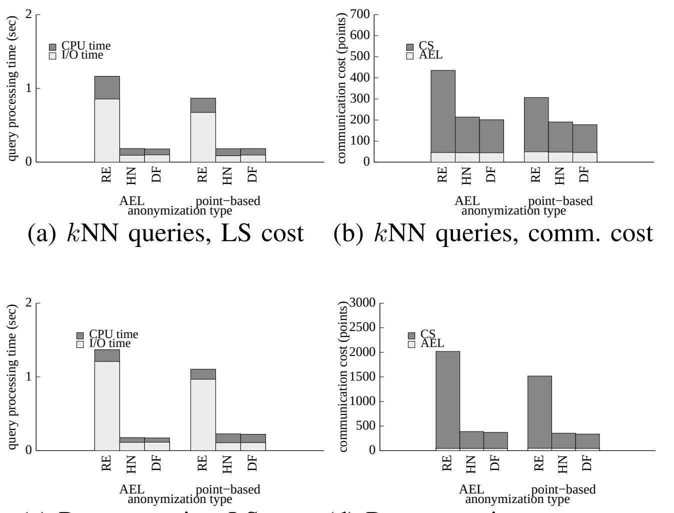


(b) border nodes

Fig. 10. AEL statistics vs. anonymity degree K

Figure 10(b) plots the number of border nodes in AEL as a function of the anonymity degree K. In RE, the AEL edges are far apart, leading to a large number of border nodes. Among our proposed edge orderings, DF is the best. The results in Figure 10(b) verify the qualitative analysis in Section 4.3. As explained in Section 4.1, the number of border nodes is an important indicator of CS size and query processing cost. This fact (that is quintessential to NAP) is verified later by the experiments in Figures 13 and 14.

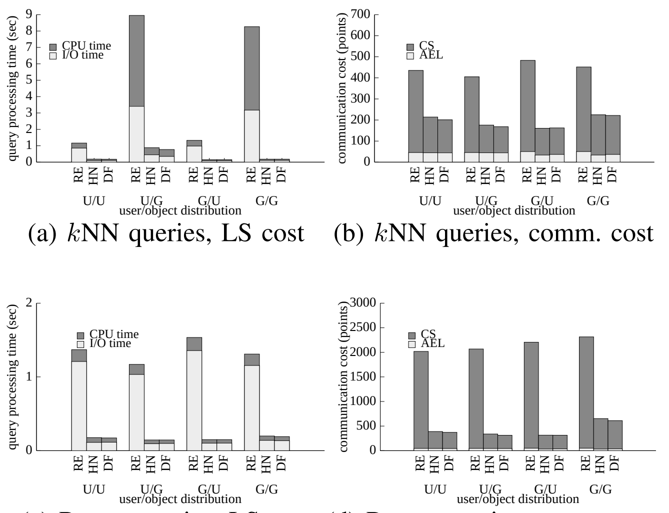
Anonymization type. In the next experiment, we examine whether the prevention of exact user locations in NAP (see Section 3) comes at the cost of poorer performance. In particular, we consider two types of anonymization: (i) edge-based anonymization (i.e., as in standard NAP), and (ii) point-based anonymization (where the AZ sends directly the exact user locations covered by the AEL). Figure 11 compares the cost between these two anonymization strategies, with respect to the anonymity degree K. In point-based anonymization, the CS is by definition a subset of the AEL-based CS.



(a) kNN queries, LS cost (b) kNN queries, comm. cost

(c) Range queries, LS cost (d) Range queries, comm. cost

Fig. 11. Effect of anonymization type



(a) kNN queries, LS cost (b) kNN queries, comm. cost

(c) Range queries, LS cost (d) Range queries, comm. cost

Fig. 12. Effect of user/object distribution

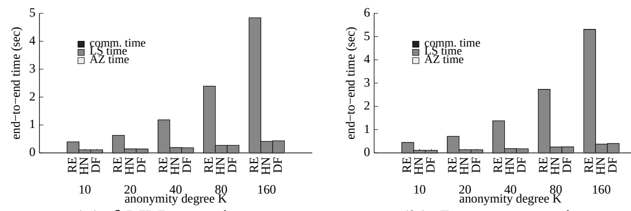
Nevertheless, for the best orderings (HN and DF), there is no observable difference between these two anonymization types. The reason is the strong AEL locality, which diminishes the additional costs of sending entire edges instead of points. This experiment verifies that the NAP guarantee of not leaking exact user locations, comes without performance sacrifices.

Effect of user/object distribution. Figure 12 illustrates the cost of queries with respect to the distribution of users and objects on the road network. The terms U and G denote uniform and Gaussian distributions, respectively; e.g., the label U/G represents the case where users are uniformly distributed and objects follow Gaussian distribution. HN and DF have consistently good performance across different user/object distributions. Under Gaussian object distribution, some AEL edges of RE fall in areas with low object density. RE becomes prohibitively expensive in such cases as its expansions need to search in large parts of the network to retrieve the candidate objects for those edges.

6.3 Scalability Experiments

In this section, we investigate the scalability of NAP with respect to various factors. To provide an indication of the space requirements, we note that for the largest tested data sizes (i.e., $|U|$=200000 and $|O|$=1024000), the AZ uses only 12.5 MBytes of main memory (including the network graph) and the LS needs a total of 23.5 MBytes hard disk storage.

End-to-end time. Before a lower level study, we present an experiment on the overall response latency. Specifically, from the user's viewpoint, the end-to-end time captures the elapsed time between issuing a query and obtaining the results. It includes the processing time at AZ, the computation time at LS, and the communication time between AZ and LS. Figure 13 shows the end-to-end time as a function of the anonymity degree K, assuming a communication bandwidth of 10Mbps. Clearly, the processing cost at LS dominates the end-to-end time, while the communication (between AZ and LS) and the AZ computations account only for a small percentage of the total time. It is worth mentioning that the processing (including anonymization and refinement) at AZ takes 0.000620 seconds for RE. HN and DF have similar costs. This implies that the AZ is capable of serving 1600 requests per second.

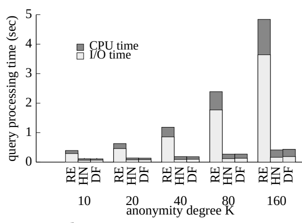


(a) kNN queries (b) Range queries

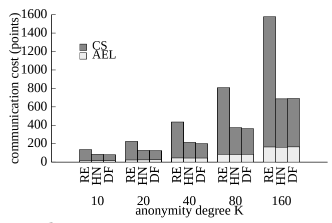
Fig. 13. End-to-end time vs. anonymity degree K

Effect of anonymity degree K. In Figure 14, we look deeper in the previous experiment, focusing on LS time and communication cost. Figure 14(a) shows the LS processing cost of kNN queries, and its breakdown in I/O and CPU time. In RE, the AEL edges are scattered in the network and many different disk pages have to be fetched at LS. Also, its AEL rarely contains edges that share common border nodes, leading to numerous network expansions and high CPU time. In HN and DF, edges close in the ordering are usually located nearby in the network. This improves the access locality of index structures at LS. Also, edges in AEL have many common border nodes and their network expansions are shared. As a result, both HN and DF have low I/O and CPU time.

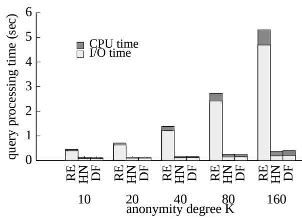
Figure 14(b) plots the communication cost of kNN queries, and its breakdown in AEL size and CS size. Even at large values of K, the AEL size accounts for a small fraction of the total communication cost. In RE, edges in AEL are broadly distributed, producing a large CS size. The performance gap between RE and HN/DF widens as K increases. Figures 14(c), 14(d) illustrate the processing cost (at LS) and the



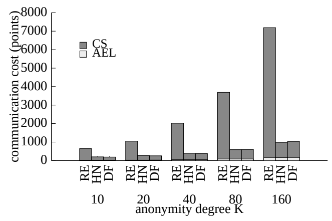
(a) kNN queries, LS cost



(b) kNN queries, comm. cost



(c) Range queries, LS cost

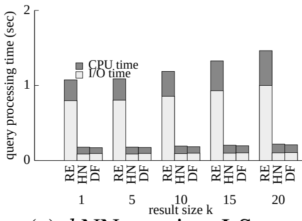


(d) Range queries, comm. cost

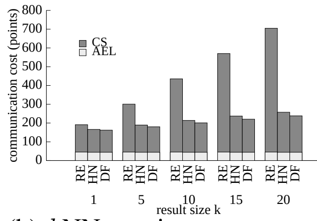
Fig. 14. Effect of anonymity degree K

communication cost of range queries, with respect to the anonymity degree K. The trends are similar to kNN queries.

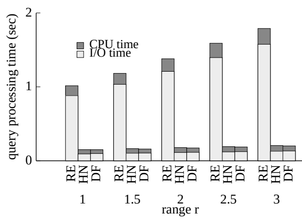
Effect of query selectivity. Figures 15(a), 15(b) plot the LS cost and communication cost of kNN queries, by varying the result size k. The orderings HN and DF preserve the locality of edges, implying a high degree of computation (and candidate object) sharing. As a result, their processing time and CS size only rises slightly as k increases. On the other hand, the cost of RE grows rapidly with k. Figures 15(c), 15(d) show the cost of range queries, as a function of the range parameter r. Again, the trends resemble those for kNN queries.



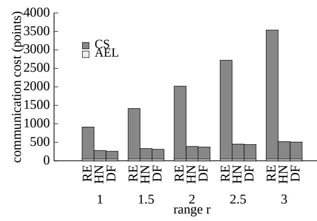
(a) kNN queries, LS cost



(b) kNN queries, comm. cost



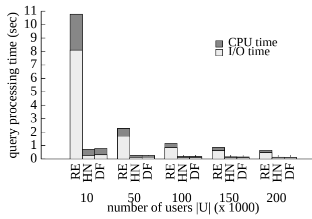
(c) Range queries, LS cost



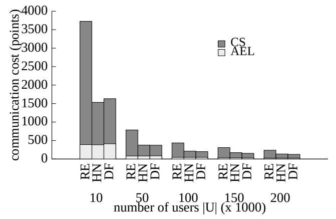
(d) Range queries, comm. cost

Fig. 15. Effect of query selectivity

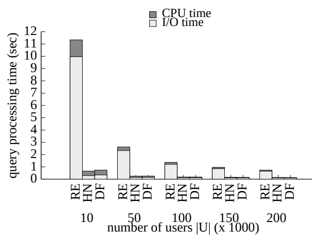
Effect of user cardinality $|U|$. Figures 16(a), 16(b) plot the LS cost and communication cost of kNN queries, with respect to user cardinality $|U|$. Observe that the AEL size (and, thus, the LS processing cost too) is inversely proportional to $|U|$; lower user density implies more edges inside the AEL. At the lowest $|U|$ value, the cost of RE becomes unacceptably high. On the other hand, for HN and DF, network expansions of common border nodes are shared, alleviating the impact of $|U|$ on the performance. Similar trends are observed for range queries in Figures 16(c), 16(d).



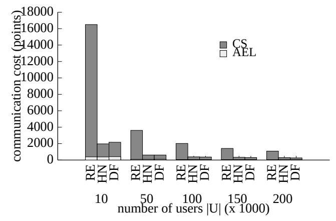
(a) kNN queries, LS cost



(b) kNN queries, comm. cost



(c) Range queries, LS cost

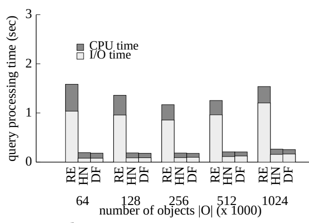


(d) Range queries, comm. cost

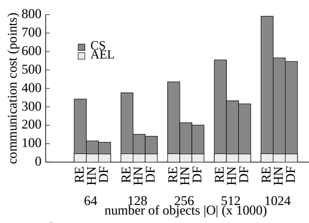
Fig. 16. Effect of user cardinality $|U|$

Effect of object cardinality $|O|$. Figures 17(a), 17(b) show the LS cost and communication cost of kNN queries, as a function of object cardinality $|O|$. Interestingly, the processing cost of RE first decreases and then increases. This is attributed to the accesses of different index structures at LS. For low $|O|$ value, the network expansion tends to be large, leading to many page accesses in the adjacency index; for high $|O|$ value, more objects fall on the edges in AEL, increasing the number of page accesses in the object R-tree (ORT). Regarding the communication cost (dominated by CS size), it mainly depends on the number of objects lying on AEL edges, which is high for a large $|O|$ value. The results for range queries are plotted in Figures 17(c), 17(d). In general, when $|O|$ is high, more objects fall within the query range of AEL edges, leading to high processing and communication cost.

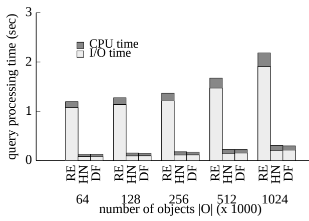
Batch processing experiment. In the next experiment, we investigate the processing of queries in batches (as opposed to one-by-one). Within the same batch, half of the queries are of kNN type and the others are range queries. Their selectivities (k and r) are randomly chosen among the corresponding values in Table 3. Figure 18 plots the average cost per query as a function of the batch size (i.e., number of queries in the batch). The I/O time of HN and DF decreases with the batch size due to improved sharing in query processing. On the other hand, the AEL edges of RE are scattered in the network, limiting the effectiveness of shared query processing. The communication cost is not sensitive to the batch size, since the queries do not have many common candidates.



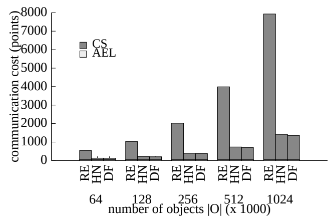
(a) kNN queries, LS cost



(b) kNN queries, comm. cost

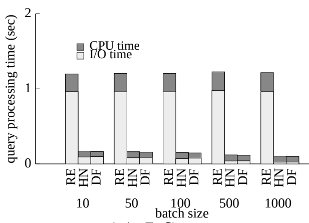


(c) Range queries, LS cost

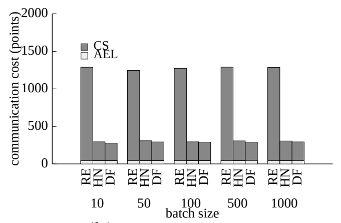


(d) Range queries, comm. cost

Fig. 17. Effect of object cardinality $|O|$



(a) LS cost



(b) comm. cost

Fig. 18. Effect of batch size

7 Conclusion

In this paper, we propose the *network-based anonymization and processing* (NAP) framework, the first system for K-anonymous query processing in road networks. NAP relies on a global user ordering and bucketization that satisfies reciprocity and guarantees K-anonymity. We identify the ordering characteristics that affect subsequent processing, and qualitatively compare alternatives. Then, we propose query evaluation techniques that exploit these characteristics. In addition to user privacy, NAP achieves low computational and communication costs, and quick responses overall. It is readily deployable, requiring only basic network operations.

In the traditional spatial anonymity model, the data owner (e.g., a location-based service) makes its data available using a location server. It may, however, be the case that the owner is outsourcing its database to a third-party (and, thus, untrusted) location server. A challenge here is how to encrypt the owner's data so that they are hidden from the location server, while it can still process anonymous queries. Another interesting question is how (anonymous) users could verify that the location server did not tamper with the original owner data.

References

[1] *http://www.anonymizer.com/*.

[2] *http://www.mailshell.com/*.

[3] *http://www.spamgourmet.com/*.

[4] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, N. Mishra, R. Motwani, U. Srivastava, D. Thomas, J. Widom, and Y. Xu. Vision Paper: Enabling Privacy for the Paranoids. In *VLDB*, 2004.

[5] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic Databases. In *VLDB*, 2002.

[6] A. R. Beresford. *Location privacy in ubiquitous computing*. PhD thesis, Computer Laboratory, University of Cambridge, 2005.

[7] T. Brinkhoff. A Framework for Generating Network-based Moving Objects. *GeoInformatica*, 6(2):153–180, 2002.

[8] A. R. Butz. Alternative Algorithm for Hilbert's Space-Filling Curve. *IEEE Trans. Comput.*, C-20(4):424–426, 1971.

[9] C.-Y. Chow and M. F. Mokbel. Enabling private continuous queries for revealed user locations. In *SSTD*, 2007.

[10] C.-Y. Chow, M. F. Mokbel, and X. Liu. A peer-to-peer spatial cloaking algorithm for anonymous location-based service. In *GIS*, 2006.

[11] E. W. Dijkstra. A Note on Two Problems in Connection with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[12] M. Duckham and L. Kulik. A Formal Model of Obfuscation and Negotiation for Location Privacy. In *PERVASIVE*, 2005.

[13] J. Edmonds and E. Johnson. Matching, Euler Tours and the Chinese Postman. *Mathematical Programming*, 5:88–124, 1973.

[14] B. Gedik and L. Liu. Location Privacy in Mobile Systems: A Personalized Anonymization Model. In *ICDCS*, 2005.

[15] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private Queries in Location Based Services: Anonymizers are not Necessary. In *SIGMOD*, 2008.

[16] G. Ghinita, P. Kalnis, and S. Skiadopoulos. PRIVE: Anonymous Location-based Queries in Distributed Mobile Systems. In *WWW*, 2007.

[17] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *MobiSys*, 2003.

[18] E. G. Hoel and H. Samet. Efficient Processing of Spatial Queries in Line Segment Databases. In *SSD*, 1991.

[19] X. Huang, C. S. Jensen, and S. Saltenis. The Islands Approach to Nearest Neighbor Querying in Spatial Networks. In *SSTD*, 2005.

[20] P. Indyk and D. Woodruff. Polylogarithmic Private Approximations and Efficient Matching. In *Theory of Cryptography Conference*, 2006.

[21] N. Jefferies, C. J. Mitchell, and M. Walker. A Proposed Architecture for Trusted Third Party Services. In *CPA*, 1995.

[22] N. Jing, Y. W. Huang, and E. A. Rundensteiner. Hierarchical Encoded Path Views for Path Query Processing: An Optimal Model and Its Performance Evaluation. *IEEE TKDE*, 10(3):409–432, 1998.

[23] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preserving Location-based Identity Inference in Anonymous Spatial Queries. *IEEE TKDE*, 19(12), 2007.

[24] A. Khoshgozaran and C. Shahabi. Blind Evaluation of Nearest Neighbor Queries Using Space Transformation to Preserve Location Privacy. In *SSTD*, 2007.

[25] H. Kido, Y. Yanagisawa, and T. Satoh. An Anonymous Communication Technique using Dummies for Location-based Services. In *ICPS*, 2005.

[26] M. R. Kolahdouzan and C. Shahabi. Voronoi-Based K Nearest Neighbor Search for Spatial Network Databases. In *VLDB*, 2004.

[27] T. J. Lehman and M. J. Carey. Query Processing in Main Memory Database Management Systems. In *SIGMOD*, 1986.

[28] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The New Casper: Query Processing for Location Services without Compromising Privacy. In *VLDB*, 2006.

[29] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao. Efficient OLAP Operations in Spatial Data Warehouses. In *SSTD*, 2001.

[30] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query Processing in Spatial Network Databases. In *VLDB*, 2003.

[31] S. Shekhar and D. Liu. CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations. *IEEE TKDE*, 19(1):102–119, 1997.

[32] S. Skiena. Eulerian Cycles. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Addison-Wesley, pages 192–196, 1990.

[33] L. Sweeney. k-Anonymity: A Model for Protecting Privacy. *IJUFKS*, 10(5):557–570, 2002.

[34] R. J. Wilson. An Eulerian Trail through Konigsberg. *Journal of Graph Theory*, 10:265–275, 1986.

[35] M. L. Yiu, C. S. Jensen, X. Huang, and H. Lu. SpaceTwist: Managing the Trade-Offs Among Location Privacy, Query Performance, and Query Accuracy in Mobile Services. In *ICDE*, 2008.

[36] M. L. Yiu and N. Mamoulis. Clustering objects on a spatial network. In *SIGMOD*, 2004.