Nearest Neighbor Search with Strong Location Privacy

Stavros Papadopoulos
The Chinese University of Hong Kong
stavros@cse.cuhk.edu.hk

Spiridon Bakiras
John Jay College, City University of New York
sbakiras@jjay.cuny.edu

Dimitris Papadias
The Hong Kong University of Science and Technology
dimitris@cse.ust.hk

ABSTRACT

The tremendous growth of the Internet has significantly reduced the cost of obtaining and sharing information about individuals, raising many concerns about user privacy. Spatial queries pose an additional threat to privacy because the location of a query may be sufficient to reveal sensitive information about the querier. In this paper we focus on $k$ nearest neighbor ($k$NN) queries and define the notion of strong location privacy, which renders a query indistinguishable from any location in the data space. We argue that previous work fails to support this property for arbitrary $k$NN search. Towards this end, we introduce methods that offer strong location privacy, by integrating private information retrieval (PIR) functionality. Specifically, we employ secure hardware-aided PIR, which has been proven very efficient and is currently considered as a practical mechanism for PIR. Initially, we devise a benchmark solution building upon previous work. Subsequently, we identify its drawbacks and present a novel scheme called AHG to tackle them. Finally, we demonstrate the performance superiority of AHG over our competitor, and its viability in applications demanding the highest level of privacy.

1. INTRODUCTION

The embedding of positioning capabilities (e.g., GPS) in mobile devices facilitates the emergence of location-based services (LBS), which is considered as the next “killer application” in the wireless data market. Location-based services allow clients to query a service provider (such as Google or Bing Maps) in a ubiquitous manner, in order to retrieve detailed information about points of interest (POIs) in their vicinity (e.g., restaurants, hospitals, etc.). However, similar to web searches or online purchases, location-dependent queries may disclose sensitive information about an individual’s health, financial status, political affiliations, etc.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Articles from this volume were invited to present their results at the 36th International Conference on Very Large Data Bases, September 13–17, 2010, Singapore.

Proceedings of the VLDB Endowment, Vol. 3, No. 1
Copyright 2010 VLDB Endowment 2150-8097/10/09... $ 10.00.

Assume, for example, that a user wishes to find the nearest night clubs to his/her location. To conceal this information, the user may choose to transmit the query through an anonymizing network (e.g., Tor [1]) that hides his/her real IP address. Nevertheless, simply removing the IP address is not sufficient to protect the user’s identity, which can be inferred from the coordinates of the query and background knowledge (e.g., the user’s home address). Hence, truly private services necessitate location privacy, i.e., the LBS should be oblivious of the query location. Additionally, location privacy is desirable independently of the concealment of the user identity. For instance, consider a mobile user who asks for the nearest night clubs, but wishes to hide that he/she has visited the specific area. In this case, the user requires location privacy even if the provider can infer his/her identity.

In this paper we focus on $k$ nearest neighbor ($k$NN) queries targeting at the highest degree of privacy, which we term strong location privacy and define as follows:

**Definition 1.** A scheme provides strong location privacy, if the adversary cannot distinguish the query location from any other location in the data space.

There exist numerous techniques that can provide a certain degree of location privacy, even if they were originally proposed in a different security domain. These solutions can be classified according to three major concepts: (i) location obfuscation, (ii) data transformation, and (iii) private information retrieval (PIR). We argue that currently no methodology can support arbitrary $k$NN queries providing strong location privacy. More specifically, in location obfuscation techniques (e.g., [20, 26]) the LBS can restrict the client in a small sub space of the total domain, leading to weak privacy. Schemes based on data transformation (e.g., [16, 25]) are vulnerable to access pattern attacks [24], which may correlate the query with outliers, popular locations, etc.

Finally, PIR-based approaches utilize a PIR protocol (e.g., [19]) implementing a simple query primitive, which retrieves a specific database block from the LBS without the latter discovering which block was retrieved. This primitive is resistant to access pattern attacks. The client reduces a spatial query to a set of such private retrievals. To the best of our knowledge, there exist only two PIR-based methods [17, 11]. [17] deals with $k$NN search, proposing algorithms that may involve a variable number of PIR block retrievals per spatial query. Although each retrieval is completely private, the cardinality of the PIR requests per $k$NN query may reveal information, similar to that in access pattern attacks for
data transformation techniques. Consequently, these methods violate strong location privacy. On the other hand, [11] entails a single PIR retrieval per query. Therefore, it renders all queries indistinguishable, satisfying strong location privacy. Nevertheless, it can only handle single NN queries. Furthermore, it leads to a prohibitive computational and communication cost even for very small POI databases (≈ 1MB), because it relies on an expensive PIR protocol ([19]).

This is the first work to propose methods for arbitrary kNN search with strong location privacy. There are two main components in our schemes: (i) the PIR functionality, and (ii) the query plan. The former ensures that the LBS is oblivious of each block retrieved by the algorithms. We employ secure hardware PIR [24], which is currently the only practical choice for PIR in databases of non-negligible size. In particular, this mechanism offers private block retrievals with constant communication cost and amortized polylogarithmic computational cost. The latter translates to processing times close to one second even for Gigabyte databases, whereas other schemes (e.g., [19]) entail hours.

The query plan ensures that every query retrieves the same number of blocks during its execution. A trivial solution would enforce each query to retrieve a fixed and arbitrarily large number of blocks. Nevertheless, such a solution may greatly impact the performance of our schemes. Therefore, we propose algorithms that compute a tight upper bound for the block accesses that any query in the data space must perform, such that all its results are retrieved.

Initially, we construct a benchmark method, called BNC, by optimizing [17] and generating a query plan in order to enforce strong location privacy. Subsequently, we point out its drawbacks and propose a novel solution, called AHG, to tackle them. We experimentally compare AHG with BNC using rigorous secure hardware simulations, and show that AHG outperforms BNC in all settings. We also demonstrate that AHG features response times in the order of a few seconds when testing with moderate POI databases (≈ 130 MB), and scales quite well under Gigabyte databases. Therefore, AHG constitutes the first viable solution for applications where strong location privacy is critical.

2. RELATED WORK

Section 2.1 reviews location obfuscation methods, Section 2.2 describes schemes that employ data transformation to protect location privacy, and Section 2.3 presents PIR-based location privacy techniques.

2.1 Location Obfuscation

This category includes every method that expands the LBS’s assumption about the actual query location to a wider sub space of the spatial domain, called obfuscation region. In [18, 7, 8], except for its actual query, the client sends to the LBS an additional set of “dummy” queries. The obfuscation region consists of the distinct locations included in the query set sent to the LBS. Cheng et al. [4] assume that the clients issue range queries, and the POIs are other clients’ locations. All locations must be protected. Therefore, each location is obfuscated into a circular region. The LBS processes the query and returns a probabilistic answer, which is modeled by the overlap of the circular regions with the query range. *SpaceTwist* [26] is an incremental NN algorithm executed at the LBS, which starts from a random location generated by the client and terminates when the client receives all its actual NNs. The obfuscation region is a subset of the data space covered by the algorithm.

In the *Spatial K-anonymity* [20, 15, 9, 12] paradigm, the client sends its query to a trusted anonymizer, which constructs an anonymizing spatial region (ASR) that contains the querier’s location along with another $K - 1$ client locations. The anonymizer then sends the ASR to the LBS. The latter executes the query with respect to the ASR, and returns a superset of the results to the anonymizer, which filters out the false positives. The obfuscation region is the set of the $K$ locations in the ASR.

All location obfuscation approaches guarantee weak location privacy because the obfuscation region is usually a small sub space of the total 2D domain. Nevertheless, they typically feature low query processing cost, due to the inexpensive operations they entail.

2.2 Data Transformation

In this setting the data owner is different from the LBS. The owner transforms the database (using some encoding methodology) prior to transmitting it to the LBS. An authorized client that possesses the secret transformation keys issues an *encoded query* to the LBS. Both the database and the queries are unreadable by the LBS and, thus, location privacy is protected. The goal is to provide the LBS with searching capabilities over the encoded data.

*OPES* [2] encodes the data in a way such that their numeric order is preserved, thus allowing simple distance comparison operations. Wong et al. [25] propose a secure point transformation, which preserves the relative distances of all the database POIs to any query point. This property renders kNN processing feasible. Another solution [16] transforms the points using the Hilbert mapping [21], and the parameters of the transformation (order, scale, orientation, etc.) are maintained secret. This technique allows approximate NN search directly on the transformed points.

Data transformation methods provide a stronger notion of location privacy than obfuscation. However, they are more computationally intensive due to the encoding/decoding operations. Additionally, they are prone to access pattern attacks [24] because the same query always returns the same encoded results. For example, the LBS may observe the frequencies of the returned ciphertexts. Having knowledge about the context of the database, it can match the most popular plaintext POI with the most frequently returned ciphertext and, thus, unravel information about the query.

2.3 PIR-based Location Privacy

Suppose that a server maintains a database consisting of $N$ sequential blocks. PIR protocols enable a client to retrieve the $i$th block from the server, without the server discovering which block was requested (i.e., index $i$). These protocols safeguard against access pattern attacks. They can be grouped into: (i) information theoretic [5, 3], (ii) computational [19, 10], and (iii) secure hardware [14, 23, 24]. The former are secure against even a computationally unbounded adversary. Nevertheless, they assume the existence of a fixed number of non-colluding servers. Computational PIR methods are applicable even for a single server, and they rely on the computational intractability of well-known problems (e.g., the φ-hiding hardness assumption in [10]). However, they entail expensive operations linear in the database size, which lead to prohibitive processing costs (in the order
of thousands of seconds even for moderate database sizes). Secure hardware PIR is currently the only practical PIR mechanism. It relies on a tamper-resistant CPU that is positioned at the server and is trusted by the clients. This CPU receives a client block request, which is unreadable by the server. It obliviously extracts the requested block from the server’s disk, and returns it to the client in an encrypted form decipherable solely by the client. This paradigm leads to constant communication cost, and amortized polylogarithmic computational cost. The latter translates to processing times close to a second even for Gigabyte databases.

There exist two PIR-based solutions. [17] proposes kNN algorithms that reduce the query to a set of PIR block retrievals performed via secure hardware PIR. An important detail overseen is that two different queries may entail a variable number of PIR requests. Therefore, although each PIR retrieval is completely private as stated above, the cardinality of these retrievals may disclose location information similar to that in access pattern attacks in data transformation. Consequently, [17] does not provide strong location privacy. On the other hand, [11] satisfies this property because every query involves a single PIR request and, hence, all queries are indistinguishable. Nevertheless, this scheme focuses only on single NN processing. Moreover, it relies on the computational PIR protocol of [19] and, thus, inherits its excessive communication and computational costs.

In Section 4 we devise a competitor by optimizing [17] and constructing a query plan in order to satisfy strong location privacy. Moreover, note that [17] assumes that the kNN algorithm runs inside the secure hardware. Considering that coding on the secure hardware is cumbersome, this implementation choice makes application development difficult. On the contrary, we consider that the secure hardware supports private block retrieval as an interface that can be used by any external algorithm, thus enhancing the utility of the secure hardware.

3. SYSTEM MODEL

Section 3.1 presents our general system architecture, and Section 3.2 formalizes our security.

3.1 Architecture

Figure 1 illustrates the entities and their interaction in our model. An LBS possesses a database of POIs $DB$, and a client wishes to issue kNN queries on $DB$ without disclosing its location. The LBS constructs an index structure on $DB$. Subsequently, the LBS generates and sends to the LBS the corresponding set $Q_{i,j}$, $Q_{u,v}$, . . . of PIR queries. The LBS processes these queries and sends replies $C_{i,j}$, $C_{u,v}$, . . . to the client, who extracts the respective plain blocks $B_{i,j}$, $B_{u,v}$, . . . These blocks contain either results or index data, which facilitate the algorithm to determine the blocks to be retrieved in the next step. The above procedure is repeated until the collection of $Q$’s results.

In other words, a kNN query translates to an ordered list of PIR queries. There are two mandatory requirements for the security of our model that all kNN queries must follow: (i) the $DB$ databases must be queried in the same order, and (ii) each $DB$ in the order must involve the same number of PIR queries. Due to these requirements, the LBS must construct a query plan, which is defined as follows:

**Definition 2.** The query plan is an ordered list $QP = ((db_1, cnt_1), (db_2, cnt_2), \ldots)$, which specifies that every kNN query $Q$ must first issue exactly $cnt_1$ PIR requests on $DB_{db_1}$, then $cnt_2$ PIR requests on $DB_{db_2}$, etc.

$QP$ depends on $k$, the kNN algorithm and the dataset. The LBS creates $QP$ in an offline pre-processing stage, and makes it publicly available. The query algorithm at the client’s side takes into account $QP$ when generating the PIR queries. Computing $QP$ in a way that guarantees the successful result retrieval of any query in the data space, without compromising the efficiency of the query algorithm, is a challenging task.

3.2 Threat Model and Security

The adversary’s access is limited in the shaded region of Figure 1. In particular, the adversary can be either the LBS, or anyone who can infiltrate the LBS’s machine and/or the communication channel between the client and the LBS. We assume that the adversary is polynomially bounded. It also knows the query algorithm. The primary privacy target in our framework is strong location privacy. We do not seek to protect the database confidentiality. Therefore, we assume that $DB$ and the index are not encrypted. Finally, the adversary is “curious but not malicious”, i.e., it does not tamper with the authenticity of the results, or $QP$.

**Theorem 1.** Our model provides strong location privacy.

**Proof.** Due to the underlying secure hardware PIR protocol, $Q_{i,j}$ and $C_{i,j}$ do not disclose information about the corresponding requested block $B_{i,j}$ to any party other than the client and the tamper-resistant secure hardware. Furthermore, access pattern attacks based on multiple pairs $(Q_{i,j}, C_{i,j})$ are prevented. Finally, the query plan forces every kNN query to process the same number of PIR retrievals, on the same databases, in the same order. Consequently, all kNN queries become indistinguishable. 

\[1\]We use calligraphic notation for the PIR elements.
4. BENCHMARK SOLUTION - BNC

We devise a solution called BNC (for benchmark), by optimizing [17] and computing a query plan in order to enforce strong location privacy. Section 4.1 describes the structures and kNN algorithm of BNC, and Section 4.2 contains the query plan calculation.

4.1 Structures and kNN algorithm

**Structures.** Let DB be a POI database, where P ∈ DB has the form (P.id, P.x, P.y, P.ttrail); P.id is the unique identifier of P, (P.x, P.y) are P’s coordinates, and P.ttrail represents additional data associated with P. The LBS constructs a regular g × g grid G over the POIs, where cell cij is in the ith row and jth column. It then builds two databases DB1 and DB2, which comprise of blocks B1,i and B2,i, respectively. The size of each block is determined by the PIR protocol (4KB in our implementation).

We first focus on DB1. For every cell c ∈ G, the LBS creates a block B, which stores an entry (P.id, P.x, P.y, P.ptr) for each POI P that resides in c; P.id, P.x, P.y have the same meaning as mentioned above, and P.ptr will be explained soon. The block is padded with dummy (i.e., random) entries if it is not full. Furthermore, if B cannot accommodate the entries of all POIs in c, the LBS creates extra blocks that form a linked list with B. Subsequently, the LBS stores the first block (i.e., the head of the list) of each cell cij consecutively in DB1, in ascending order of cell row and column numbers. The extra blocks are appended in the end of DB1.

We illustrate the above in the example of Figure 2, which assumes database DB = {P1, P2, ..., P20}, a 6 × 6 grid, and block capacity equal to four (id, x, y, ptr) entries. The first block of DB1, B1,1, corresponds to the first cell in the row/column order, c11. This cell contains only one POI (P1). Therefore, B1,1 stores (P1.id, P1.x, P1.y, P1.ptr) and three dummy entries. Block B1,2 stores only dummy entries, since it corresponds to c12 that is empty. Now consider block B1,24 associated with c46. This cell contains five POIs, whose entries cannot fit in B1,24. Therefore, B1,24 stores the entries of P13, P14, P15, P16, and extra block B1,37 stores the entry of P17 (along with dummies). Observe that the extra block is not appended after B1,24. Instead, it is added in the end of DB1, and B1,24 stores the index of B1,37, i.e., 37.

![Figure 2: BNC example](image_url)

In order to create database DB2, the LBS scans the cells of G in the row/column order; for every encountered POI P, it appends entry (P.id, P.ttrail) in the end of DB2. Assuming block capacity equal to three (id, tail) entries in Figure 2, the LBS reads POIs P1, P2, ..., P20 in this order, and thus creates blocks B2,1 for P1, P2, P3, block B2,2 for P4, P5, P6, etc. If the last block of DB2 is not full, it is padded with dummy entries. Finally, P.ptr in a block entry of DB1 points to the block of DB2 that stores (P.id, P.ttrail) (e.g., P1.ptr points to B2,1 in Figure 2). We assume that the client is aware of the specifications of G (e.g., its granularity g) and the block organization policy of DB1 and DB2.

**Algorithm.** The kNN algorithm runs at the client and consists of two phases. The first phase implements CPM [22], the state-of-the-art grid-based kNN technique. CPM retrieves cells from the grid in ascending minimum distance from the query. This method always leads to the optimal cell retrieval, which corresponds to the cells that overlap with the circle centered at the query, with radius its distance to its kth NN. When the process determines that a cell cij must be accessed, it privately retrieves all the blocks associated with cij from DB1. This is feasible because the client can identify the index of the head block of cij in DB1 as (i − 1) · g + j, and thus access it issuing the respective PIR request. Moreover, it can locate and privately retrieve the potential extra blocks of cij via the linked list pointers. In the second phase, the algorithm determines the kNN result based on the coordinates included in the DB1 entries retrieved in the first phase. Subsequently, it locates the DB2 blocks that accommodate the result tails using the ptr pointers, and extracts them through the appropriate PIR requests.

Consider in Figure 2 the 2NN query Q. The algorithm privately retrieves from DB1 the blocks corresponding to the cells in the light grey region, i.e., B1,19, B1,20, B1,25, B1,26, B1,31 and B1,32. Next, it computes the final result {P18, P19}, and extracts from DB2 the blocks that contain their tails (i.e., B2,6 and B2,7).

Different kNN queries may involve a different number of PIR requests on DB1 and/or DB2. For example, the 2NN query Q’ in Figure 2 requires 4 PIR retrievals from DB1 (for blocks B1,11, B1,12, B1,17 and B1,18 corresponding to the cells in the dark grey area) and 3 from DB2 (for blocks B2,2, B2,3 and B2,4). On the other hand, Q necessitates 6 and 2, respectively. In order to render all queries indistinguishable, the LBS provides a query plan QP = ((1, cm11), (2, cm12)) to the client (its calculation is explained in Section 4.2). If the PIR requests on DB1 (DB2) in the first (second) phase of the kNN algorithm do not agree with QP, the client forms dummy PIR requests. For example, if QP = ((1, 6), (2, 2)) in Figure 2, the client must issue 2 dummy PIR requests in the first phase of Q’, whereas it does not need to issue any dummy PIR request for Q. The pseudo code of the kNN algorithm of BNC is included in Appendix B.

A final remark concerns the motivation behind the use of two different databases DB1 and DB2. Alternatively, we could create a single database DB, by including the tails in DB1 and discarding DB2. Nevertheless, (i) a populated cell is assigned a larger number of DB blocks than DB1 blocks, because of the added tails, (ii) the block segmentation in DB may lead to more PIR retrievals than in DB1 and DB2 collectively, and (iii) each PIR retrieval in DB is more expensive than in DB1/DB2, because of the increased size of DB (the PIR cost raises with the database size). The above facts suggest that using DB1 and DB2 is more likely to lead to a lower total query cost than employing DB.

**Comparison with [17].** In addition to some minor structure differences, BNC differs from [17] mainly in three respects: (i) BNC provides strong location privacy, whereas
[17] does not support this property. (ii) All kNN techniques in [17] lead to suboptimal cell accesses and, thus, suboptimal total PIR block retrievals from \( DB_1 \). For example, the progressive expansion technique first identifies a square region that contains at least \( k \) POIs, starting from query \( Q \)'s cell, and expanding the search around it in a concentric pattern. Then, it privately retrieves the corresponding blocks.

In Figure 2, this method accesses the cells within the thick square. On the other hand, BNC always achieves optimal cell accesses. (iii) [17] assumes that the secure hardware executes the kNN algorithm, whereas in our model the secure hardware implements private block retrieval as an interface (see related discussion in Section 2.3).

4.2 Query Plan

We present an algorithm for computing query plan \( QP = ((1, cnt_1), (2, cnt_2)) \), which forces all kNN queries first to perform \( cnt_1 \) PIR requests on \( DB_1 \), and then \( cnt_2 \) PIR requests on \( DB_2 \). \( cnt_1 \) and \( cnt_2 \) must be set in a way such that any query \( Q \) following \( QP \) successfully retrieves all its results (for algorithm correctness). This happens if and only if \( cnt_1 \) (\( cnt_2 \)) is larger than or equal to the number of PIR retrievals performed in \( DB_1 \) (\( DB_2 \)) by any \( Q \), executing the kNN algorithm without the plan. The challenge lies in the fact that assigning to the above variables arbitrarily large numbers may gravely impact the performance of BNC. Our algorithm tightly bounds \( cnt_1 \) and \( cnt_2 \). It relies on the following construction and theorem:

**Construction 1.** Let \( G_{QP} \) be a regular grid (potentially different from index grid \( G \)) capturing the entire data space, and \( c \) a cell of \( G_{QP} \). We run a range kNN algorithm [13] with \( c \) as the input range, which computes the kNN sets of every possible location in \( c \). Let \( PS \) be the union of these sets. We calculate for every vertex \( v_i \) of \( c \) its distance maxdist, to its farthest POI in \( PS \). Finally, we generate the Minkowski sum \( |c| \) of \( c \) with a circle of radius \( \max(\text{dist}, \text{maxdist}) \). We call the derived region as the **safe region** of \( c \), and denote it by \( SR_c \). We also denote the set of cells of \( G \) overlapping \( SR_c \) as \( CS_c \).

**Theorem 2.** Consider Construction 1 for cell \( c \in G_{QP} \). Let \( Q \) be a kNN query in \( c \), and \( BS_c \) represent the \( DB_1 \) blocks associated with the cells in \( CS_c \). The number of PIR requests performed on \( DB_1 \) for \( Q \) is upper bounded by max \( = |BS_c| \), where \( |BS_c| \) is the cardinality of \( BS_c \).

**Proof.** See Appendix C.

Simply stated, based on Construction 1 and Theorem 2, we can bound the maximum number \( \text{max}_c \) of PIR retrievals on \( DB_1 \) required by any query \( Q \) in a cell \( c \). Additionally, we can bound the maximum PIR retrievals on \( DB_1 \) required by any query \( Q \) in the entire data space, denoted by \( \text{max}_1 \), as follows. We perform Construction 1 for every \( c \in G_{QP} \), and calculate \( \text{max}_1 = \max_{c \in G_{QP}} \text{max}_c \). Furthermore, we can trivially bound the maximum number of PIR retrievals in \( DB_2 \) by \( \text{max}_2 = k \cdot \text{size(id, tail)} \), where \( \text{size(id, tail)} \) is the number of PIR blocks storing a \( DB_2 \) entry. Finally, we set \( cnt_1 = \text{max}_1 \) and \( cnt_2 = \text{max}_2 \) to derive query plan \( QP \), which satisfies the correctness of BNC. Observe that \( QP \) depends on \( k \), the underlying kNN algorithm, the dataset, and the granularity of \( G_{QP} \). The LBS generates \( QP \) in an offline, preprocessing stage and publishes it.

Varying the granularity of \( G_{QP} \) we can adjust a trade-off between the plan computation time and the plan effectiveness. The finer the granularity, the higher the effectiveness of the plan because \( SR_c \) becomes smaller in Construction 1 and, thus, leads to a smaller \( \text{cnt}_1 \). Therefore, each query entails fewer PIR retrievals. However, a finer granularity implies more cells and, hence, more executions of the range kNN algorithm involved in Construction 1. Consequently, the plan computation time raises.

5. OUR SOLUTION - AHG

There are two main shortcomings in BNC: (i) it privately retrieves one \( DB_1 \) block for every empty grid cell accessed by its kNN algorithm. (ii) The block segmentation in the \( DB_1 \) blocks inflates the database size and, thus, the cost of each PIR retrieval. As a result, BNC features an increased total query response time. In this section we present AHG (for Aggregate Hilbert Grid), which overcomes the above drawbacks by eliminating the empty space in the database blocks (i.e., the dummy entries). Section 5.1 discusses the database organization in AHG and its kNN algorithm, and Section 5.2 explains the plan computation.

5.1 Structures and kNN algorithm

**Structures.** The LBS constructs a regular grid \( G \) over the POI database \( DB \), where \( P \in DB \) has the same form as in BNC (i.e., \( \langle P.id, P.x, P.y, P.tail \rangle \)). Moreover, it creates a Hilbert curve [21] with the following properties: (i) its underlying grid \( G_H \) has the same cell size with \( G \), and granularity larger than or equal to that of \( G \), and (ii) the cells of \( G \) and \( G_H \) coincide, and \( G \) is completely contained in \( G_H \). Figure 3 depicts a Hilbert curve with granularity 8 × 8 (i.e., with order 3) considering the example setting of Figure 2. Notice that the lower left cell of \( G_H \) coincides with the lower left cell of \( G \) (the figure omits \( G_H \) for clarity). This particular curve construction enables each cell \( c_{ij} \in G \) to be mapped to a unique Hilbert value \( c_{ij}, H \), e.g., \( c_{11}, H = 0 \), \( c_{21}, H = 1 \), etc.

![Figure 3: AHG example](image-url)
Furthermore, \( c_{ij} \) is associated with an aggregate pair \((c_{ij}, S, c_{ij} \cdot N)\); \( c_{ij} \cdot N \) is the number of POIs contained in \( c_{ij} \), and \( c_{ij} \cdot S \) signifies the sum of the \( N \) values of the cells preceding \( c_{ij} \) in the order of their \( H \) values. In our example, pair \((c_{44}, S, c_{44} \cdot N) = (3, 1)\) indicates that there exist 3 POIs in the cells preceding \( c_{44} \) along the curve (i.e., \( P_1, P_2, P_3 \)), and that there is 1 POI in \( c_{44} \) (i.e., \( P_2 \)). The LBS creates a database \( DB_1 \) from the aggregate pairs, by storing them sequentially according to the Hilbert values of their respective cells. In our example and assuming that a block can accommodate 8 pairs, \( B_{1,1} \) contains the pairs that correspond to the first 8 cells along the Hilbert curve.

The LBS builds a second database \( DB_2 \) that stores entries of the form \((id, x, y, ptr)\). These entries are stored sequentially according to the Hilbert values of the cells that accommodate the respective POIs (ties are broken arbitrarily). In Figure 3, \( P_1, P_2, P_3 \) and \( P_4 \) are the first four POIs encountered along the Hilbert curve and, thus, are stored in the first block of \( DB_2 \) (i.e., \( B_{2,1} \)). Finally, the LBS constructs a third database \( DB_3 \) that sequentially stores \((id, tail)\) entries for the POIs, based on their corresponding entries in \( DB_2 \). In our running example, block \( B_{3,1} \) stores the entries of \( P_1, P_2 \) and \( P_3 \), whose entries appear first in \( DB_2 \). The \( ptr \) pointer of a \( DB_2 \) entry points to the \( DB_3 \) block that accommodates the respective POI tail. Observe that padding is only necessary for the last block of each database (e.g., dummy entries \( d' \) are inserted in \( B_{1,3} \) in our example). The client is aware of the specifications of \( G \), the Hilbert curve, and the database organization.

As we shall see, \( DB_2 \) and \( DB_3 \) in AHG serve the same purpose as \( DB_1 \) and \( DB_2 \) in BNC. Observe that \( DB_2 \) in AHG is smaller than \( DB_1 \) in BNC, because it contains dummy entries only in its last block. However, it lacks of index structure, i.e., we cannot efficiently locate the entries associated with a cell in \( DB_2 \). This motivates the use of \( DB_1 \) that acts as an index to \( DB_2 \). Finally, the Hilbert order in AHG allows a block to store entries based on the locality of their associated POIs/cells. This is likely to lead to fewer block retrievals during query processing.

**Algorithm.** We explain AHG focusing on the 2NN example query \( Q \) of Figure 3. The algorithm consists of three phases. The first phase entails two steps. In the first step, the algorithm identifies the minimum set of cells that are closest to \( Q \) and collectively contain at least \( k = 2 \) POIs. This is achieved by privately retrieving and checking their corresponding aggregate pairs from \( DB_1 \). In our example, the algorithm first finds that \( c_{52} \) is the closest cell to \( Q \), and locates its aggregate pair in \( B_{1,3} \) (since \( c_{52} \cdot H = 18 \) and a \( DB_2 \) block has capacity 8 aggregate pairs). Then, it retrieves \( B_{1,3} \) through the appropriate PIR request. Next, it reads the pair of the next closest cell \( c_{51} \) (also in \( B_{1,3} \)), at which point it knows that the two cells store 2 POIs.

In the second step of the first phase, the algorithm calculates \( \text{maxdist} \) as the maximum distance from \( Q \) to \( c_{51} \) (the farthest from the visited cells). Subsequently, it extracts the pairs of the cells overlapping with the circle centered at \( Q \) with radius \( \text{maxdist} \), if they have not already been extracted. This step requires one additional block retrieval (of \( B_{1,2} \)). The examined cells (inside the thick square in the figure) guarantee to include the actual 2NN result of \( Q \).

In the second phase, AHG runs CPM [22] on the cells in the thick square region (i.e., it visits them in ascending order of their minimum distance from \( Q \)). \( c_{52} \) is the first cell to be accessed by CPM. Using the aggregate pair of \( c_{52} \) (i.e., \((10, 1)\)), AHG locates the entry of \( P_{19} \in c_{52} \) in block \( B_{2,3} \) (since it appears \( 11^{\text{th}} \) in \( DB_2 \), and each \( DB_2 \) block has capacity 4 entries). The algorithm continues similarly until CPM terminates its execution, which occurs when \( P_{18} \) is read from \( B_{2,3} \). Note that AHG can determine if a cell is empty (e.g., \( c_{41} \)) through its aggregate pair and, thus, it does not require a PIR retrieval from \( DB_2 \). AHG leads to an optimal PIR block retrieval from \( DB_2 \) due to CPM.

The third phase involves retrieving only the 2NN results \((P_{18}, P_{39})\) from \( DB_3 \) (stored in \( B_{3,1} \)), using the \( ptr \) pointers of their corresponding \( DB_2 \) entries. This phase is identical to the second phase of BNC. Finally, in order to enforce strong location privacy, the LBS must provide the client with a query plan \( QP = ((1, cnt_1), (2, cnt_2), (3, cnt_3)) \), whose computation is described in Section 5.2. Every query \( Q \) must perform exactly \( cnt_1 \cdot cnt_2 \cdot cnt_3 \) PIR retrievals on \( DB_1 \), \( DB_2 \) and \( DB_3 \), respectively. Similar to the case of BNC, if \( Q \) requires fewer retrievals from a database than indicated by the plan, the client issues additional dummy PIR requests. The pseudo code of AHG is contained in Appendix D.

Compared to BNC, AHG incurs PIR retrievals from one extra database, i.e., \( DB_1 \). However, this cost is balanced out by the following facts: (i) The \( DB_1 \) retrievals are cheap because \( DB_1 \) is typically very small. (ii) \( DB_2 \) in AHG is smaller than \( DB_1 \) in BNC and, thus, entails a lower PIR cost. (iii) The cells visited by CPM in phase two of AHG are the same as those accessed in phase one of BNC. Nevertheless, their associated \( DB_2 \) entries in AHG appear in fewer blocks than in \( DB_1 \) in BNC. This is because of the elimination of the dummy entries and the effective entry grouping due to the Hilbert order. Therefore, AHG involves fewer PIR retrievals. For example, \( Q \) in AHG (Figure 3) entails a single retrieval from \( DB_2 \), contrary to BNC where it requires 6 retrievals from \( DB_1 \) (Figure 2).

**5.2 Query Plan**

We present an algorithm that computes query plan \( QP = ((1, cnt_1), (2, cnt_2), (3, cnt_3)) \), which forces every \( kNN \) query to perform exactly \( cnt_1 \cdot cnt_2 \cdot cnt_3 \) PIR retrievals on \( DB_1 \) (resp. \( DB_2 \)). Our algorithm is based on the following construction and theorems:

**Construction 2.** Let \( G_{QP} \) be a regular grid (potentially different from index grid \( G \)) capturing the entire data space, and \( c \) a cell of \( G_{QP} \). We run a range \( kNN \) algorithm [13] with \( c \) as the input range, which computes the \( kNN \) sets of every possible location in \( c \). Let \( PS \) be the union of these sets. We construct a square region \( R \), by initially setting it equal to the \( G \) cells that overlap with \( c \), and expanding it by including all the \( G \) cells that surround it in a concentric pattern, until \( R \) covers \( PS \). We calculate for every vertex \( v \) of \( c \) its maximum distance \( \text{maxdist} \), from the vertices of \( R \). Finally, we generate the Minkowski sum [6] of \( v \) with a circle of radius \( \text{maxdist} \cdot v \), called the derived region as the \textit{safe region} of \( c \), and denote it by \( SR_c \). We also denote the set of cells of \( G \) overlapping \( SR_c \) as \( CS_c \).

**Theorem 3.** Consider Construction 2 for cell \( c \in G_{QP} \). Let \( Q \) be a \( kNN \) query in \( c \), and \( BS_c \) represent the \( DB_1 \) blocks associated with the cells in \( CS_c \). The number of PIR requests performed on \( DB_1 \) for \( Q \) is upper bounded by \( \max_{c} |BS_c| \), where \( |BS_c| \) is the cardinality of \( BS_c \).

**Proof.** See Appendix E.  □
Theorem 4. Consider Construction 1 (Section 4.2) for cell \( c \in G_Q \). Let \( Q \) be a \( k \)NN query in \( c \), and \( B_S \) represent the \( DB_2 \) blocks associated with the cells in \( CS \). The number of \( PIR \) requests performed on \( DB_2 \) for \( Q \) is upper bounded by \( \max_1^2 = |B_S| \), where \( |B_S| \) is the cardinality of \( B_S \).

Proof. See Appendix F.

Similar to BNC, the query plan algorithm in AHG performs Construction 2 for every cell \( c \in G_Q \), and computes an upper bound for the maximum \( PIR \) requests in \( DB_1 \) as \( \max_1 = \max_{c \in G_Q} \max_1^2 \). In a similar manner, through Construction 1 it bounds the maximum \( PIR \) requests in \( DB_2 \) as \( \max_2 = \max_{c \in G_Q} \max_2^2 \). Next, it trivially bounds the maximum \( PIR \) requests in \( DB_3 \) by \( \max_3 = k \cdot \text{size}((id, tail)) \) (this is identical to the \( DB_2 \) case of BNC). Finally, it sets \( \text{cnt}_1 = \max_1, \text{cnt}_2 = \max_2 \), and \( \text{cnt}_3 = \max_3 \). The derived plan \( QP \) guarantees algorithm correctness.

6. EXPERIMENTAL EVALUATION

Setup. We implemented BNC and AHG in C++, and experimentally compared their performance on a Linux server with an Intel Core2 Duo CPU 2.53GHz and 4GB of RAM. The performance metrics under investigation are the computational cost at the LBS, the query response time, and the overall communication cost. We tested the algorithms using a real (skewed) dataset\(^2\) containing postal addresses from the North East USA (123K POIs). We assume that each POI is associated with a 1KB tail, resulting in a database \( DB \) of size 128MB. The \( DB \) databases of BNC and AHG derived from \( DB \) reside on secondary storage at the LBS. All database blocks consume 4 KB. We adopt rigorous models for simulating a private \( DB \) block retrieval with secure hardware \( PIR \), which are based on [24] and thoroughly described in Appendix A. Our simulation assumes the IBM 4764 secure coprocessor, and the Seagate Barracuda 7200.11 SATA 3Gb/s 1TB, 7200RPM hard drive. Finally, the clients submit their queries to the secure hardware (at the LBS) via encrypted connections. The bandwidth at the client side is 1 MB/s, while the network round-trip time (RTT) is 50 ms.

Query processing. In the first experiment we fine-tune the granularity of index grid \( G \), setting \( k = 10 \). Moreover, we assume that the query plans have been computed using a \( 200 \times 200 \) grid \( G_Q \), which provides high plan effectiveness for both methods. Figure 4(a) shows the computational cost of the two approaches. The colored regions in the bars indicate the total processing cost at the LBS, whereas the white regions correspond to the network overhead and the computational burden at the client. Therefore, the complete double bars represent the overall query response time. When the granularity is very coarse (5 x 5), each grid cell contains a large number of POIs. Consequently, BNC performs numerous \( DB_1 \) \( PIR \) requests for every visited cell, in order to retrieve the associated POI entries. Similarly, AHG entails many \( PIR \) retrievals in \( DB_2 \) for the same reason. As the grid granularity raises, both methods converge to their optimal configuration, which is \( 10 \times 10 \) for BNC and \( 50 \times 50 \) for AHG. Increasing the granularity above the optimal configuration has a negative effect because more cells are visited during the \( k \)NN algorithms. In BNC, this increases the number of \( PIR \) retrievals in \( DB_1 \), since a \( PIR \) request is performed even for an empty cell. The performance of AHG deteriorates mainly because more \( PIR \) accesses are involved in \( DB_1 \); the costly \( DB_2 \) \( PIR \) retrievals are minimized in the presence of empty cells due to the elimination of block segmentation. Since the \( PIR \) accesses in \( DB_1 \) are cheap (due to the small size of \( DB_1 \)), AHG degrades more slowly than BNC.

![Figure 4: Performance vs. \( G \) granularity](image)

Figure 4(b) depicts the overall communication cost between the LBS and the client for the same experiment. Note that each \( PIR \) request involves one query from the client to the LBS (128 bytes) plus the result block (4 KB) from the LBS to the client. The communication cost follows the same trend as in Figure 4(a), because it is determined solely by the total number of \( PIR \) requests. In the remaining set of experiments we set the grid granularity for BNC and AHG according to their optimal configurations derived here.

The next experiment assesses the effect of \( k \). Figure 5(a) illustrates the computational cost for BNC and AHG. Based on the secure hardware specifications described in Appendix A, each \( PIR \) request in AHG consumes 34 ms at \( DB_1 \), 367 ms at \( DB_2 \), and 992 ms at \( DB_3 \). For BNC, the \( PIR \) costs are 384 ms for \( DB_1 \) and 992 ms for \( DB_2 \). As \( k \) increases, the algorithms require more \( PIR \) retrievals from the LBS (in all databases), and the cost at the LBS increases. The response times are dominated by the processing at the LBS. The performance of AHG is 3 to 6 times better than that of BNC. The main reason is that AHG significantly reduces the costly \( DB_2 \) retrievals due to the effective Hilbert grouping, and the elimination of empty cells. The query times in AHG are within 7-29 seconds, and private 10NN queries (default setting) require 18 seconds, which is acceptable for “real-time” applications. The communication cost (Figure 5(b)) also increases slightly with \( k \), with AHG being considerably cheaper than BNC for the same reason as discussed above. Furthermore, AHG requires less than 200 KB of data for all cases, which is lower than 0.2% of the \( DB \) size.

![Figure 5: Performance vs. \( k \)](image)

Figure 5: Performance vs. \( k \)

Query plan. Figures 6(a) and 6(b) illustrate the computational and communication cost as a function of the \( G_Q \) granularity (\( k = 10 \)). A finer grid produces more effec-
tive query plans for both methods, which reduce the total PIR queries and, thus, the overall query response times and bandwidth consumption. Observe that a granularity greater than 50 × 50 has small impact on the effectiveness of the plan. This is because the plan already tightly bounds the necessary PIR retrievals, which cannot be decreased any further.

7. CONCLUSION

We propose sophisticated solutions that decompose a kNN search in spatial databases, which are common in both the BNC and AHG plans. Consequently, this cost is practically identical in the two methods. Furthermore, it raises quadratically as the $G_{QP}$ granularity elevates, due to the quadratic increase in the number of cells (and, thus, the range kNN executions), ranging from 248 seconds for a $10 \times 10 G_{QP}$, to 9755 seconds for a $200 \times 200 G_{QP}$. Recall, however, that the plan computation is offline.

Scalability. Finally, we discuss the scalability of AHG under larger database sizes. We perform the same experiment as in Figure 5, increasing this time the size of the POI tails from 1KB to 10KB, and omitting BNC from the discussion. This modification captures the case where the POIs include large additional data (e.g., images). The size of $DB_3$ becomes larger than 1 GB, and a PIR retrieval requires 1.51 seconds. $DB_1$ and $DB_2$ are unaffected. A POI entry in $DB_3$ now occupies 3 blocks instead of one, thus increasing the total number of PIR requests during query processing. The response times are now 11-99 seconds, whereas the bandwidth consumption is 93KB-363KB. Although we increased the database size by a factor of 10, the query cost does not significantly affect the overall cost.

8. REFERENCES


![Figure 6: Performance cost vs. $G_{QP}$ granularity](image-url)
APPENDIX

A. SECURE HARDWARE PIR

We present in detail the secure hardware PIR scheme of [24], which is utilized in our experimental evaluation in Section 6. This technique allows a client to retrieve a block $B_i$, from a database $DB = \{B_1, B_2, \ldots, B_N\}$ hosted by a server, without the latter discovering which block is requested (i.e., index $i$). Appendix A.1 describes the protocol, and Appendix A.2 discusses its performance.

A.1 Protocol

Figure 7 outlines the system architecture. A Secure Co-processor (SCOP) is positioned at the server, which contains a small cache capable of storing $\sigma \sqrt{N}$ blocks, where $N$ is the total number of blocks in $DB$ and $\sigma \geq 10$ is a security parameter. The SCOP is a general-purpose, tamper-resistant CPU, which can be trusted to carry out its processing unmolested and unobserved, even if the adversary has physical access to it. The SCOP communicates with the client through a secure SSL channel. Moreover, it has access to the server’s disk, where $DB$ resides.

Setup. A setup stage occurs before the system is set to motion. The SCOP scans $DB$, encrypts each block $B_i \in DB$ using a secret key, and secretly permutes the blocks. For simplicity, we omit the algorithm that obliviously permutes $N$ blocks using cache capacity of $\sigma \sqrt{N}$ blocks (for details see [24]). The secret key and permutation are stored in the cache of the SCOP, which is inaccessible to the adversary.

The SCOP creates a pyramid data structure with $\log_4 N$ levels in the server’s disk, where level $l$ ($1 \leq l \leq \log_4 N$) contains $4^l$ buckets. Each bucket accommodates up to $\log_4 N$ blocks. We assume that the blocks in the same bucket are stored sequentially in the disk. The pyramid structure is constructed incrementally as follows. The SCOP inserts the encrypted and permuted blocks of $DB$ in the top level one by one; each block is assigned to one of the buckets of this level as determined by a hash function. When the level becomes full, the SCOP empties it into the next level, after re-encrypting the blocks, and obliviously re-permuting them into the new buckets with a new hash function. The process continues by always inserting every new block in the top level, and performing level overflows recursively as described above.

Query. Suppose that a client asks for block $B_i$. It forms and sends a request $Q_i$ to the SCOP through the SSL channel, such that $Q_i$ is readable solely by the SCOP. The SCOP accesses the pyramid structure top-down as follows. In every visited level, it scans exactly one bucket as determined by $Q_i$ and the hash function used in this level. One of the scanned buckets is guaranteed to contain the encrypted form of $B_i$. The SCOP extracts $B_i$ and sends it to the client through the SSL channel in an encrypted form $C_i$, which is decipherable solely by the client. Finally, the SCOP re-encrypts $B_i$ and inserts it into the top level of the pyramid structure. Note that this insertion may lead to level overflows that are handled as previously discussed.

Security. The security of the scheme relies on two invariants: (i) the SCOP does not disclose the level accommodating the result block of the query, and (ii) it never looks at the same place for the same block. The former invariant holds because the SCOP accesses one bucket per every level of the structure. The latter is satisfied because the SCOP re-encrypts and inserts the result block into the top level, thus allowing the next query asking for the same block to find it in a different bucket than its previous retrieval (and also in a different encoded form).

A.2 Performance

The protocol features constant communication cost between the server and the client; $C_i$ is the ciphered version of the requested $B_i$, which typically consumes the same space as $B_i$. We next analyze the computational time required by the server to perform an oblivious block retrieval. This overhead involves an online cost and an offline cost. The online cost accounts for the overhead of the SCOP to retrieve the encrypted $B_i$ from the pyramid structure, re-encrypt it and store it into the top level. The offline cost captures the potential level overflows and re-shuffling.

In more detail, the online cost entails one bucket read per pyramid level, and one block write to the top level. Let $disk_{seek}$ be the disk seek time, $disk_{rw}$ the read/write throughput of the disk, $SCOP_{ed}$ the encryption/decryption throughput of the SCOP, and $SCOP_{rw}$ the read/write throughput of the SCOP. The online cost is given by:

\[
online\_cost = \frac{(\log_4 N + 1) \cdot disk_{seek}}{disk_{rw}} + \frac{(\log_4 N + 1) \cdot \log N \cdot block\_size}{disk_{rw}} + \frac{\log_4 N \cdot \log N + 1 \cdot block\_size}{SCOP_{ed}}
\]

Next, we focus on the analysis of the offline cost. Note that level $l$ overflows (and thus is re-shuffled) every $4^l$ queries (i.e., block insertions in the top level). Instead of computing the re-shuffling cost of each level $i$ per $4^l$ queries, [24] provides an amortized cost per query. Specifically, it estimates that $4^l \cdot \log_4 N \cdot \log N$ blocks are read from/written to the disk and get re-encrypted in every query due to level overflows. Moreover, the disk seek time is hidden by the above cost. Consequently, the amortized offline cost is calculated.
as:
\[
    offline\_cost = \frac{4^2 \cdot \log_4 N \cdot \log N \cdot block\_size}{disk\_rw} + \frac{4^2 \cdot \log_4 N \cdot \log N \cdot block\_size}{SCOP\_rw} + \frac{4^2 \cdot \log_4 N \cdot \log N \cdot block\_size}{SCOP\_ed} \tag{2}
\]

Table 1 illustrates typical values for the variables of Equations 1 and 2, assuming that we utilize the IBM 4764 secure coprocessor\(^3\) (similarly to [24]), and hard drive Seagate Barracuda 7200.11 SATA 3Gb/s 1TB, 7200RPM\(^4\). The database size in this setting is larger than 1GB. Note that the SCOP cache is equal to 32MB and, thus, can accommodate 8000 blocks. For \(\sigma = 10\), \(\sigma\sqrt{N} = 5000\) and, hence, the protocol can work under this sample configuration. Evaluating Equations 1 and 2 by substituting their variables with the values of Table 1, we derive that the amortized processing cost for one oblivious block retrieval in a 1GB database is equal to 1.432 seconds (0.133 seconds for the online cost, and 1.299 seconds for the offline cost).

### B. THE BNC PSEUDO CODE

Figure 8 illustrates the pseudo code of the \(k\)NN algorithm in BNC. The procedure takes as arguments query \(Q\), value \(k\), and query plan \(P = (1, cnt_1), (2, cnt_2)\) (treated as a two-dimensional array), where \(cnt_1, cnt_2\) indicates the number of PIR retrievals that must be performed on \(DB_1\) (\(DB_2\)). Lines 1-9 capture the first phase of the algorithm, whereas lines 10-13 correspond to the second phase.

### C. PROOF OF THEOREM 2

**Proof.** Figure 9(a) illustrates an example \(SR_c\) generated by Construction 1 for cell \(c \in G_{Q_P}\). We assume that \(c\) (depicted in solid thin black lines) coincides with a cell of grid \(G\) (shown in dashed grey lines) for simplicity. The proof for the case when \(c\) partially overlaps \(G\) cells is very similar and, thus, omitted. The illustrated points correspond to the set of POIs \(PS\) retrieved by a range \(2NN\) algorithm for \(c\), i.e., the \(2NN\) result of any query \(Q\) in \(c\) is a subset of \(PS\). Distance \(maxdist_c\) represents the distance from vertex \(V_3\) to \(P_1\), which is its farthest POI in \(PS\). Moreover, \(maxdist_c = \max_{V \in c} maxdist_c\). Therefore, \(SR_c\) is the Minkowski sum constructed as the union of all circles with radius \(maxdist_c\) and center any point in \(c\). \(SR_c\) is the shaded area in our figure. The cells of \(G\) overlapping with \(SR_c\) are within the thick square region, and are denoted by \(CS_c\). Finally, the \(DB_1\) blocks associated with the cells in \(CS_c\) comprise set \(BS_c\), whose cardinality is \(|BS_c|\).

\[\text{http://www-03.ibm.com/security/cryptocards} /\text{pcixcc/overhardware.shtml}\]

### BNC \(k\)NN(\(Q, k, P\))

1. \(cnt_1 = QP[0][1], cnt_2 = QP[1][1]\)
2. \(entries\_DB_1 = \emptyset\)
3. While \(entries\_DB_1.size < k\)
   4. \(c = \text{cell with the next smallest minimum distance from } Q\)
   5. Privately retrieve the blocks from \(DB_1\) associated with \(c\), and insert their entries in list \(entries\_DB_1\)
6. \(dst_k = \text{distance from } Q \text{ to its } k\text{th } NN \text{ in } entries\_DB_1\)
7. \(c\_set = \text{set of yet unseen cells overlapping with circle } C(Q, dst_k)\)
8. Privately retrieve the \(DB_2\) blocks associated with every cell \(c \in c\_set\), and insert their entries in \(entries\_DB_2\)
9. Issue dummy PIR requests until the total number of PIR accesses in \(DB_1\) becomes \(cnt_1\)
10. \(kNN\_set\_DB_1 = \text{set of } k\text{NN result entries in } entries\_DB_1\)
11. Privately retrieve the blocks pointed by the ptr pointers of \(kNN\_set\_DB_2\), and insert their entries in list \(entries\_DB_2\)
12. Issue dummy PIR requests until the total number of PIR accesses in \(DB_2\) becomes \(cnt_2\)
13. Compute the final result by joining \(kNN\_set\_DB_1\) with \(entries\_DB_2\) on \(id\)

**Figure 8:** The \(k\)NN query algorithm in BNC

**Figure 9:** Illustration of Construction 1 and proof of Theorem 2

Recall from Section 4.1 that a query \(Q\) in BNC retrieves the \(DB_1\) blocks associated with the \(G\) cells overlapping circle \(C(Q, dst(Q, P_k))\), i.e., the circle centered at \(Q\) with radius \(Q\)'s distance from its \(k\)th \(NN\) \(P_k\). In Figure 9(a), the 2nd \(NN\) of \(Q\) is \(P_2\) and circle \(C(Q, dst(Q, P_2))\) is illustrated in dark grey. If this circle is completely included in \(SR_c\), then its overlapping cells is a subset of \(CS_c\), and, thus, the corresponding \(DB_1\) blocks are a subset of \(BS_c\). This means that \(Q\)'s PIR accesses in \(DB_1\) are bounded by \(|BS_c|\). Consequently, if we prove that \(C(Q, dst(Q, P_k))\) is covered by \(SR_c\) for any \(Q, c\), we conclude the proof of our theorem.

Consider Figure 9(b), which shows a cell \(c \in G_{Q_P}\), a query \(Q \in c\) and the \(k\)th \(NN\) \(P_k\) of \(Q\). \(P_k\) can be either outside \(c\), or inside. We focus on the case \(P_k\) lies outside \(c\). We draw line segment \(QP_k\) and extend it towards the direction of \(Q\), until it meets cell side \(V_2V_3\) at point \(A\). Angle \(a = \angle V_2AP_k\) is greater than or equal to \(90^\circ\). Consequently, in triangle \(V_2AP_k\), side \(V_2P_k\) is the largest as it lies opposite of \(a\). This means that \(dst(V_2, P_k) \geq dst(Q, P_k)\), where the equality holds when \(Q\) coincides with \(V_2\).

According to Construction 1 and due to the definition of the Minkowski sum, circle \(C(Q, max_{V \in c} maxdist_c)\) is completely included in \(SR_c\). Moreover, \(dst(Q, P_k) \leq dst(V_2, P_k)\)
\( \leq \max_{V_{v,i} \in c} \max \text{dist} \). Therefore, \( C(Q, \text{dst}(Q, P_i)) \) is covered by \( C(Q, \max_{V_{v,i} \in c} \max \text{dist}_i) \) and, thus, also by \( SR_c \). The proof for the case when \( P_i \) lies inside \( c \) is identical. \( \square \)

**D. THE AHG PSEUDO CODE**

Figure 10 illustrates the pseudo code of the \( k \)NN algorithm in AHG. The procedure takes as arguments query \( Q \), value \( k \), and query plan \( Q^P = ((1, c_{nt_1}), (2, c_{nt_2}), (2, c_{nt_3})) \) (treated as a two-dimensional array), where \( c_{nt_1} \) and \( (c_{nt_2}, c_{nt_3}) \) indicates the number of PIR retrievals that must be performed on \( DB_1 \) \( (DB_2, DB_3) \). Lines 1-11 capture the first phase of the algorithm, whereas line 12 corresponds to the second and third phase.

**Figure 10: The \( k \)NN query algorithm in AHG**

**E. PROOF OF THEOREM 3**

**Proof.** Figure 11(a) illustrates an example \( SR_c \) generated by Construction 2 for cell \( c \in G_Q \). We focus on the case where \( c \) coincides with a cell of the index grid \( G \) for simplicity. The proof for the case when \( c \) partially overlaps \( G \) cells is very similar and, thus, omitted. The illustrated points correspond to the set of POIs \( P_S \) retrieved by a range \( 2 \)NN algorithm for \( c \), i.e., the \( 2 \)NN result of any query \( Q \) in \( c \) is a subset of \( P_S \). We calculate the square range \( R \) by first setting it equal to \( c \) and checking whether it contains all the POIs in \( P_S \). Since it does not, we expand it in a concentric pattern by including the \( G \) cells that surround \( c \). The resulting \( R \) covers all \( P_S \) and, thus, constitutes the final square region. Distance \( \max \text{dist}_i \) represents the maximum distance from vertex \( v_i \) to \( R \). Moreover, \( \max \text{dist}_i = \max_{V_{v,i} \in c} \max \text{dist}_i \). Therefore, \( SR_c \) is the Minkowski sum computed as the union of all circles with radius \( \max \text{dist}_i \) and center any point in \( c \). The \( SR_c \) is the shaded area in our figure. The cells of \( G \) overlapping with \( SR_c \) are within the thick square, and are denoted by \( CS_c \). Finally, the \( DB_1 \) blocks associated with the cells in \( CS_c \) comprise set \( BS_c \), whose cardinality is \( |BS_c| \).

Recall from Section 5.1 that, in the first step of the first phase, the \( k \)NN algorithm of AHG visits the minimum \( G \) cells that are closest to \( Q \) and collectively include at least \( k \) POIs (note that every cell access implies the private retrieval of the associated \( DB_1 \) blocks). In Figure 11(a), the \( 2 \)NN query \( Q \) visits the dark gray cells. Then, the maximum distance \( \max \text{dist} \) of \( Q \) to the vertices of these cells is calculated, and all the cells overlapping circle \( C(Q, \max \text{dist}) \) are visited in the second step of the first phase of AHG. Suppose that the cells visited in the first step are included in \( R \) as shown in our figure. Then, the cells accessed in the second step are completely covered by \( SR_c \). This is because \( \max \text{dist} \) is smaller than or equal to \( \max \text{dist}_i \) and, thus, \( C(Q, \max \text{dist}) \) is included in \( C(Q, \max \text{dist}_i) \), which is a part of \( SR_c \). Consequently, the cells overlapping \( C(Q, \max \text{dist}) \) are a subset of \( CS_c \) and, therefore, their corresponding \( DB_1 \) blocks are a subset of \( BS_c \). This means that \( |BS_c| \) upper bounds the necessary PIR requests in \( DB_1 \) by any \( Q \in c \), which proves our theorem. What remains is to prove that the cells visited by \( Q \) during the first step of the first phase of AHG are always included in \( R \), which we conduct below.

We prove by contradiction, utilizing Figure 11(b). Suppose that cell \( c' \) is the last cell retrieved in the first step, which lies outside \( R \). This means that (i) all cells in \( R \) have already been visited before \( c' \) because they are closer to \( Q \), and (ii) \( R \) contains strictly fewer POIs than \( k \). However, by definition \( R \) includes the \( k \)NN set of any query in \( c' \) and, hence, also of \( Q \). Consequently, \( R \) accommodates at least \( k \) POIs which reaches our contradiction. \( \square \)

**F. PROOF OF THEOREM 4**

**Proof.** Recall that, during the second phase of AHG and due to CPM, the \( k \)NN algorithm extracts the \( DB_2 \) blocks associated with the cells overlapping circle \( C(Q, \text{dst}(Q, P_i)) \), which is centered at \( Q \) and has radius the distance \( \text{dst}(Q, P_i) \) from \( Q \) to its \( k^{th} \) NN \( P_i \). As we explained in Appendix C, \( C(Q, \text{dst}(Q, P_i)) \) is completely contained in \( SR_c \). Therefore, the cells overlapping this circle are a subset of \( CS_c \) and, thus, their associated \( DB_2 \) blocks are a subset of \( BS_c \). Consequently, \( |BS_c| \) bounds the \( DB_2 \) PIR retrievals of \( Q \). \( \square \)