Casper*: Query Processing for Location Services without Compromising Privacy

CHI-YIN CHOW
MOHAMED F. MOKBEL
University of Minnesota
and
WALID G. AREF
Purdue University

In this paper, we present a new privacy-aware query processing framework *Capser** in which mobile and stationary users can obtain snapshot and/or continuous location-based services without revealing their private location information. In particular, we propose a *privacy-aware query processor* embedded inside a location-based database server to deal with snapshot and continuous queries based on the knowledge of the user's cloaked location rather than the exact location. Our proposed *privacy-aware query processor* is completely independent of how we compute the user's cloaked location. In other words, any existing location anonymization algorithms that blur the user's private location into cloaked rectilinear areas can be employed to protect the user's location privacy. We first propose a *privacy-aware query processor* that not only supports three new privacy-aware query types, but it also achieves a trade-off between query processing cost and answer optimality. Then, to improve system scalability of processing continuous privacy-aware queries, we propose a *shared execution paradigm* that shares query processing among a large number of continuous queries. The proposed scalable paradigm can be tuned through two parameters to trade off between system scalability and answer optimality. Experimental results show that our query processor achieves high quality snapshot and continuous location-based services while supporting queries and/or data with cloaked locations.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*query processing*; H.2.8 [**Database Management**]: Database Applications—*spatial databases and GIS*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Location privacy, continuous queries, privacy-aware query processing, location-based services

Authors' addresses: Chi-Yin Chow and Mohamed F. Mokbel, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455, USA; email: {cchow, mokbel}@cs.umn.edu. Walid G. Aref, Department of Computer Science, Purdue University, West Lafayette, IN 47907, USA; email: aref@cs.purdue.edu.
Chi-Yin Chow and Mohamed F. Mokbel's research was partially supported by NSF Grants IIS-0811998, IIS-0811935, and CNS-0708604. Walid G. Aref's research was partially supported by NSF Grant IIS-0811954.

1. INTRODUCTION

Location-based services (LBS) combine the functionality of location-aware devices (e.g., GPS-like devices), wireless and cellular phone technologies, and information management to provide personalized services to users based on their current locations. Examples of LBS include location-aware emergency services (*"Dispatch the nearest ambulance"*), location-based advertisement (*"Send e-coupons to all cars that are within two miles of my gas station"*), live traffic reports (*"Let me know if there is congestion within ten minutes of my route"*), and location-based store finders (*"Where is my nearest restaurant"*). Users registered with LBS continuously send their locations to a location-based database server. Upon requesting a service, a registered user issues a location-based query that is executed at the server based on the knowledge of the user's location [Jensen 2004; Mokbel and Aref 2005; Mouratidis et al. 2005; Wolfson et al. 2002]. Location-based queries are either *snapshot* or *continuous* queries. Examples of *snapshot* queries include "*Where is my nearest gas station*" and "*What are the restaurants within one mile of my location*", while examples of *continuous* queries include "*Continuously report my nearest police car*" and "*Continuously report the gas stations within one mile of my car*".

Although LBS promise safety and convenience, they threaten the privacy and security of their users. The privacy threat comes from the fact that LBS providers rely mainly on an implicit assumption that users agree to reveal their private locations to get services. In other words, a user trades her privacy with the service. If a user wants to keep her private location information, she has to turn off her location-aware device and (temporarily) unsubscribe from the service. With untrustworthy servers, such a model poses several privacy threats. For example, an employer may check on her employee's behavior by knowing the places where she visits and the time of each visit, the personal medical records can be inferred by knowing which clinic a person visits, or someone can track the locations of his ex-friends. In fact, in many cases, GPS devices have been used in stalking personal locations [FoxNews 2004; USAToday 2002]. Unfortunately, the traditional approach of *pseudonymity* (i.e., using a fake identity) [Pfitzmann and Kohntopp 2000] is not applicable to LBS where the location information of a person can directly lead to the true identity. For example, asking about the nearest Pizza restaurant to my house using a fake identity will reveal my true identity, as a resident of the house.

In an attempt to preserve the privacy of LBS users, several research groups have presented the concept of a *location anonymizer* that is responsible for blurring actual users' locations into *cloaked areas* (e.g., see [Bamba et al. 2008; Mokbel et al. 2006; Chow et al. 2006; Chow and Mokbel 2007; Gruteser and Grunwald 2003; Ghinita et al. 2007a; Gedik and Liu 2005; 2008; Kalnis et al. 2007; Xu and Cai 2008]). Upon registration with the *location anonymizer*, mobile users specify their own desired level of privacy through a user-specified *privacy profile* that may contain one or more of the following parameters: k-anonymity, minimum area A_{min}, and maximum area A_{max}. k-anonymity indicates that the user wants to be k-anonymous, i.e., not distinguishable among k users, while A_{min} and A_{max} indicate that the user wants to hide her location information within an area of at least A_{min} and at most A_{max}, respectively. The *location anonymizer* is basically a trusted third party that acts as a middle layer between mobile users and the location-based database server

in order to: (1) receive the exact location information from mobile users along with a *privacy profile* of each user, (2) employ an existing location anonymization technique to blur users' exact locations into *cloaked areas* that satisfy each user's *privacy profile*, (3) send the *cloaked areas* to the database server, and (4) compute the exact answer from a *candidate list* of answers returned by the database server and send the exact answer to the user.

In this paper, we go beyond the location anonymization problem as we address the challenging problem of providing *snapshot* and *continuous* LBS even when receiving the user's blurred location information from the *location anonymizer* rather than the exact locations from system users. Basically, we propose a *snapshot/continuous privacy-aware query processor* that is embedded inside the location-based database server to tune its functionalities to deal with anonymous location-based queries with *cloaked areas* received from the *location anonymizer* rather than the exact location information. Our *privacy-aware query processor* is completely independent of the underlying location anonymization algorithm. Thus, any existing location anonymization technique that cloaks users' locations into rectilinear areas can be employed. The proposed query processor supports three privacy-aware query types: (1) Private queries over public data, e.g., *"Where is my nearest gas station"*, in which the person who issues the query is a private entity while the data (i.e., gas stations) is public, (2) Public queries over private data, e.g., *"How many cars within a certain area"*, in which a public entity asks about personal private locations, and (3) Private queries over private data, e.g., *"Where is my nearest buddy"* in which both the person who issues the query and the requested data are private. With this classification in mind, traditional location-based query processors can support only public queries over public data. This query classification is applicable regardless of having the underlying query as *snapshot* or *continuous*.

Due to the lack of exact location information on the server side, the proposed *privacy-aware query processor* provides a *candidate list* of answers instead of an exact answer. We prove that the candidate list is *inclusive*, i.e., contains the exact answer, and is *minimal*, i.e., given certain conditions, the candidate list is of minimal size. In addition, our proposed query processor can be tuned through a tuning parameter to provide a trade-off between query processing cost and answer optimality, i.e., the candidate list size. For *continuous* location-based queries, we propose a *shared execution paradigm* that enables the *privacy-aware query processor* to scale to a large number of concurrent continuous queries. The *shared execution paradigm* maintains the answer of a set of selected static continuous queries, and the answer is shared by all outstanding continuous queries. The proposed *shared execution paradigm* provides two other tuning parameters to achieve a trade-off between system scalability and answer optimality. In general, the contributions of this paper can be summarized as follows:

—We introduce a system framework that allows mobile users to anonymously obtain snapshot and continuous location-based services by specifying their privacy requirements through a user *privacy profile*.

—We identify three new privacy-aware query types that are not supported by existing location-based database servers, namely, *private queries over public data*, *public queries over private data*, and *private queries over private data*.

—We introduce a *privacy-aware query processor* that provides a unified framework to support all introduced privacy-aware query types. We prove that our query processor provides an *inclusive* and *minimal candidate list* of answers. In addition, the performance of the query processor can be tuned through a parameter to achieve a trade-off between query processing cost and answer optimality.

—We introduce a *shared execution paradigm* that shares query processing among a large number of continuous privacy-aware queries for all introduced query types. Such a scalable paradigm can be tuned through two other parameters to trade off between system scalability and query answer optimality.

—We provide experimental evidence that our *privacy-aware query processor* is *efficient* in terms of query processing time, is *scalable* in terms of supporting large numbers of users and snapshot/continuous queries, and is *privacy-aware* as it provides high-quality answers without the need for exact location information.

The rest of the paper is organized as follows. Section 2 highlights the related work to the proposed privacy-aware query processing framework. The underlying architecture is outlined in Section 3. The *snapshot* and *continuous privacy-aware query processors* are described in Sections 4 and 5, respectively. Extensive experimental evaluation of our privacy-aware query processor is presented in Section 6. Finally, Section 7 concludes the paper.

2. RELATED WORKS

In this section, we highlight the related work to the proposed privacy-aware query processing framework in four different areas, namely, location privacy, location-based query processing, privacy models, and privacy-aware query processing.

2.1 Location Privacy

Motivated by the privacy threats of location-detection devices [Ackerman et al. 2003; Barkhuus and Dey 2003; Beresford and Stajano 2003; Warrior et al. 2003], recent attempts for providing location privacy in location-based services (LBS) (e.g., [Bamba et al. 2008; Beresford and Stajano 2003; Chow et al. 2006; Chow and Mokbel 2007; Cheng et al. 2006; Duckham and Kulik 2005; Gedik and Liu 2008; Gruteser and Grunwald 2003; Ghinita et al. 2007a; 2007b; Gruteser and Liu 2004; Hashem and Kulik 2007; Hengartner and Steenkiste 2003; Hong and Landay 2004; Kalnis et al. 2007; Kido et al. 2005; Li et al. 2008; Xu and Cai 2007; 2008]) and other location-aware applications (e.g., context-aware computing [Smailagic and Kogan 2002] and sensor networks [Gruteser et al. 2003]) focus only on the *location anonymizer* part. Although such techniques would be valuable for protecting users' private locations in LBS, the practicality in real location-based database servers is doubtful as these techniques lack privacy-aware query processing capacity. By protecting users' location information from being disclosed to the location-based database server, processing these location privacy-preserving queries becomes challenging where new techniques need to be presented to provide efficient query processing while not being able to know exact users' locations.

In general, four different approaches have been explored: (1) False dummies [Kido et al. 2005]. For every location update, a user sends n different locations to the server with only one of them is true while the rest are dummies. Thus, the server

cannot know which one of these locations is the actual one. (2) Landmark objects [Hong and Landay 2004]. Rather than sending the exact location to a location-based database server, the user refers to the location of a certain landmark or a significant object. (3) Location perturbation [Bamba et al. 2008; Chow et al. 2006; Chow and Mokbel 2007; Cheng et al. 2006; Duckham and Kulik 2005; Gedik and Liu 2008; Gruteser and Grunwald 2003; Ghinita et al. 2007a; 2007b; Gruteser and Liu 2004; Hashem and Kulik 2007; Kalnis et al. 2007; Li et al. 2008; Xu and Cai 2007; 2008]. The main idea is to blur a user's exact location into a spatial area using either spatial or temporal cloaking [Bamba et al. 2008; Chow et al. 2006; Chow and Mokbel 2007; Gedik and Liu 2008; Gruteser and Grunwald 2003; Ghinita et al. 2007a; 2007b; Hashem and Kulik 2007; Kalnis et al. 2007; Li et al. 2008; Xu and Cai 2007; 2008] or location obfuscation [Duckham and Kulik 2005]. The blurred spatial area can be based either on the k-anonymity concept [Samarati 2001; Sweeney 2002a; 2002b] (i.e., the area should contain at least k users) or on a graph model that represents a road network [Duckham and Kulik 2005]. (4) Avoid location tracking [Beresford and Stajano 2003; Gruteser and Liu 2004]. While the previous three approaches focus only on hiding a certain instance of the user location, this approach aims to avoid tracking the user behavior.

Among these location anonymization techniques, our proposed *privacy-aware query processor* supports the location perturbation techniques that blur users' exact locations into rectilinear areas, i.e., *cloaked areas*, as this is the most commonly used form of location anonymization in many various environment settings, e.g., [Bamba et al. 2008; Chow et al. 2006; Gedik and Liu 2008; Gruteser and Grunwald 2003; Ghinita et al. 2007a; 2007b; Hashem and Kulik 2007; Kalnis et al. 2007] for snapshot locations, [Chow and Mokbel 2007; Cheng et al. 2006; Xu and Cai 2007; 2008] for continuous locations, [Xu and Cai 2008] for spatial networks, and [Chow et al. 2008; Gruteser et al. 2003] for wireless sensor networks.

2.2 Location-based Query Processing

There has been a plethora of techniques to deal with various *snapshot* location-based queries (e.g., [Hadjieleftheriou et al. 2005; Lin and Su 2005; Papadias et al. 2004; Sun et al. 2004; Tao and Papadias 2005; Tao et al. 2003; Wolfson et al. 2000]) and *continuous* location-based queries (e.g., [Gedik and Liu 2004; Hu et al. 2005; Iwerks et al. 2003; Kolahdouzan and Shahabi 2005; Lazaridis et al. 2002; Mouratidis et al. 2005; Mokbel et al. 2004; Zhang et al. 2003]). The main idea of *snapshot queries* is to provide an efficient and real-time execution of location-based queries using spatio-temporal index structures for frequently updated data. On the other hand, query processors for *continuous* location-based queries have mainly focused on efficiency and scalability. In terms of efficiency, several techniques have been proposed to use grid-based structures to support location-based services for moving data and moving queries (e.g., [Hu et al. 2005; Iwerks et al. 2003; Mouratidis et al. 2005; Mouratidis et al. 2006; Mokbel et al. 2004]). In terms of scalability, several techniques have proposed to employ a shared execution paradigm in which multiple concurrent continuous queries can be evaluated simultaneously at the location-based database server (e.g., [Cai et al. 2004; Gedik and Liu 2004; Mokbel et al. 2004; Prabhakar et al. 2002]). However, all these query processors for *snapshot* and *continuous* location-based queries rely on the knowledge of the exact user locations

as none of these techniques have considered private data and/or private queries.

2.3 Privacy Models

During the last decade, several paradigms of architecture have been explored to provide secure data transformation from the client to the server machines. Secure-multi-party communication [Du and Atallah 2001; Haas et al. 1999] organizes the communication among m parties such that each party can have the knowledge of only a certain function but not the actual data for other parties. However, the computational overhead of such a scheme prevents its direct application to database problems. Thus, the minimal information sharing [Agrawal et al. 2003] paradigm is proposed where it uses cryptographic techniques to perform join and intersection operations. However, the computational cost and the inability to serve other queries make such a paradigm not suitable for real time applications. The untrustworthy third party [Emekci et al. 2006] paradigm has been proposed in the context of peer-to-peer systems. The main idea is to employ a third party that executes queries by collecting secure information from multiple data sources, i.e., peers. The most commonly used model is the trusted third party [Aggarwal et al. 2004; Jefferies et al. 1995] paradigm. The main idea is to employ a third party that is trusted by the users and acts as a middle layer between the user and the database server.

Among these models, our framework employs the trusted third party model as it requires less computational overhead and is more suitable for real-time query processing. The trusted third party model is already utilized by existing location privacy techniques (e.g., [Beresford and Stajano 2003; Bamba et al. 2008; Chow et al. 2006; Chow and Mokbel 2007; Gedik and Liu 2008; Gruteser and Grunwald 2003; Hashem and Kulik 2007; Kalnis et al. 2007; Li et al. 2008; Mokbel et al. 2006; Xu and Cai 2007; 2008]) and is commercially applied in other fields. For example, the Anonymizer [Anonymizer 2008] is for anonymous web surfing while the PayPal [Paypal 2008] system is a trusted third party where a user can buy products without giving her credit card information to the provider.

2.4 Privacy-Aware Query Processing

Recent research efforts have been dedicated to deal with location privacy-preserving queries, i.e., getting anonymous services from location-based applications (e.g., [Cheng et al. 2006; Ghinita et al. 2008; Kalnis et al. 2007; Khoshgozaran and Shahabi 2007; Hu and Lee 2006; Mokbel et al. 2006; Yiu et al. 2008]). These query processing frameworks can be divided into three main categories. (1) Location obstruction [Yiu et al. 2008]. The basic idea is that a querying user first sends a query along with a false location to a database server, and the database server keeps sending the list of nearest objects to the reported false location to her until the list of received objects satisfies the user's privacy and quality requirements. (2) Space transformation [Ghinita et al. 2008; Khoshgozaran and Shahabi 2007]. This approach converts the original location of data and queries into another space through a trusted third party. The space transformation maintains the spatial relationship among the data and query, in order to provide accurate query answers. (3) Cloaked area processing [Cheng et al. 2006; Kalnis et al. 2007; Hu and Lee 2006; Mokbel et al. 2006]. In this framework, a *privacy-aware query processor* is embedded in the database server side to deal with the cloaked spatial area received

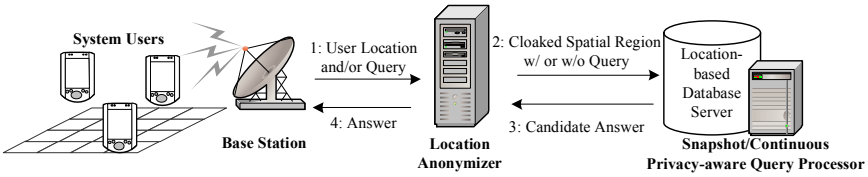


Fig. 1. System architecture

either from a querying user [Cheng et al. 2006; Hu and Lee 2006] or from a trusted third party [Kalnis et al. 2007; Mokbel et al. 2006].

Among the existing cloaked area processing frameworks, the works [Hu and Lee 2006; Kalnis et al. 2007] are closest to ours. These works find the exact set of objects as a query answer based on the linear nearest neighbor search algorithm [Tao et al. 2002]. The difference between these two works is that the work [Hu and Lee 2006] considers rectilinear *cloaked areas* while the other one considers circular *cloaked areas* [Kalnis et al. 2007]. The key distinctions between these two works and our proposed *privacy-aware query processor* are follows: (1) Our query processor has the ability to ease the optimality of query answers by finding a superset of the minimal answer set that contains the exact answer, in order to achieve system scalability. We use a tuning parameter to trade off between system scalability and answer optimality. (2) According to our classification of privacy-aware queries, these two previous works consider only *private queries over public data*, so they cannot be applied to the case of private data. (3) We consider continuous privacy-aware queries by proposing a *shared execution paradigm* that aims to improve system scalability when dealing with a numerous number of privacy-aware continuous queries. The *shared execution paradigm* also provides two other tuning parameters to trade off between system scalability and answer optimality.

3. SYSTEM ARCHITECTURE

Figure 1 depicts the underlying system architecture which has two main components: the *location anonymizer* and the *privacy-aware query processor*.

Location Anonymizer. Mobile users who are willing to share their private location information can register directly with the location-based database server. On the other hand, mobile users who want to protect their private location information should register with the *location anonymizer* by specifying a certain *privacy profile* that outlines their privacy requirements. The *privacy profile* would support the most commonly used privacy requirements, namely, *k-anonymity*, *minimum area* A_{min}, and *maximum area* A_{max}. k-anonymity indicates that the mobile user wants to be k-anonymous, i.e., not distinguishable among k users, while A_{min} and A_{max} ($A_{min} \leq A_{max}$) are the minimum and maximum acceptable size of the *cloaked area*, respectively. A_{min} is particularly useful in a dense area where even a large k would not achieve high privacy protection. For example, a user in a stadium with $k = 100$ may result in a very small *cloaked area*. Similarly, a user in a shopping mall may want to guarantee that her *cloaked* area is beyond the mall boundary. On the other hand, A_{max} indicates the largest amount of spatial inaccuracy of the *cloaked area* the user is willing to tolerate. Larger values for k, A_{min} and A_{max} indicate stricter privacy requirements. Finally, mobile users have the ability to change their *privacy*

profiles at any time to achieve their personal trade-off between privacy requirements and the quality of services.

The *location anonymizer* receives location updates from mobile users, uses a location anonymization algorithm to blur the locations into *cloaked areas* that match each user's *privacy profile*, and sends the *cloaked areas* to the location-based database server. While *cloaking* the location information, the *location anonymizer* also removes any user identity to ensure the pseudonymity of the location information [Pfitzmann and Kohntopp 2000]. Similar to the exact location point, the *location anonymizer* also blurs the query location information before sending a *cloaked query area* to the database server. After the *location anonymizer* gets the *candidate list* of answers from the *privacy-aware query processor*, it computes the exact answer from the *candidate list*, and then sends the exact answer to the user. Since location anonymization algorithms have been widely studied, we will not discuss any specific location anonymization algorithm in this paper. To this end, we propose a *privacy-aware query processor* that is completely independent of the location anonymization algorithm employed by the *location anonymizer*. Note that the privacy requirements defined in a *privacy profile* are based on the employed location anonymization algorithm.

Privacy-aware Query Processor. The *snapshot/continuous privacy-aware query processor* is embedded inside the location-based database server to anonymously deal with *cloaked areas* from the *location anonymizer* rather than exact point locations. Instead of returning an exact answer, the *privacy-aware query processor* returns a *candidate list* of answers in which the exact query answer to the user issuing the query through the *location anonymizer* must be included. Then, the *location anonymizer* filters out the false objects from the *candidate list*, and sends the exact answer to the mobile user. The size of the *candidate list* heavily depends on the user *privacy profile*. A stricter *privacy profile* would result in a larger *candidate list* of answers. Using their *privacy profiles*, mobile users have the ability to adjust a personal trade-off between the amount of information they would like to reveal about their locations and the quality of services that they obtain from our framework. Location-based queries processed at the *privacy-aware query processor* may be received either from the mobile users or from public administrators. Queries that come from mobile users are considered as *private queries*. *Private queries* should be passed by the *location anonymizer* to hide the user identity, and the location of the user who issues the query should be blurred. Location-based queries that are issued from public administrators are considered as *public queries* and do not need to pass through the *location anonymizer*. Instead, they are directly submitted to the database server. The database server will answer such public queries based on the stored blurred location information of all mobile users.

Two types of data are stored in the *privacy-aware* location-based database server, *public* data and *private* data. *Public* data includes stationary objects such as hospitals, restaurants, and gas stations or moving objects such as police cars and on-site workers. These persons and facilities do not want to hide their location information. Thus, they are stored directly in the location-based database server without interference from the *location anonymizer*. *Private* data mainly contains personal information of mobile or stationary users who are not willing to reveal their loca-

tion information, e.g., they specify a *privacy profile* with non-zero k or non-zero A_{min}. Private data is received at the *privacy-aware* location-based database server as *cloaked areas* from the *location anonymizer*. Based on the stored data, we identify three new query types that are supported by our *privacy-aware query processor*:

—***Private queries over public data.*** In this query type, the query location is hidden while the exact location information of stored data is known. For example, a person (private query) asks about her nearest gas station (public data). In this case, the *privacy-aware query processor* does not know the exact location of the person who issues the query while the exact locations of gas stations are known.

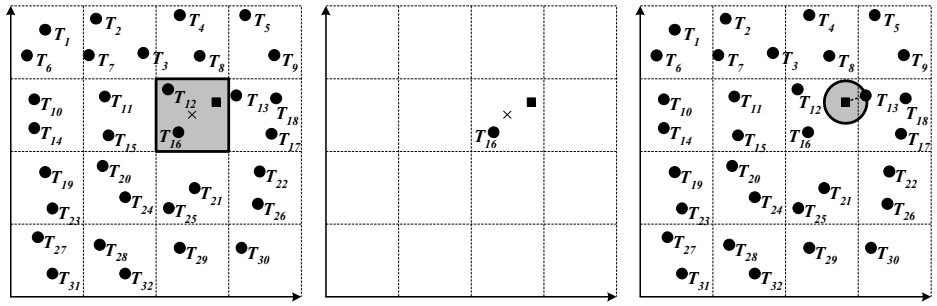
—***Public queries over private data.*** In this query type, the query location is exactly known while the exact locations of the data of interest are not available. Instead, only blurred information is available for the data of interest. For example, an administrator (public query) asks about the number of mobile users (private data) within a certain area. In this case, the *privacy-aware query processor* knows the exact query location information, but it does not know the exact locations of mobile users.

—***Private queries over private data.*** In this query type, neither the query location nor data locations are known. For example, a person (private query) asks about her nearest buddy (private data). The exact locations of both the person who issues the query and her buddies are not available at the *privacy-aware query processor*.

With this classification, traditional location-based database servers (e.g., [Güting et al. 2005; Mokbel et al. 2004; Wolfson et al. 2002]) can support only *public queries over public data* where the exact location information of both data and queries is available.

4. SNAPSHOT PRIVACY-AWARE QUERY PROCESSING

In this section, we present the privacy-aware query processing for snapshot queries. Although previous approaches can be used to compute a minimal *candidate list* of answers for *private queries over public data* [Hu and Lee 2006; Tao et al. 2002], the minimal *candidate list* would be expensive to compute in many cases, e.g., private queries with large cloaked areas and the number of data is very large. On the other hand, our proposed algorithms for privacy-aware query processing provide a distinct feature to compute a superset of the minimal candidate list that contains the exact answer to the user with lower computational cost. We can adjust between computational cost and candidate list size through a tuning parameter *refine*. A larger value of *refine* gives a smaller candidate list, but incurs higher computational cost. When *refine* $= \infty$, our algorithm provides the same minimal *candidate list* as the previous approaches. Furthermore, although these previous approaches give conditions for pruning internal nodes with rectangular regions in an R-tree to obtain a set of candidate objects for query processing [Hu and Lee 2006; Tao et al. 2002], the given conditions are not sufficient for computing a minimal candidate list for *private queries over private data*, and simply returning all such candidate objects would result in a large candidate list that incurs high transmission time. Similar to *private queries over public data*, our algorithm for *private queries over private*



(a) The Server (b) Client: Trivial Approach I (c) Client: Trivial Approach II

Fig. 2. Two trivial approaches for processing private queries over public data

data also supports the tuning parameter *refine*. When $refine = \infty$, our algorithm provides a minimal *candidate list* for private data.

The rest of this section is organized as follows. First, we consider *private queries over public data*. Then, we extend the query processing algorithm to deal with *public queries over private data* and *private queries over private data*.

4.1 Private Queries over Public Data

In this section, we will consider a nearest-neighbor query issued by a user in a form "*What is my nearest gas station*". In this case, the *privacy-aware query processor* does not know the exact location information of the user. Instead, the query processor knows only a cloaked area in which the user resides. On the other hand, the exact location of the gas stations is known. Figure 2a depicts such a scenario by showing the data stored on the server side. There are 32 target objects, i.e., gas stations, T_1 to T_{32} represented by circles. The shaded area represents the cloaked area of the user who issued the query. For clarity, the actual user location is plotted as a square inside the cloaked area, but this information is not revealed to the database server.

Figures 2b and 2c give two trivial approaches that represent two different extremes for evaluating private nearest-neighbor queries over public data. In the first approach (Figure 2b), the server computes the nearest object to the center of the *cloaked area*, i.e., T_{16}, as the query answer. Although this approach minimizes the data transmitted from the server to the client, it gives an inaccurate answer where the actual nearest object to the client is T_{13}. In the second approach (Figure 2c), the server sends all target objects to the client. Then, the client evaluates her query locally to get T_{13} as the query answer. Although this approach provides the exact answer, it is not practical due to the overhead of transmitting large numbers of target objects and the limited processing and storage capabilities on the client side.

Our approach is to design a *privacy-aware query processor* that achieves a compromise between these two extremes. The main idea is to compute a candidate list of answers that includes the exact answer, i.e., the nearest object to the user who issues the query. To guarantee efficiency and enhance utility, the computed candidate list should be of minimal size. In the rest of this section, we will describe our proposed *privacy-aware query processor* for the case of nearest-neighbor queries along with a detailed example. Then, we will prove that the candidate list produced

Algorithm 1 Private NN Queries over Public Data

```
1: function PrivateNNPublicData(CloakedArea A, Int refine)
2: for each vertex v_i of A
3:    t_i ← the nearest object to v_i
4: candidate_list ← {∅}; R ← {A}
5: for each edge e_ij = v_i v_j of region A do
6:    if t_i = t_j then
7:       candidate_list ← candidate_list ∪ {t_i}
8:    else
9:       R = R ∪ RecursiveRefinement(v_i v_j, t_i, t_j, refine, candidate_list)
10:   end if
11: end for
12: for each range search area R ∈ R
13:    candidate_list ← candidate_list ∪ {all target objects within R}
14: return candidate_list
```

by our algorithm does include the exact answer and of minimal size.

4.1.1 *Algorithm for Nearest-Neighbor Queries.* Algorithm 1 gives the pseudo code for private nearest-neighbor queries over public data. The inputs to Algorithm 1 are: (a) the *cloaked area A* that is received from the *location anonymizer* and (b) a tuning parameter, termed *refine*. A larger value of *refine* requires higher computational cost, yet it gives a smaller candidate list that reduces both the transmission time of sending the candidate list from the database server to the *location anonymizer* and the processing time of computing an exact answer from the candidate list at the *location anonymizer*. Setting *refine* to *zero* would result in a similar, yet better, algorithm to [Mokbel et al. 2006] that returns a candidate list with a size equal to or larger than our algorithm. On the other hand, setting *refine* to ∞ would result in a minimal candidate list with the highest computational cost. The output of Algorithm 1 is a candidate list of answers to be sent to the *location anonymizer*. In this section, we consider *refine* as a system specified parameter, and we will describe how to adaptively adjust *refine* to minimize overall response time which includes processing and transmission time. For the ease of description, Figure 3 gives a running example for a private nearest-neighbor query over public data where it presents a zoom view of the shaded area of Figure 2a along with its neighbor cells and the tuning parameter *refine* is set to one. In general, our algorithm has the following three steps:

STEP 1: *Filter Selection Step.* The main objective of this step is to choose a set of filters that prunes the set of all target objects to a smaller set of objects that includes the exact answer. Basically, for each vertex v_i of the *cloaked area A*, we choose the nearest object of v_i as its filter t_i (Lines 2 to 3 in Algorithm 1). Thus, at most four filters can be chosen. In our example, Figure 3a depicts that the nearest objects for vertices v_1, v_2, v_3, and v_4 are T_{16}, T_{16}, T_{13}, and T_{12}, respectively, i.e., we end up selecting only three objects $\{T_{16}, T_{13}, T_{12}\}$.

STEP 2: *Range Selection Step.* The input to this step is the set of filter objects chosen from the previous step. The output of this step has two components: (a) a set of areas, $\mathcal{R}$, encloses target objects that should be considered in the candidate list, and (b) a set of target objects should be included in the *candidate list*. Initially, we add the *cloaked area A* to the set of areas $\mathcal{R}$, and the candidate list

Algorithm 2 Private NN Queries over Public Data: Recursive Refinement

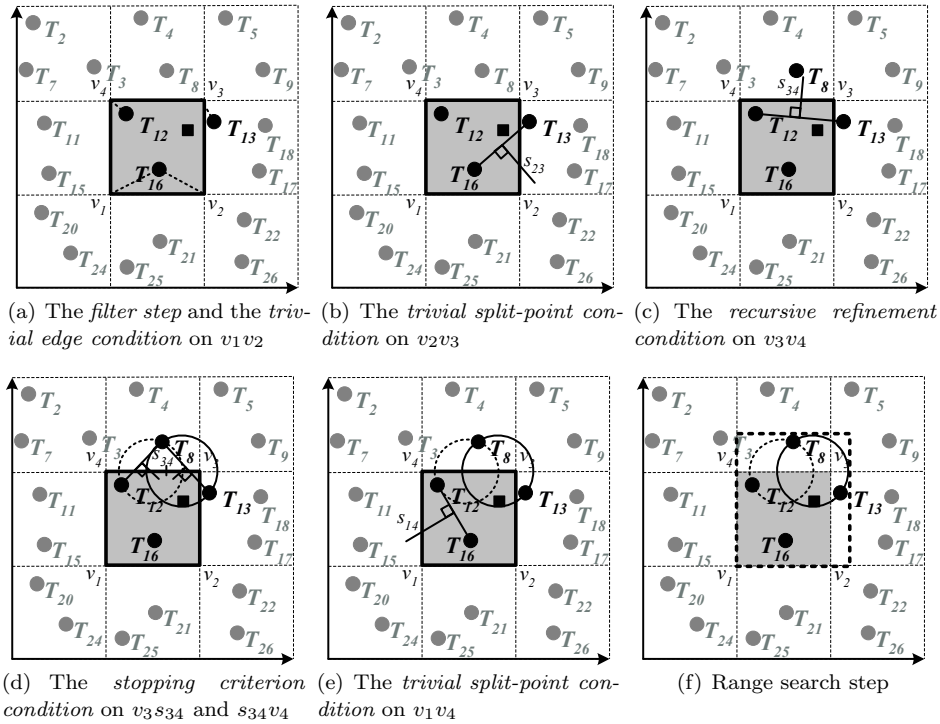
```
1: function RecursiveRefinement(Edge e_ij = v_i v_j, Obj t_i, Obj t_j, Int refine, Set candidate_list)
2: s_ij ← the intersection point of e_ij and the perpendicular bisector (⊥) of t_i and t_j
3: if refine > 0 then
4:    t_s ← the nearest object to s_ij
5:    if t_s = t_i or t_s = t_j then
6:       candidate_list ← candidate_list ∪ {t_i, t_j}
7:       return {∅}
8:    else
9:       refine ← refine − 1
10:      search_area ← search_area ∪ RecursiveRefinement(v_i s_ij, t_i, t_s, refine, candidate_list)
11:      search_area ← search_area ∪ RecursiveRefinement(s_ij v_j, t_s, t_j, refine, candidate_list)
12:   end if
13: else
14:   search_area ← a circle centered at s_ij of a radius dist(s_ij, t_i)
15:   return search_area
16: end if
```

is set to be empty (Line 4 in Algorithm 1). The initialization of $\mathcal{R}$ to A indicates that the object within A should be considered in the *candidate list* as the actual user object could be anywhere in A.

In this step, we deal with each edge $e_{ij} = v_i v_j$ of the *cloaked area* A separately. Based on the two filters t_i and t_j of the edge $v_i v_j$ and the tuning parameter *refine*, we have one of following four possibilities:

(1) *The trivial edge condition* ($t_i = t_j$). We first check for the case that $t_i = t_j$, i.e., one object serves as the nearest object for both vertices v_i and v_j. If this is the case, we just add t_i to the candidate list (Lines 6 to 7 in Algorithm 1). Since we guarantee that t_i is the nearest object to any point on $v_i v_j$, we do not need to perform any additional search on $v_i v_j$, i.e., we do not need to consider any further steps for this edge. In our example (Figure 3a), this case is applied to edge $v_1 v_2$ where $t_1 = t_2 = T_{16}$. In this case, we just add T_{16} to the candidate list as we guarantee that T_{16} is the nearest object to the user if the user location is anywhere on edge $v_1 v_2$.

(2) *The trivial split-point condition* ($t_i \neq t_j$, *refine* > 0, $t_s = t_i$). In the case that $t_i \neq t_j$, (i.e., the two vertices v_i and v_j have different filters), we will use Algorithm 2 to process the edge. Basically, we compute the split point s_{ij} of the edge $v_i v_j$ as the intersection point of $v_i v_j$ and the perpendicular bisector of t_i and t_j where $\mathsf{dist}(s_{ij}, t_i) = \mathsf{dist}(s_{ij}, t_j)$ (Line 2 in Algorithm 2). Since the *refine* parameter is greater than zero, we go ahead and find the nearest object t_s to the split point s_{ij}. If it ends up that $t_s = t_i$, we add both t_i and t_j to the candidate list and return an empty *range search area* (Lines 6 to 7 in Algorithm 2). It is important to note that if $t_s = t_i$, then $t_s = t_j$ as $\mathsf{dist}(s_{ij}, t_i) = \mathsf{dist}(s_{ij}, t_j)$. The idea behind returning the empty *range search area* is that if $t_s = t_i$, then we guarantee that t_i is the nearest object of any point on line segment $v_i s_{ij}$, while t_j is the nearest object of any point on line segment $s_{ij} v_j$. This means that there is no need to have more search on the edge $v_i v_j$. In our example, this case takes place for two edges, $v_2 v_3$ and $v_1 v_4$.



(a) The *filter step* and the *trivial edge condition* on v_1v_2 (b) The *trivial split-point condition* on v_2v_3 (c) The *recursive refinement condition* on v_3v_4

(d) The *stopping criterion condition* on v_3s_{34} and $s_{34}v_4$ (e) The *trivial split-point condition* on v_1v_4 (f) Range search step

Fig. 3. Example of a private nearest-neighbor query over public data (*refine* = 1)

For edge v_2v_3, since $t_2 = T_{16} \neq t_3 = T_{13}$, we compute the split point s_{23} (Figure 3b). Since the nearest object of s_{23} could be either T_{13} or T_{16}, i.e., both T_{13} and T_{16} are of the same distance from s_{23}, we just add T_{13} and T_{16} to the candidate list. Similarly, for edge v_1v_4, we figure out that both T_{12} and T_{16} could be the nearest object to the split point s_{14} (Figure 3e). Thus, we just add T_{12} to the candidate list because T_{16} is already there.

(3) *The recursive refinement condition* ($t_i \neq t_j$, *refine* > 0, $t_s \neq t_i$). In the case that the nearest object t_s to the split point s_{ij} is different from t_i and t_j, we split the edge v_iv_j into two separate line segments v_is_{ij} and $s_{ij}v_j$. Then, we decrease the tuning parameter *refine* by one while recursively splitting each edge separately until we either: (a) the tuning parameter *refine* reaches zero, or (b) we end up at the *trivial split-point condition* (Lines 9 to 11 in Algorithm 2). In our example, edge v_3v_4 depicts this case where $t_3 = T_{13} \neq t_4 = T_{12}$. The nearest object to the split point s_{34} is T_8 (Figure 3c). Since T_8 is different from T_{12} and T_{13} and the *refine* parameter is one, we decrease *refine* by one to zero while dividing the edge v_3v_4 into two separate segments v_3s_{34} and $s_{34}v_4$ (Figure 3d). Since *refine* reaches zero for the two separate line segments, we end up with the next case of the *stopping criterion condition*.

(4) *The stopping criterion condition* ($t_i \neq t_j$, *refine* $= 0$). Once the tuning parameter *refine* reaches zero, we terminate our algorithm by returning a *range search area* as a circle centered at the split point s_{ij} with a radius of $\mathsf{dist}(s_{ij}, t_i)$,

i.e., $\mathsf{dist}(s_{ij}, t_i) = \mathsf{dist}(s_{ij}, t_j)$ (Line 14 in Algorithm 2). The idea behind this circular *range search area* is that all target objects within that area could be the answer of some point on $v_i v_j$, so these objects should be added to the candidate list. It is important to note two main issues in this step: (a) We may not reach to this step as repetitive recursive splitting of the edges may always result in either the *trivial edge condition* or the *trivial split-point condition*; and (b) Based on the number of recursive calls and the tuning parameter *refine*, we may end up with a large number of circular *range search areas* that include target objects to be added to the *candidate list*. In our running example, we end up having the two separate line segments $v_3 s_{34}$ and $s_{34} v_4$ with *refine* set to zero. Thus, we conclude these segments by returning two circular *range search areas* of their split points. The *range search area* of line segment $v_3 s_{34}$ is represented by a circle, while the *range search area* of $s_{34} v_4$ is represented by a dotted circle, as depicted in Figure 3d.

STEP 3: *The Range Search Step.* In this step, we can have one of two options that either provide a candidate list with the minimal size (minimality will be proved later) or provide a larger candidate list with less computation. In both cases, the candidate list is guaranteed to include the exact answer (inclusion will be proved later). The two options are:

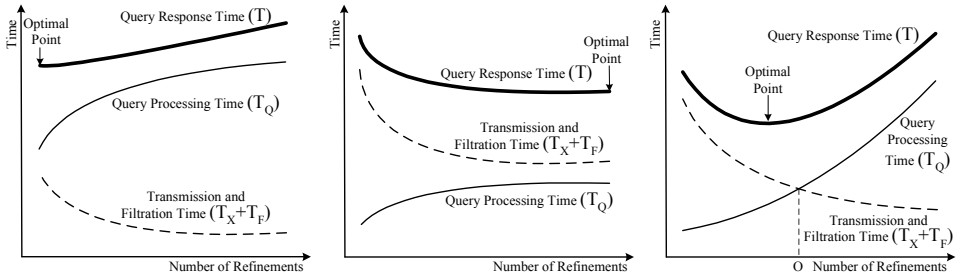
(1) To get the candidate list of the minimal size, we issue a range query for each *range search area* R in the *range query set* $\mathcal{R}$. All the results of these range queries are added to the candidate list (Lines 12 to 13 in Algorithm 1), and the candidate list is sent to the *location anonymizer*. In our example, we will execute three range queries as one range query for the shaded *cloaked area* A and two range queries for the two circles depicted in Figure 3f.

(2) To reduce computational cost while getting a larger candidate list, we execute only one range query that corresponds to the minimum boundary rectangle of all *range search areas* in $\mathcal{R}$. In our example, we will execute only one range query with the minimum bounding rectangle (represented by a bold dotted rectangle) as the query region that covers the circles and the *cloaked area* A (Figure 3f).

In our example, both options give the same *candidate list* (i.e., four target objects T_8, T_{12}, T_{13}, and T_{16}) that contains the actual query answer T_{13}.

Finally, it is important to know that our *privacy-aware query processor* is independent of the underlying nearest-neighbor and range query algorithms used in the nearest-neighbor search for filters or the nearest object of each split point in STEP 1 and STEP 2 and the range search in STEP 3, respectively. These algorithms are assumed to be implemented in traditional location-based database servers. We do not have any assumptions about these algorithms as they can be employed using an R-tree or any other methods. In fact, our approach can be seamlessly integrated with any traditional location-based database servers to turn them to be *privacy-aware*.

4.1.2 *Adaptive Parameter Tuning.* As we have discussed earlier, we can use the tuning parameter *refine* to trade off between system scalability (i.e., computation time) and the query answer optimality (i.e., candidate list size), from a database



(a) $T_Q > T_X + T_F$ (b) $T_X + T_F > T_Q$ (c) As *refine*>O, T_Q>T_X+T_F

Fig. 4. The optimal value of *refine*

server's perspective. On the other hand, from a user's perspective, the most important performance measure is total query response time which includes overall processing and transmission time, as we always guarantee to provide an exact answer within the candidate list. We define query response time T as the sum of three components, (i) the query processing time of computing a candidate list of answers at the database server T_Q, (ii) the transmission time of sending the candidate list from the database server to the *location anonymizer* T_X, and (iii) the filtration time of computing an exact answer from the candidate list at the *location anonymizer* T_F. We know that T_Q is monotonically increasing with respect to the number of refinements, while the candidate list size is monotonically decreasing with respect to the number of refinements. Since T_X and T_F are monotonically decreasing with the decrease of the candidate list size, they are also monotonically decreasing with respect to the number of refinements. With these properties, we can find an optimal value of *refine* which results in the shortest query response time.

Figure 4 depicts three cases for the optimal value of *refine* where the x and y axes represent the value of *refine* and time, respectively, the thin and dotted curves represent the query processing time, i.e., T_Q, and the sum of the transmission time and the filtration time, i.e., $T_X + T_F$, and the bold curve represents the query response time $T = T_Q + T_X + T_F$. Case 1: *T_Q is always higher than $T_X + T_F$, i.e., $T_Q > T_X + T_F$.* In this case, we set *refine* to zero, because any refinement increases the query response time (Figure 4a). Case 2: *$T_X + T_F$ is always higher than T_Q, i.e., $T_X + T_F > T_Q$.* In this case, we set *refine* to a maximum limit, i.e., $refine_{max}$, because minimizing the transmission and filtration time results in the minimum query response time (Figure 4b). Case 3: *There is a point O in which T_Q is always higher than $T_X + T_F$ when refine $> O$, and T_Q is always less than $T_X + T_F$ when refine $< O$.* In this case, we know that there is an optimal point for *refine*, where the optimal point is between zero and ∞, in which the query response time is minimized (Figure 4c).

We will describe an analysis model to find the optimal value of *refine* so that the query response time is minimized. To determine the optimal *refine*, we need to estimate the query response time for the current iteration of refinement $\widehat{T}_i$ and the next iteration of refinement, $\widehat{T}_{i+1}$. Whenever $\widehat{T}_{i+1} > \widehat{T}_i$, we predict that further refinements will increase the query response time. Thus, we stop the query processing and return the current candidate list as the query answer. We first estimate the query response time for the current i-th iteration of refinement, $\widehat{T}_i$, which is the

sum of the following three components:

(1) *Query processing time* (T_{Q_i}). This is the *actual* time incurred in the computation of the current candidate list at the database server.
(2) *Transmission time* (T_{X_i}). The database server will estimate the transmission time of sending the current candidate list to the *location anonymizer* as follows: $T_{X_i} = N_a \times S/B$, where N_a is the number of objects in the current candidate list, S is the average object size, and B is the bandwidth of the communication link between the database server and the *location anonymizer*.
(3) *Filtration time* (T_{F_i}). The database server will also estimate the filtration time of computing an exact answer from the current candidate list at the *location anonymizer* as follows: $T_{F_i} = N_a \times T_f$, where T_f is the average time of computing an exact answer per object from the candidate list at the *location anonymizer*.

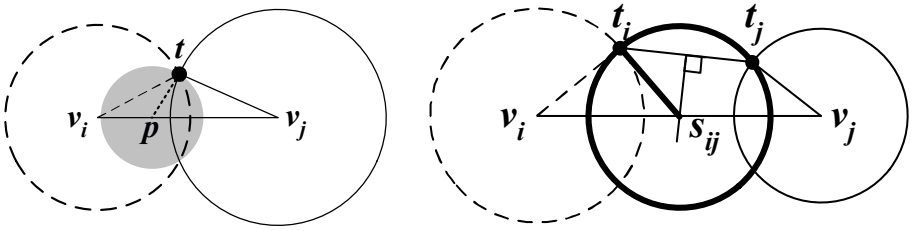
Then, the database server estimates the query response time for the next iteration of refinement, $\widehat{T}_{i+1}$, which is the sum of the following three components:

(1) *Query processing time* ($T_{Q_{i+1}}$). The database server will estimate the query processing time of the next iteration of refinement based on the current one as follows: $T_{Q_{i+1}} = T_{Q_i} + N_o \times T_p$, where N_o is the number of line segments of A that do not meet the *trivial edge condition* or *trivial split-point condition*, and T_p is the average time of computing a bisector and performing a nearest-neighbor search for a split point.
(2) *Transmission time* ($T_{X_{i+1}}$). The database server will estimate the candidate list size for the next iteration of refinement and the transmission time of sending the estimated candidate list to the *location anonymizer* as follows: $T_{X_{i+1}} = (N_a \times R_i \times A_a) \times S/B$, where R_i is the average reduction ratio of N_a from i- to $(i+1)$-th iteration per unit square of a *cloaked area* A and A_a is the area of A.
(3) *Filtration time* ($T_{F_{i+1}}$). The database server will also estimate the filtration time of computing an exact answer from the estimated candidate list at the *location anonymizer* as follows: $T_{F_{i+1}} = (N_a \times R_i \times A_a) \times T_f$.

To estimate T_p and R_i at the database server, and T_f at the *location anonymizer*, we have a learning period with a set of privacy-aware queries, and then update their values during processing newly received queries. Since the tuning parameter *refine* is applicable to both the *private queries over public data* and *private queries over private data*, the proposed adaptive parameter tuning model can be applied to both of these query types.

4.1.3 *Proof of Correctness.* In this section, we show the correctness of the algorithm of *private nearest-neighbor queries over public data* (Algorithm 1) by proving that: (1) **Minimality** - the algorithm is *optimal*, i.e., it returns the minimal candidate list, when the tuning parameter *refine* is set to ∞, and (2) **Inclusion** - the algorithm is *inclusive*, i.e., it returns the exact answer within the candidate list.

THEOREM 1. **Minimality.** *Given a cloaked area A, a user U who issues a query within A, and a tuning parameter refine which is set to ∞, the algorithm computes a minimal candidate list of answers for A.*



(a) Trivial edge condition ($t_i = t_j$) (b) Trivial split-point condition ($t_i \neq t_j$, $t_s = t_i$)

Fig. 5. Two termination cases for the query processing of private queries over public data

PROOF. The algorithm results in a candidate list of objects which reside in A and/or the nearest object of some point on the edge of A. We will show that a minimal candidate list of answers contains all such objects. First, U could be located anywhere within A. An object within A is the nearest object to U, when the object and U are at the same location. Thus, the object within A could be the exact answer to U. Second, U could be located at any point on the edge of A. For each line segment v_iv_j with objects t_i and t_j as the nearest object of its endpoints v_i and v_j, respectively, the algorithm ends up having the line segment with either the *trivial edge condition* (i.e., $t_i = t_j$) or the *trivial split-point condition* (i.e., $t_i \neq t_j$, $t_s = t_i$). This is because the *stopping criterion condition* will not take place as $refine = \infty$. We will show that the algorithm finds the nearest object to any point on the line segment where either the *trivial edge condition* or the *trivial split-point condition* takes place.

(1) *The trivial edge condition.* Figure 5a depicts this case where t is the nearest object to any point p on v_iv_j, the dotted, shaded and solid circles are the required nearest-neighbor search space of v_i, p, and v_j, respectively. The shaded circle touches the intersection points of the dotted and solid circles, and it is totally covered by the dotted and solid circles. Suppose that there is another object t' that is closer to p than t; and hence, t' is within the shaded circle (i.e., t' is within the dotted and/or solid circles). However, if t' is within the dotted circle (or solid circle), it contradicts to the minimality, i.e., t is the nearest object to v_i (or v_j). Therefore, t is the nearest object to any point on v_iv_j.

(2) *The trivial split-point condition.* Figure 5b depicts this case where s_{ij} is the split point of the edge v_iv_j and the nearest object of s_{ij} could be either t_i or t_j. For the line segment v_is_{ij}, v_i and s_{ij} have the same nearest object t_i. The proof of the *trivial edge condition* shows that t_i is the nearest object to any point on v_is_{ij}. Similarly, for the line segment $s_{ij}v_j$, t_j is the nearest object to any point on $s_{ij}v_j$.

Since we guarantee that only the objects within A and the nearest objects to some point on the edge of A are added to the candidate list, the candidate list contains the minimal set of objects that could be the exact answer to U. □

THEOREM 2. **Inclusion.** *Given a cloaked area A, a user U who issues a query within A, and a tuning parameter $refine \geq 0$, the algorithm computes a candidate list of answers for A that contains the exact answer to U.*

PROOF. When *refine* is set to ∞, the proof of Theorem 1 shows that the target objects which could be the nearest object to U are added to the candidate list, so the exact answer to U is included in the candidate list. On the other hand, when *refine* is finite, the *stopping criterion condition* could take place for some line segments. We will show that the exact answer of any point on such line segments is included in the candidate list. When the *stopping criterion condition* is applied to a line segment v_iv_j, we do not find the nearest object to the split point s_{ij} of v_iv_j. Thus, we distinguish two cases.

Case 1: If t_i and t_j are the nearest objects of s_{ij}, the proof of the *trivial split-point condition* in Theorem 1 can be applied to this case. Thus, t_i is the nearest object to any point on the line segment v_is_{ij}, while t_j is the nearest object to any point on the line segment $s_{ij}v_j$. In this case, the algorithm adds the objects within the *range search area* of s_{ij} to the candidate list. Since t_i and t_j are the only objects within the *range search area*, these two objects are added to the candidate list.

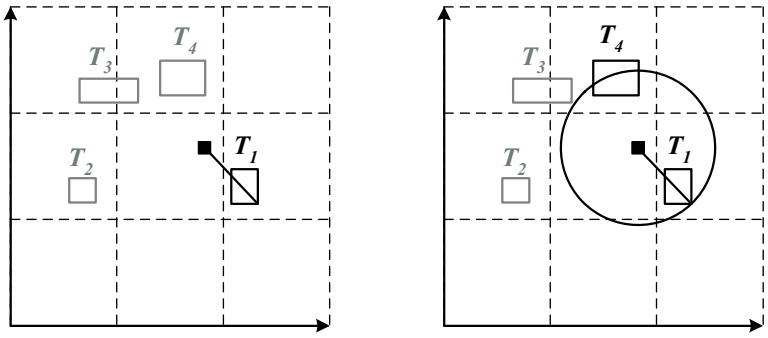
Case 2: If neither t_i nor t_j is the nearest object of s_{ij}. There is an object t that is closer to some point on v_iv_j than t_i and/or t_j. For the line segment v_is_{ij}, if t is closer to some point on v_is_{ij} than t_i, the proof of the *trivial edge condition* in Theorem 1 gives that t must be within the dotted and/or bold circles (Figure 5b). This is because if t is outside the dotted and bold circles, t_i is closer to any point on v_is_{ij} than t. Similarly, for the line segment $s_{ij}v_j$, if t is closer to some point on $s_{ij}v_j$ than t_j, t is within the thin and/or bold circles. Since t_i and t_j are the nearest objects to v_i and v_j, respectively, there is no other objects within the dotted and/or thin circles. Therefore, t must be within the bold circle, i.e., the *range search area* of s_{ij}. In this case, the algorithm adds the objects within the *range search area* of s_{ij} to the candidate list.

For both cases, the nearest object to any point on a line segment where the *stopping criterion condition* takes place is included in the candidate list. Therefore, the exact answer to U is included in the candidate list whenever *refine* ≥ 0. □

4.2 Public Queries over Private Data

In this section, we will consider a public nearest-neighbor query over private data issued by a user in a form "*What is the nearest customer to my taxi*". In this case, the *privacy-aware query processor* is aware of the exact location of the query issuer, i.e., the taxi. However, the query processor does not know the exact location of the data, i.e., customers' locations. Instead, the query processor knows only a *cloaked area* in which each customer resides. The query processor returns a candidate list of answers that includes the exact query answer. The query processing of *public queries over private data* is a very simple one as we are describing it here just as a basis for processing *private queries over private data* in Section 4.3.

4.2.1 *Algorithm for Nearest-Neighbor Queries.* As the idea of the algorithm is very simple, we just show the modifications that we need to have in Algorithm 1 to deal with *public queries over private data*. The input to the algorithm is a point-size area A where the four vertices are the same as the query location point. Since the query location point has no edges, the tuning parameter *refine* takes no effect on the algorithm; and thus, the *range selection step* (STEP 2) in Algorithm 1 is neglected. The other two steps will be slightly modified as follows. Figure 6 acts as a running



(a) The filter selection step (b) The range selection step

Fig. 6. Example of a public nearest-neighbor query over private data

example where the location of the user who issues the query is represented by a square while four private target objects, i.e., T_1 to T_4, are represented by rectangles.

STEP 1: ***The Filter Selection Step.*** This step is similar to the *filter selection step* in Algorithm 1. The only modification is in the nearest object search where we consider that the location of a private object T is its furthest corner from the user's location. In other words, the distance between a user's location U and a private object t with a *cloaked area* A_t, $\mathsf{dist}_{\mathsf{max}}(U, A_t)$, is the distance from U to the furthest corner of A_t from U. Figure 6a depicts that T_1 is a *filter* object and the distance between the user and T_1, i.e., $\mathsf{dist}_{\mathsf{max}}(U, A_{T_1})$, is represented by a line.

STEP 2: ***The Range Search Step.*** In this step, we compute only one circular *range search area* centered at the user's location U with a radius of the maximum distance between U and the *filter* object t, i.e., $\mathsf{dist}_{\mathsf{max}}(U, A_t)$. All objects which intersect the *range search area* could be the exact query answer, so they are added to the candidate list of answers. Figure 6b depicts the *range search area* represented by a circle. The objects which intersect the *range search area*, i.e., T_1 and T_4, are added to the candidate list.

After the *location anonymizer* gets the candidate list of answers from the *privacy-aware query processor*, the exact location of a private object in the candidate list can be disclosed to the user who issues the query, if (1) the user has the required privilege, e.g., police, to access these objects' exact location information; and/or (2) the private object has granted the user the required privilege, e.g., the user is on the object's friend list. However, if the user does not have the required privilege to access the exact location of an object, the user can only get a *cloaked area* as the object's location, in order to preserve the object's location privacy.

4.2.2 *Proof of Correctness.* In this section, we show the correctness of the algorithm of *public nearest-neighbor queries over private data* by proving that: (1) **Minimality** - the algorithm returns the minimal candidate list, and (2) **Inclusion** - the algorithm returns the exact answer within the candidate list.

THEOREM 3. **Minimality.** *Given a user U who issues a query, a filter object t with a cloaked area A_t, and a range search area R centered at U with a radius of the distance between U and the furthest corner of A_t from U, the set of objects which intersects R constitutes a minimal candidate list of answers.*

PROOF. If t is the only object in R, t is the exact answer to U. However, if there is another object t' and its cloaked area $A_{t'}$ intersects R, t' could be the exact answer to U. Since t' intersects R, t' is added to the candidate list. Since only the objects which could be the exact answer are added to the candidate list, the result candidate list is the minimal set of objects that contains the exact answer. □

THEOREM 4. **Inclusion.** *Given a user U who issues a query, a filter object t with a cloaked area A_t, and a range search area R centered at U with a radius of the distance between U and the furthest corner of A_t from U, the exact answer intersects R.*

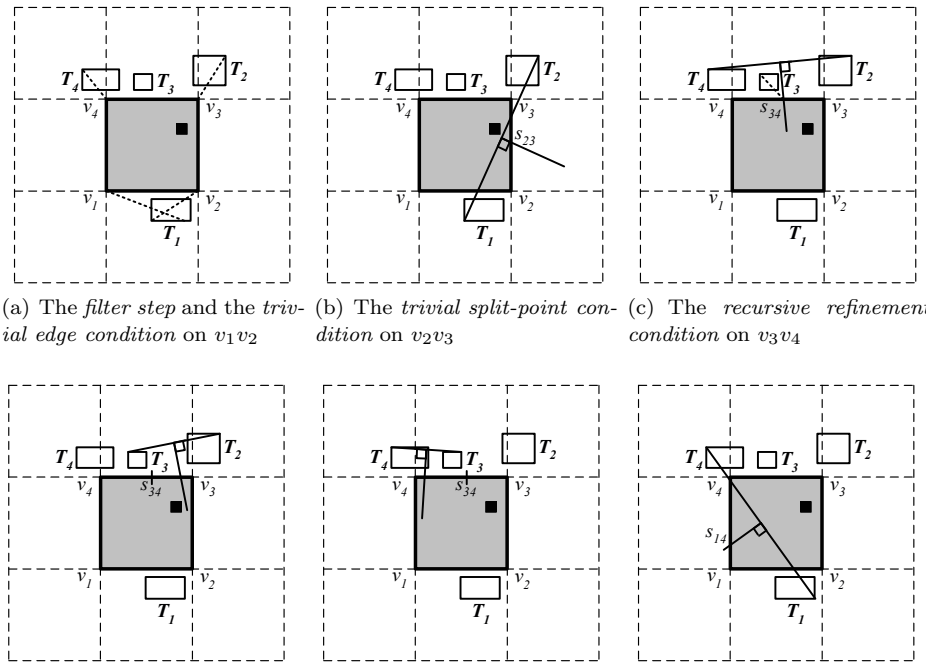
PROOF. We know that the distance between the exact answer and U is not larger than $\mathsf{dist}_{\mathsf{max}}(U, A_t)$, and all objects intersecting R are added to the candidate list. Suppose that t' is the exact answer to U and t' is not included in the candidate list. Thus, we know that the distance between U and t' is larger than $\mathsf{dist}_{\mathsf{max}}(U, A_t)$, i.e., t must be closer to U than t', that contradicts to the assumption that t' is the exact answer. Thus, the candidate list includes the exact answer. □

4.3 Private Queries over Private Data

In this section, we will consider the case of private nearest-neighbor queries over private data in which the query issued by the user is in a form "*What is my nearest buddy*". In this case, the *privacy-aware query processor* does not know the exact location information of both the user who issued the query and the data, i.e., her buddies. Instead, the query processor knows only a *cloaked area* in which the user or each of her buddies resides. The query processor returns a candidate list of answers that includes the exact query answer to the query issuer. The query processing of *private queries over private data* is similar to Algorithm 1, as described in Section 4.1, while using the query processing of *public queries over private data*, as presented in Section 4.2, for nearest neighbor searches.

4.3.1 *Algorithm for Nearest-Neighbor Queries.* As the basic idea of the algorithm is to employ the algorithm of *public queries over private data* for nearest neighbor searches in Algorithm 1. Thus, we show only the modifications that we need to have in Algorithm 1 to deal with *private queries over private data*. The input to the algorithm is exactly the same as Algorithm 1, i.e., the *cloaked area A* received from the *location anonymizer* and a tuning parameter *refine*, where a larger value of *refine* gives a candidate list with a smaller size, but incurs higher query processing time. The output of the algorithm is a *candidate list* of answers to be sent to the *location anonymizer*. Figure 7 depicts a running example for a private nearest-neighbor query over private data where the *cloaked area A* is represented by a shaded area and the private data is represented by rectangles. For clarity, the actual location of the user who issued the query is represented by a square within A, but this information is not revealed to the database server. Also, we show only the private data that is involved in the query processing for the sake of simplicity. In this example, the tuning parameter *refine* is set to one. The algorithm has the same three steps as in Algorithm 1 with following modifications:

STEP 1: ***The Filter Selection Step.*** The only modification in this step is that we use the algorithm of *public queries over private data* to find a filter t_i for



(a) The *filter step* and the *trivial edge condition* on v_1v_2 (b) The *trivial split-point condition* on v_2v_3 (c) The *recursive refinement condition* on v_3v_4

(d) The *stopping criterion condition* on v_3s_{34} (e) The *stopping criterion condition* on $s_{34}v_4$ (f) The *trivial split-point condition* on v_1v_4

Fig. 7. Example of a private nearest-neighbor query over private data (*refine* = 1)

each vertex v_i of the *cloaked area* A. This means that t_i with a *cloaked area* A_{t_i} has the smallest distance $\mathsf{dist}_{\mathsf{max}}(v_i, A_{t_i})$, i.e., the distance between v_i and the furthest corner of A_{t_i} from v_i, among all objects. In our example, Figure 7a depicts that the nearest objects for vertices v_1, v_2, v_3, and v_4 are T_1, T_1, T_2, and T_4, respectively. The distance between each vertex and its filter is represented by a dotted line.

STEP 2: *The Range Selection Step.* The basic idea of this step is similar to the *range selection step* in Algorithm 1, but we have some modifications for each possible condition.

(1) *The trivial edge condition* ($t_i = t_j$). The only modification of this condition is that we add two *range search areas* to the *range query set* $\mathcal{R}$. One *range search area* is a circular region centered at v_i with a radius of $\mathsf{dist}_{\mathsf{max}}(v_i, A_{t_i})$ and the other *range search area* is a circular region centered at v_j with a radius of $\mathsf{dist}_{\mathsf{max}}(v_j, A_{t_j})$. The idea of adding these two *range search areas* to $\mathcal{R}$ instead of adding t_i and t_j to the candidate list is that all objects intersecting these *range search areas* could be the answer of some point on the edge v_iv_j. Thus, these objects should be added to the candidate list. In our example, this condition takes place for edge v_1v_2 where $t_1 = t_2 = T_1$, so we add the *range search areas* of v_1 and v_2 to $\mathcal{R}$.

(2) *The trivial split-pint condition* ($t_i \neq t_j$, *refine* > 0, $t_s = t_i$). We have two modifications for this case. The first modification is in the computation of the split point s_{ij} of the edge v_iv_j. s_{ij} is computed as an intersection point of v_iv_j

and the perpendicular bisector of t_i and t_j in which we consider the furthest corners of t_i and t_j from the opposite vertices v_j and v_i, respectively. Since the tuning parameter *refine* is greater than zero, we find the nearest object t_s to the split point s_{ij}. The second modification is that if it results in a case $t_s = t_i$, we add the *range search areas* of t_i, t_s, and t_j, i.e., these areas are the circles centered at v_i, s_{ij}, and v_j with a radius of $\mathsf{dist}_{\mathsf{max}}(v_i, A_{t_i})$, $\mathsf{dist}_{\mathsf{max}}(s_{ij}, A_{t_i}) = \mathsf{dist}_{\mathsf{max}}(s_{ij}, A_{t_j})$, and $\mathsf{dist}_{\mathsf{max}}(v_j, A_{t_j})$, respectively, to the *range query set* $\mathcal{R}$. The reason of returning these three *range search areas* is that the set of objects intersecting the *range search areas* of v_i and s_{ij} constitutes the minimal set of objects that could be the answer of some point on $v_i s_{ij}$. On the other hand, the set of objects intersecting the *range search areas* of s_{ij} and v_j is the minimal set of objects that could be the answer of some point on $s_{ij} v_j$. In our example, this case is applied to two edges $v_2 v_3$ and $v_1 v_4$. For edge $v_2 v_3$, since $t_2 = T_1 \neq t_3 = T_2$, and the tuning parameter *refine* is larger than zero, we compute the split point s_{23} (Figure 7b). Since the nearest object of s_{23} is T_1 or T_2, i.e., $\mathsf{dist}_{\mathsf{max}}(s_{23}, A_{T_1}) = \mathsf{dist}_{\mathsf{max}}(s_{23}, A_{T_2})$, we add the *range search areas* of v_2, s_{23}, and v_3 to the *range query set* $\mathcal{R}$. Likewise, for edge $v_1 v_4$, we know that the nearest object to the split point s_{14} is T_1 or T_4 (Figure 7f). Since the *range search area* of v_1 is already added to $\mathcal{R}$, we merely add the *range search areas* of s_{14} and v_4 to $\mathcal{R}$.

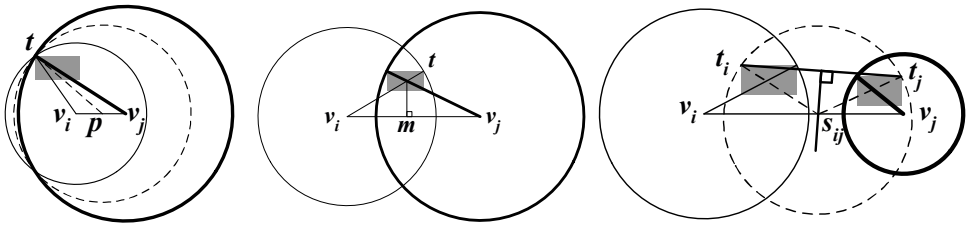
(3) *The recursive refinement condition* ($t_i \neq t_j$, *refine* > 0, $t_s \neq t_i$). This step is exactly the same as the *recursive refinement condition* in Algorithm 1. In our example, this case is applied to edge $v_3 v_4$ where v_3, s_{34}, and v_4 have different nearest objects, i.e., $t_3 = T_2$, $t_{s_{34}} = T_3$, and $t_4 = T_4$, and *refine* is larger than zero. We split $v_3 v_4$ into two separate line segments $v_3 s_{34}$ and $s_{34} v_4$, and then recursively execute this step on these two separate line segments while decreasing *refine* by one to zero, as illustrated in Figures 7d and 7e, respectively. In this example, *refine* reaches zero after the first recursive refinement for both line segments, we end up with the next case of the *stopping criterion condition*.

(4) *The stopping criterion condition* ($t_i \neq t_j$, *refine* $= 0$). The only modification of this step is the same as the second modification of the *trivial split-point condition*. When the tuning parameter *refine* reaches zero, we add the *range search areas* of the endpoints v_i and v_j, and the split point s_{ij} of the edge $v_i v_j$ to the *range query set* $\mathcal{R}$.

STEP 3: ***The Range Search Step.*** The basic idea of this step is the same as in Algorithm 1. The only modification for each option is follows:

(1) To get the candidate list of the minimal size, we issue a range query for each *range search area* R in the *range query set* $\mathcal{R}$. For each range query, all objects intersecting R are returned as the answer. The answers of these range queries are added to the candidate list.

(2) To reduce computational cost while getting a larger candidate list, we execute only one range query with a query region that corresponds to the minimum boundary rectangle of all *range search areas* in $\mathcal{R}$. Then, all objects intersecting the query region are added to the candidate list.



(a) $t_i = t_j$: same vertex (b) $t_i = t_j$: different vertices (c) $t_i \neq t_j$, $t_s = t_i$ and $t_s = t_j$

Fig. 8. Some termination cases for private queries over private data

4.3.2 *Proof of Correctness.* In this section, we show the correctness of the *private nearest-neighbor queries over private data* algorithm by proving that: (1) **Minimality** - the algorithm is *optimal*, i.e., it returns the minimal *candidate list*, when the tuning parameter *refine* is set to ∞, and (2) **Inclusion** - the algorithm is *inclusive*, i.e., it returns the exact answer within the candidate list.

THEOREM 5. **Minimality.** *Given a cloaked area A, a user U who issues a query within A, and a tuning parameter refine which is set to ∞, the algorithm computes a minimal candidate list of answers for A.*

PROOF. The algorithm results in a candidate list of objects which intersect with A or could be the nearest object to some point on the edge of A. We will show that the minimal candidate list contains all such objects. First, U could be anywhere within A. The object intersecting A could be the exact answer because the object and U could be at the same location. Second, U could be located at any point on the edge of A. For each line segment v_iv_j with objects t_i and t_j as the nearest object of its endpoints v_i and v_j, respectively, the algorithm ends up having the line segment with either the *trivial edge condition* (i.e., $t_i = t_j$) or the *trivial split-point condition* (i.e., $t_i \neq t_j$, $t_s = t_i$). This is because the *stopping criterion condition* will not take place when *refine* $= \infty$. We will show that the algorithm finds all objects that could be the exact answer of some point on a line segment where either the *trivial edge condition* or the *trivial split-point condition* takes place.

(1) *The trivial edge condition.* We distinguish two cases.
Case 1: *The furthest corner of the cloaked area of the same filter t of v_i and v_j, A_t, from v_i and v_j is the same* (Figure 8a). The *search range area* of any point p on v_iv_j is a circle with a radius of a distance from p to the furthest corner of A_t from p (represented by a dotted circle). By Theorems 3 and 4, the nearest object to p intersects the *range search area* of p. All possible *range search areas* of p are within the *range search areas* of v_i and v_j, so the objects intersecting the *range search area* of v_i and v_j constitute the minimal set of objects that could be the exact answer to p.
Case 2: *The furthest corners of A_t from v_i and v_j are different* (Figure 8b). Let l_i be the line from v_i to the furthest corner of A_t from v_i (represented by a thin line) and l_j be the line from v_j to the furthest corner of A_t from v_j (represented by a bold line). m is a point on v_iv_j projected from the intersection point of l_i and l_j. The point p on the line segment v_im has the same furthest corner, while p on the line segment mv_j also has the same furthest corner. Thus, the first case can be applied to both line segments v_im and mv_j.

(2) *The trivial split-point condition.* We split v_iv_j into two separate line segments v_is_{ij} and $s_{ij}v_j$. Since both the line segments have the same nearest object, as depicted in Figure 8c, the proof of the *trivial edge condition* can be applied to this case.

Since we guarantee that only the objects within A and the objects that could be the exact answer of some point on the edge of A are added to the candidate list, the candidate list is minimal. □

THEOREM 6. **Inclusion.** *Given a cloaked area A, a user U who issues a query within A, and a tuning parameter $refine \geq 0$, the algorithm computes a candidate list of answers that contains the exact answer to U.*

PROOF. When *refine* is set to ∞, the proof of Theorem 5 shows that target objects that could be the nearest object to U are added to the candidate list, so the exact answer to U is included in the candidate list. When *refine* is finite, the *stopping criterion condition* could take place for some line segments. The proof of the *trivial split-point condition* in Theorem 5 shows that the nearest object to any point on the line segment v_iv_j interests the *range search areas* of v_i, v_j, and/or the split point s_{ij} of v_iv_j. Since the *stopping criterion condition* adds these three *range search areas* to the *range query set* $\mathcal{R}$, the algorithm adds all objects intersecting these *range search areas* to the candidate list. This means that the nearest object to any point on a line segment where the *stopping criterion condition* takes place is included in the candidate list. Therefore, the exact answer to U is included in the candidate list whenever the tuning parameter $refine \geq 0$. □

5. CONTINUOUS PRIVACY-AWARE QUERY PROCESSING

In this section, we propose a *shared execution paradigm* that turns the *snapshot* privacy-aware query processor proposed in Section 4 into a scalable and efficient query processor for *continuous* privacy-aware queries. Examples of continuous queries include "*Continuously send e-coupons to the car that is within one mile of my restaurant*" and "*Continuously report my nearest police car*". The user issues a continuous query by registering the query with a database server for a specified period of time. After the user gets an initial query answer from the database server, she is notified with the changes in the query answer. Since a numerous number of *continuous* privacy-aware queries could be lasted for a long time at the database server, the most important challenges for processing such continuous queries are system scalability and computational efficiency. However, a *basic paradigm* that simply extends the *snapshot* privacy-aware query processing algorithm to deal with continuous queries individually is not scalable and efficient. To this end, we propose a *shared execution paradigm* that aims to share computational resources among *continuous* privacy-aware queries, in order to improve system scalability and efficiency.

The rest of this section is organized as follows. First, we use a *basic paradigm* that extends the *snapshot* privacy-aware query processor to deal with *continuous* privacy-aware queries and analyze the computational cost of maintaining their query answers. Then, we give the detail of our proposed *shared execution paradigm*, analyze the computational cost of employing the *shared execution paradigm* to process *continuous* privacy-aware queries, and describe how to modify the *snapshot* privacy-

aware query processing algorithm to incorporate the *shared execution paradigm* for all introduced privacy-aware query types.

5.1 Basic Continuous Privacy-aware Query Processing

This section describes a *basic paradigm* that extends the *snapshot* privacy-aware query processing algorithm proposed in Section 4 to deal with *continuous* privacy-aware queries. The main idea of the *basic paradigm* is that the *snapshot* privacy-aware query processor computes an initial query answer. Due to mobility, the query answer would become stale at any time. Thus, the query processor has to continuously detect the change in the query answer and notify the user with the change immediately. Continuously maintaining the answer of a privacy-aware query needs to detect two cases of changes: (1) The nearest object of the vertex of a *cloaked area* A or the split point of the edge of A changes its location or there is a new nearest object for the vertex or split point, and (2) Some objects move to or out from a *range search area* in the *range query set* $\mathcal{R}$. For the first case of changes, we issue a *continuous nearest-neighbor (NN) query* for each vertex or split point to monitor its nearest object. For the second case of changes, we issue a *continuous range query* for each *range search area* in $\mathcal{R}$ to detect the change of the target objects within the area. We will describe how the query processor maintains the answer of continuous privacy-aware queries. Whenever the query processor is notified with some changes in the *continuous NN query*, the query processor re-evaluates the query answer. Then, we only send the changes in the query answer to the *location anonymizer*, i.e., an incremental query answer update, in order to reduce communication overhead. On the other hand, when we detect changes only in the *continuous range queries*, we simply send the changes in the query answer to the *location anonymizer* without re-evaluating the query answer. If we re-evaluate a query answer due to the change in a *continuous NN queries*, any change in *continue range queries* can be neglected. This is because all *range search areas* will be removed from $\mathcal{R}$ before the re-evaluation.

Analysis. We will study the computational cost of the *basic paradigm* for each privacy-aware query type. In our analytical model, we consider the second option of the *range search step* in Algorithm 1, i.e., only one *continuous range query* is issued to monitor the minimum bounding rectangle of all *range search areas* in the *range query set* $\mathcal{R}$ and let N_Q be the number of continuous privacy-aware queries in the system. Hence, the total number of *continuous range queries* is N_Q. The computational cost of each privacy-aware query type in terms of the number of *continuous NN queries* is analyzed as follows:

—*Private queries over public data.* For each query, we issue one *continuous NN query* to monitor the nearest object of each vertex of the *cloaked area* A, i.e., four *continuous NN queries* for each query. For each edge of A, we have to monitor the nearest object of at most $2^{refine} - 1$ split points of the edge; and hence, we issue at most $4 \times (2^{refine} - 1)$ *continuous NN queries* for all the split points of A. Thus, the computational cost is $O(N_Q \times [4 + 4 \times (2^{refine} - 1)]) = O(N_Q \times 2^{refine+2})$.

—*Public queries over private data.* For each query, we issue only one *continuous NN query* to monitor its filter object, Thus, the computational cost is N_Q.

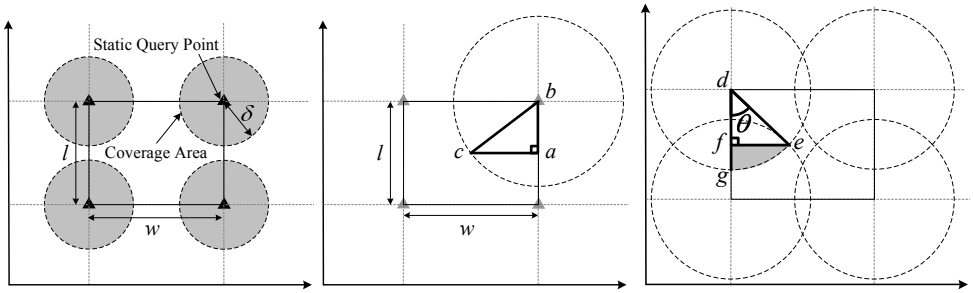
—*Private queries over private data.* The computational cost of this query type is exactly the same as that of the *private queries over public data*, i.e., $O(N_Q \times 2^{refine+2})$.

5.2 Shared Execution Paradigm for Continuous Privacy-aware Query Processing

Although the *basic paradigm* is simple, the computational cost is dependent on the number of continuous privacy-aware queries and the tuning parameter *refine*. The *basic paradigm* would suffer from a scalability issue when a location-based database server processes a numerous number of continuous privacy-aware queries with a strict requirement on the answer optimality, i.e., a large value of *refine*. To this end, we propose a *shared execution paradigm* for continuous privacy-aware queries to improve system scalability and computational efficiency. The basic idea is that we maintain a set of *static query points* whose nearest objects would be utilized as part of the query processing for all continuous privacy-aware queries.

Static Query Points. In the *shared execution paradigm*, we maintain a set of *static query points* that is uniformly distributed in the system. A *static query point* is either in an *on* or *off* state. Initially, all *static query points* are in the *off* state. When the query processor needs to find the nearest object of the vertex of a *cloaked area* A or the split point of the edge of A, it finds the nearest *static query point* of the vertex or split point, and then turns the *static query point* on and finds the nearest object to the *static query point* as an approximate answer for the vertex or split point. We maintain the answer of this *static query point* until it is no longer needed by any continuous privacy-aware queries, i.e., we turn the *static query point* off. Since the answer of a *static query point* may not be the actual nearest object of the vertex of a *cloaked area* A or the split point of the edge of A, using the *static query point* would result in a larger candidate list of answers. Although the number of *static query points* can be served as a system-wide performance tuning parameter, it would not satisfy the need of individual queries. Thus, we introduce another tuning parameter δ for individual continuous privacy-aware queries. The idea is to use the answer of a *static query point* as the nearest object of the vertex or split point of A if the distance between the vertex or split point and its nearest *static query point* is less than δ. When we set $\delta = \delta_{max}$, where δ_{max} is the maximum value of δ, we always use the *static query points* for query processing. At the other extreme case $\delta = 0$, we will not use any *static query points* for query processing.

The *privacy-aware query processor* benefits from the *shared execution paradigm* in three aspects. (1) The number of *static query points* can be served as a performance tuning parameter to trade off between system scalability and query answer optimality. In other words, a larger number of *static query points* gives better quality of query answers, i.e., a smaller candidate list of answers, but it incurs higher computational cost. It is important to note that the *shared execution paradigm* guarantees that the exact query answer is included in the candidate list, regardless of the number of *static query points*. (2) The *shared execution paradigm* provides a fashion to bound the total number of *continuous NN queries* maintained at the database server by the number of *static query points* when $\delta = \delta_{max}$. (3) When we use the answer of a *static query point* as the nearest object of the vertex of a *cloaked area* A or the split point of the edge of A, the answer is immediately available for the query processor without performing any nearest-neighbor search,



(a) Static query points (b) Maximum value of δ (c) Intersection area

Fig. 9. The effect of δ

and thus reducing computational overhead.

Analysis. Table I summarizes the computational cost of the *basic paradigm* and the *shared execution paradigm* with different values of δ for each privacy-aware query type. Similar to the analysis of the *basic paradigm*, we let N_Q be the number of continuous privacy-aware queries and maintain a *continuous range query* for all *range search areas* in the *range query set* $\mathcal{R}$ of each continuous privacy-aware query. Furthermore, we assume that there are N_S *static query points* that are uniformly distributed in the system. Figure 9a depicts a cell (represented by a rectangle) enclosed by four *static query points* (represented by triangles) where the length and width of the cell are l and w, respectively, and the *coverage area* of each *static query point* is represented by a shaded circle. A *coverage area* of a *static query point* is a circular area centered at the query point with a radius of δ. We first develop a function to determine the probability of using the answer of a *static query point* for the vertex of a *cloaked area* A or the split point of the edge of A with respect to δ, $P_A(\delta)$, and then analyze the computational cost of each privacy-aware query type for different values of δ.

Figure 9b gives the case of the maximum value of δ, δ_{max}, i.e., the distance from a *static query point* to the center of the cell. Thus, $\delta_{max} = |\overline{bc}| = \sqrt{|\overline{ab}|^2 + |\overline{ac}|^2} = \sqrt{(l^2 + w^2)/4}$, where c is the center of the cell and $|\overline{bc}|$ represents the distance of line segment $\overline{bc}$. For simplicity, we assume that $P_A(\delta)$ of the vertices and split points of a *cloaked area* A is independent; and thus, $P_A(\delta)$ can be computed as a ratio of the area of the union of the *coverage area* of the four *static query points* of a cell to the cell area. To determine $P_A(\delta)$ for different values of δ, we distinguish three cases:

Case 1: $0 \leq \delta \leq \min(w, l)$. In this case, there is no intersection among the *coverage area* of the four *static query points* of a cell (e.g., see Figure 9a); and hence, $P_A(\delta) = [4 \times (\delta^2\pi \times 90/360)]/(w \times l) = \delta^2\pi/(w \times l)$.

Case 2: $\min(w, l) < \delta < \delta_{max}$. Figure 9c illustrates this case where $|\overline{de}| = |\overline{dg}| = \delta$, $|\overline{fg}| = (2\delta - l)/2 = \delta - l/2$, $|\overline{df}| = |\overline{dg}| - |\overline{fg}| = \delta - (\delta - l/2) = l/2$, $|\overline{ef}| = \sqrt{|\overline{de}|^2 - |\overline{df}|^2} = \sqrt{\delta^2 - (l/2)^2}$, and $\theta = \cos^{-1}(|\overline{df}|/|\overline{de}|) = \cos^{-1}[(l/2)/\delta]$. The area of the shaded area is equals to the area of the triangle EDF subtracted from the area of the sector EDG. The area of the sector EDG is $\delta^2\pi(\theta/360)$ and the area of the triangle EDF is $(|\overline{df}| \times |\overline{ef}|)/2$; and hence, the shaded area is

Table I. The computational cost of each privacy-aware query type

Privacy-aware Query Types	Basic Paradigm	Shared Execution Paradigm ($\delta = \delta_{max}$)	Shared Execution Paradigm ($\delta < \delta_{max}$)
Private Queries over Public Data	$O(N_Q \times 2^{refine+2})$	$O(\min(N_Q \times 2^{refine+2}, N_S))$	$O((1-P_A(\delta)) \times N_Q \times 2^{refine+2} + \min(P_A(\delta) \times N_Q \times 2^{refine+2}, N_S))$
Public Queries over Private Data	N_Q	$O(\min(N_Q, N_S))$	$O((1-P_A(\delta)) \times N_Q + \min(P_A(\delta) \times N_Q, N_S))$
Private Queries over Private Data	$O(N_Q \times 2^{refine+2})$	$O(\min(N_Q \times 2^{refine+2}, N_S))$	$O((1-P_A(\delta)) \times N_Q \times 2^{refine+2} + \min(P_A(\delta) \times N_Q \times 2^{refine+2}, N_S))$

$\delta^2\pi(\theta/360) - (|\overline{df}| \times |\overline{ef}|)/2 = \delta^2\pi(\cos^{-1}[(l/2)/\delta]/360) - [(l/2)\sqrt{\delta^2 - (l/2)^2}]/2$. In general, the intersection area of the *coverage area* of the two *static query points* on the vertical or horizontal edge of a cell is $A_{Int}(s) = 2 \times \{\delta^2\pi(\cos^{-1}[(s/2)/\delta]/360) - [(s/2)\sqrt{\delta^2 - (s/2)^2}]/2\}$, where $s = l$ and $s = w$ for the vertical and horizontal edge of a cell, respectively. Thus, $P_A(\delta) = [\delta^2\pi - 2(A_{Int}(w) + A_{Int}(l))]/(w \times l)$.

Case 3: $\delta \geq \delta_{max}$. In this case, the cell area is totally covered by the *coverage area* of the four *static query points* that enclose the cell; and hence, the query processor always uses the answer of *static query points* for query processing, $P_A(\delta) = 1$.

Therefore, the probability of using the answer of a *static query point* as the nearest object for the vertex or split point of a *cloaked area* A, $P_A(\delta)$, can be summarized as the following equations:

$$P_A(\delta) = \begin{cases} \delta^2\pi/(w \times l), & 0 \leq \delta \leq \min(w, l); \\ \left[\delta^2\pi - 2(A_{Int}(w) + A_{Int}(l))\right]/(w \times l), & \min(w, l) < \delta < \delta_{max}; \\ 1, & \delta_{max} \leq \delta < \infty. \end{cases}$$

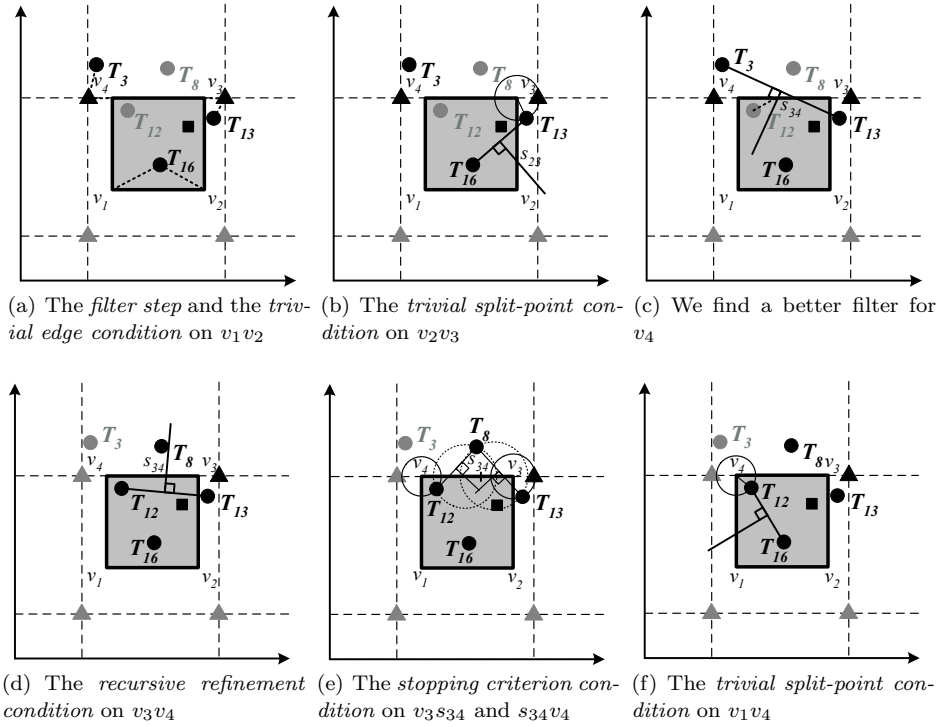
The computational cost of each privacy-aware query type with different values of δ is analyzed as follows:

—*Private queries over public data.* For a set of N_Q queries, we use a total of at most $\min(P_A(\delta) \times N_Q \times 2^{refine+2}, N_S)$ *static query points* for query processing and maintain at most $(1 - P_A(\delta)) \times N_Q \times 2^{refine+2}$ *continuous NN queries*. Thus, the computational cost is $O((1 - P_A(\delta)) \times N_Q \times 2^{refine+2} + \min(P_A(\delta) \times N_Q \times 2^{refine+2}, N_S))$.

—*Public queries over private data.* For a set of N_Q queries, we use a total of at most $\min(P_A(\delta) \times N_Q, N_S)$ *static query points* for query processing and maintain at most $(1 - P_A(\delta)) \times N_Q$ *continuous NN queries*. Thus, the computational cost is $O((1 - P_A(\delta)) \times N_Q + \min(P_A(\delta) \times N_Q, N_S))$.

—*Private queries over private data.* The computational cost of this query type is exactly the same as that of the *private queries over public data*.

5.2.1 *Algorithm for Private Nearest-Neighbor Queries over Public Data.* In this section, we extend Algorithm 1 to employ the *shared execution paradigm* to compute the query answer of a private continuous nearest-neighbor query over public data. Figure 10 depicts a running example of the *shared execution paradigm* for a private



(a) The *filter step* and the *trivial edge condition* on v_1v_2 (b) The *trivial split-point condition* on v_2v_3 (c) We find a better filter for v_4

(d) The *recursive refinement condition* on v_3v_4 (e) The *stopping criterion condition* on v_3s_{34} and $s_{34}v_4$ (f) The *trivial split-point condition* on v_1v_4

Fig. 10. Example of the *shared execution paradigm* for a private continuous nearest-neighbor query over public data (*refine* = 1)

continuous nearest-neighbor query over public data where the *cloaked area* A is represented by a shaded area, the four *static query points* of the cell intersecting A are represented by triangles, and the tuning parameter *refine* is set to one. If a *static query point* has been turned on by the query processor, the *static query point* is represented by a black triangle; otherwise, the *static query point* is represented by a gray triangle. For simplicity, we show only the objects that will be used by the algorithm. The actual location of the user who issued the query is represented by a square for illustration only, but this actual location information is not revealed to the database server. Initially, the candidate list of answers is set to be empty and the *range query set* $\mathcal{R}$ is set to the *cloaked area* A. Then, the same idea of Algorithm 1 can be applied to the *shared execution paradigm* with the following modifications:

STEP 1: ***The Filter Selection Step.*** The only modification is that for each vertex v_i of A, if the distance between v_i and the nearest *static query point* is less than δ, we use the answer of the nearest *static query point* as the filter t_i of v_i. Although the filter may not be the actual nearest object to the vertex, we guarantee that the exact answer to the user is included in the *candidate list* (inclusion will be proved later). Otherwise, we find the actual nearest object of v_i as the filter t_i as in Algorithm 1. Note that if the *static query point* has been turned on, its answer is immediately available for the query processor without performing any nearest-neighbor search. In our example, Figure 10a depicts that only the vertices

v_3 and v_4 with a distance to their nearest *static query point* is less than δ. Thus, the query processor needs to find the actual nearest object to the other vertices v_1 and v_2 as their filters, i.e., $t_1 = t_2 = T_{16}$. Since the distance between v_3 and the top right *static query point* is less than δ, the filter of v_3 is the answer of the top right *static query point*, i.e., $t_3 = T_{13}$. Likewise, the filter of v_4 is the answer of the top left *static query point*, i.e., $t_4 = T_3$.

STEP 2: *The Range Selection Step.* We will present the minor modification for each possible condition.

The trivial edge condition ($t_i = t_j$). This condition takes place for an edge v_iv_j when the same object serves the filter of v_i and v_j. In this condition, the only modification is that if we use the answer of a *static query point* as the filter of v_i, we do not add t_i to the candidate list. Instead, we add a *range search area* of v_i that is a circle centered at v_i with a radius of the distance from v_i to t_i to the *range query set* $\mathcal{R}$. The idea behind this modification is that when we use a *static query point*, its answer may not be the actual nearest object of the vertex. In fact, the objects that could be the actual nearest object of v_i are within the *range search area*. Thus, all these objects should be added to the candidate list. The same scenario is also applied to the vertex v_j. In our example, this condition takes place for edge v_1v_2 because the same object T_{16} serves the filter of both vertices v_1 and v_2 (Figure 10a). Since T_{16} is the actual nearest object to v_1 and v_2, we simply add T_{16} to the candidate list and no further refinement on v_1v_2 is needed.

The trivial split-point condition ($t_i \neq t_j$, *refine* > 0, $t_s = t_i$). We have two modifications before checking for this condition, and one modification when this condition holds for an edge v_iv_j. The first modification is similar to the *filter step*. If the distance between the split point s_{ij} of v_iv_j and the nearest *static query point* is less than δ, we use the answer of the nearest *static query point* as the nearest object t_s of s_{ij}. The second modification is that if t_s is closer to v_i than v_i's current filter t_i, t_s is considered as the filter of v_i. Then, we start over this step on v_iv_j with the new filter. The idea behind this modification is that if we find t_i for v_i based on a *static query point*, t_s could be closer to v_i than t_i, i.e., t_s is a better filter for v_i. The same scenario is also applied to the vertex v_j. We update the filter of v_i and/or v_j whenever we find a better one. The third modification that is similar to the modification of the *trivial edge condition* is for the case that this condition takes place. If we use the *static query point* to find the nearest object of v_i, we add a *range search area* of v_i to the *range query set* $\mathcal{R}$. This is because the objects within *range search area* could be the actual nearest object of v_i. The same scenario is also applied to the vertex v_j and the split point s_{ij} of v_iv_j. In our example, this condition takes place for edges v_2v_3 and v_1v_4. For edge v_2v_3, we compute the split point s_{23} of v_2v_3. Since the distance between s_{23} and the nearest *static query point* is larger than δ, we find the actual nearest object to s_{23}. Figure10b gives that the nearest object of s_{23} is the same as the filter of v_2 and v_3. Since we use only the answer of the top right *static query point* as the filter of v_3, we add the *range search area* of v_3 to the *range query set* $\mathcal{R}$ and T_{13} to the candidate list. Similarly, for edge v_1v_4, the actual nearest object of the split point s_{14} is the same as the filter of v_1 and v_4, as depicted in Figure 10f. As we use the answer of the top left *static query point* for v_4, only the *range search area* of v_4 and T_{16} are added to $\mathcal{R}$ and the

candidate list, respectively.

The recursive refinement condition ($t_i \neq t_j$, *refine* > 0, $t_s \neq t_i$). The query processing of this condition is exactly the same as in Algorithm 1. In our example, this condition takes place on edge v_3v_4 where we compute the split point s_{34} of v_3v_4, and then find the actual nearest object to s_{34}, i.e., the distance between s_{34} and the nearest *static query point* is larger than δ. Figure 10c depicts that the nearest object of s_{34} is T_{12}. Since T_{12} is closer to v_4 than v_4's current filter $t_4 = T_3$, we set T_{12} as v_4's filter, i.e., $t_4 = T_{12}$. If the left top *static query point* is no longer used by any other continuous privacy-aware queries, we turn off this *static query point*. Then, we start over the *range selection step* on v_3v_4 with $t_3 = T_{13}$ and $t_4 = T_{12}$. Since the *trivial edge* and *stopping criterion conditions* still do not take place, we recompute the split point s_{34} and find the actual nearest object to s_{34}, i.e., T_8, as depicted in Figure 10d. T_8 is different from the filters of v_3 and v_4, so this condition takes place for edge v_3v_4 again. We split v_3v_4 into two separate line segments v_3s_{34} and $s_{34}v_4$, and execute the *range selection step* on these two separate line segments while decreasing the tuning parameter *refine* by one to zero. In the recursive calls on these line segments, since *refine* $= 0$, we end up with the next case of the *stopping criterion condition* for these line segments.

The stopping criterion condition ($t_i \neq t_j$, *refine* $= 0$). The only modification for this condition is similar to the *trivial edge condition*. As in Algorithm 1, we first add the *range search area* of the split point s_{ij} of the edge v_iv_j to the *range query set* $\mathcal{R}$. However, if we use the answer of a *static query point* as the nearest object of v_i, we add the *range search area* of v_i to the *range query set* $\mathcal{R}$. The same scenario is also applied to the other vertex v_j. In our example, we end up with this condition for edge v_3v_4, in which we compute the split point of each of the two separate line segments v_3s_{34} and $s_{34}v_4$ without finding the nearest object of these two split points. For the line segment v_3s_{34}, we add the *range search areas* of v_3 and the split point of v_3s_{34} that are represented by solid and dotted circles, respectively, at the right side in Figure 10e to the *range query set* $\mathcal{R}$ and T_8 to the candidate list. The idea of not adding the *search range area* of s_{34} to $\mathcal{R}$ is that T_8 is the actual nearest object to s_{34}, so s_{34}'s *range search area* contains only T_8. Likewise, for the line segment $s_{34}v_4$, we add the *range search areas* of v_4 and the split point of $s_{34}v_4$ to $\mathcal{R}$ and T_8 to the candidate list.

STEP 3: ***The Range Search Step.*** This step is exactly the same as in Algorithm 1. In our example depicted in Figure 10, the examples for the two options are:

(1) We get the minimal candidate list of answers by issuing a *continuous range query* for each *range search area* in the *range query set* $\mathcal{R}$, i.e., the *cloaked area* A and four distinct *range search areas* represented by circles, and then adding the answer to the candidate list.

(2) We issue only one *continuous range query* with a query region that corresponds to a minimum bounding rectangle covering all the *range search areas* in $\mathcal{R}$, and then add the answer to the candidate list.

Proof of Correctness. In this section, we show the correctness of the *shared execution paradigm* for private queries over public data by proving that the paradigm

is *inclusive*, i.e., it returns the exact answer within the *candidate list*.

THEOREM 7. **Inclusion.** *Given a cloaked area A, a user U who issues the query within A, the algorithm of private queries over public data employing the shared execution paradigm computes the candidate list of answers that contains the exact answer to U.*

PROOF. We first show that the exact nearest object t_p to some point p on an edge v_iv_j is included in the candidate list for the *trivial edge*, *trivial split-point*, or *stopping criterion condition*.

The trivial edge condition. By Theorem 1, t_p is within the dotted and/or solid circles (Figure 5a). We distinguish three cases. Case 1: We use the answer of a *static query point*, t, as the nearest object of v_i, while t is the actual nearest object of v_j. Since t is the actual nearest object of v_j, there is no other objects within the solid circle. If the actual nearest object of v_i is t, there is no other objects within the dotted circle; and hence, $t = t_p$. On the other hand, if the actual nearest object of v_i is not t, t_p is closer to p than t, t_p is within the *range search area* of v_i, i.e., the dotted circle. Thus, t_p is included in the candidate list. Case 2: We use the answer of a *static query point*, t, as the nearest object of v_j while t is the actual nearest object of v_i. This case is symmetric with Case 1. Case 3: We use the answer of *static query points*, t, as the nearest object of both v_i and v_j. The proof of Case 1 is applied to the case that t is the actual nearest object of v_i and/or v_j. If t is not the actual nearest object of v_i or v_j, t_p is closer to p than t, t_p is within the *range search area* of v_i and/or v_j. Thus, t_p is included in the candidate list.

The trivial split-point condition. As depicted in Figure 5b, we consider that the edge v_iv_j is split into two separate line segments v_is_{ij} and $s_{ij}v_j$. For both the line segments, the endpoints have the same nearest object, so the proof of the *trivial edge condition* can be applied to these two line segments.

The stopping criterion condition. In this case, the *range search area* R of s_{ij} is added to the *range query set* $\mathcal{R}$, i.e., all objects within R are added to the candidate list. For line segment v_is_{ij}, the proof of the Case 1 and Case 3 of the *trivial edge condition* is applied to the case that we find the actual nearest object of v_i and use the answer of a *static query point* as the nearest object of v_i, respectively. The proof for the line segment $s_{ij}v_j$ is symmetric with that for the line segment v_is_{ij}.

Then, we show that the exact answer to U is within the candidate list. When *refine* is set to ∞, each line segment ends up with having either the *trivial edge condition* or the *trivial split-point condition*. When *refine* is finite, the *stopping criterion condition* could take place for some line segments. We already show that the nearest object to any point on each line segment is added to the candidate list for these three conditions. The objects within A are also added to the candidate list. Therefore, the exact answer to U is included in the candidate list. □

5.2.2 *Algorithms for Public Nearest-Neighbor Queries over Private Data.* The query processing algorithm for *public queries over private data* presented in Section 4.2.1 can be applied to the *shared execution paradigm* with only one modification in the *filter selection step*. The modification is that if the distance between the user who issues the query and the nearest *static query point* is less than δ, we use the answer of the nearest *static query point* as the filter. Otherwise, we find the

actual nearest object of the user as the filter. The *range search step* is the same as in Section 4.2.1.

Proof of Correctness. We will show the correctness of the proposed *shared execution paradigm* for public queries over private data by proving that the paradigm is *inclusive*, i.e., it returns the exact answer within the candidate list.

THEOREM 8. **Inclusion.** *Given a cloaked area A, a user U who issues the query within A, the algorithm of public queries over private data employing the shared execution paradigm computes the candidate list of answers that contains the exact answer to U.*

PROOF. We use the answer of a *static query point*, t, as the nearest object of U. Let t' be the actual nearest object of U. If $t = t'$, the proof is the same as in Theorem 4. However, if $t \neq t'$, $\mathsf{dist}_{\mathsf{max}}(U, A_{t'}) < \mathsf{dist}_{\mathsf{max}}(U, A_t)$, i.e., t' intersects the *range search area* of U. Thus, t' is added to the candidate list. □

5.2.3 *Algorithms for Private Nearest-Neighbor Queries over Private Data.* The *private queries over private data* algorithm presented in Section 4.3.1 only returns *range search areas*, so we have only three minor modifications to apply the *shared execution paradigm* to this algorithm. The modifications are as follows.

STEP 1: *The Filter Selection Step.* The only modification for this step is that for each vertex v_i of a *cloaked area* A, if the distance between v_i and the nearest *static query point* is less than δ, we use the answer of the *static query point* as the filter t_i of v_i. Otherwise, we find the actual nearest object of v_i as its filter t_i.

STEP 2: *The Range Selection Step.* We have only two modifications in the *trivial split-point condition*. These modifications are exactly the same as the first two modifications in the *trivial split-point condition* of the *range selection step* in the *private queries over public data* algorithm (Section 5.2.1).

STEP 3: *Range Search Step.* We do not have any modification in this step.

Proof of Correctness. We will show the correctness of the proposed *shared execution paradigm* for private queries over private data by proving that the paradigm is *inclusive*, i.e., it returns the exact answer within the candidate list.

THEOREM 9. **Inclusion.** *Given a cloaked area A, a user U who issues the query within A, the algorithm of private queries over private data employing the shared execution paradigm computes the candidate list of answers that contains the exact answer to U.*

PROOF. We first show that the exact nearest object t_p to some point p on an edge v_iv_j is included in the candidate list for the *trivial edge*, *trivial split-point*, or *stopping criterion condition*.

The trivial edge condition. We distinguish two cases.

Case 1: *The furthest corner of the cloaked area of the same filter t of v_i and v_j, A_t, from v_i and v_j is the same* (Figure 8a). By Theorem 5, the *cloaked area* of t_p intersects the thin and/or bold circles. We distinguish three cases. Case I: We use the answer of a *static query point*, t, as the nearest object of v_i while the actual nearest object of v_j is t. Since the distance from the actual nearest object of v_i to v_i is either the same as the distance from t to v_i, i.e., t is the actual nearest

object of v_i, or smaller that the distance from t to v_i, so the *cloaked area* of the actual nearest object of v_i is totally covered by the thin circle; and thus, the actual *range search area* of v_i is within the thin and/or bold circles. As the *trivial edge condition* adds both the thin and bold circles to the *range query set* $\mathcal{R}$, all objects within the thin and/or bold circles are added to the candidate list. This means that all objects intersecting the actual *range search area* of v_i will be added to the candidate list. Case II: We use the answer of a *static query point*, t, as the nearest object of v_j while the actual nearest object of v_i is t. This case is symmetric with Case I. Case III: We use the answer of *static query points* as the nearest object of both v_i and v_j. The thin and bold circles are the *range search areas* of v_i and v_j, respectively. Since the actual *range search areas* of v_i and v_j are within the thin and/or bold circles, all objects within these two circles are added to candidate list.

Case 2: *The furthest corners of A_t from v_i and v_j are different* (Figure 8b). The proof of this case is the same as the Case 2 of the *trivial edge condition* in Theorem 5.

The trivial split-point condition. The proof of the *trivial split-point condition* in Theorem 7 is applied to this case.

The stopping criterion condition. The proof of the *stopping criterion condition* in Theorem 7 is applied to this case.

Then, we show that the exact answer to U is within the candidate list. When *refine* is set to ∞, each line segment ends up with having either the *trivial edge condition* or the *trivial split-point condition*. When *refine* is finite, the *stopping criterion condition* could take place for some line segments. We already show that the nearest object to any point on each line segment is added to the candidate list for these three conditions. The objects within A are added to the candidate list. Therefore, the exact answer to U is included in the candidate list. □

6. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of the *basic paradigm* using the *snapshot* privacy-aware query processing algorithm and the *shared execution paradigm* for all privacy-aware query types, i.e., *private queries over public data*, *public queries over private data*, and *private queries over private data*, in our *Casper** framework. Since the *basic paradigm* uses exact nearest-neighbor queries to compute candidate lists, this paradigm is denoted as "Exact" in this section, while the *shared execution paradigm* is denoted as "Shared". We evaluate our algorithms with respect to performance tuning parameters, system scalability, and privacy requirements. In all experiments, the performance evaluation is in terms of (a) *total processing time*, which includes the query processing time of computing an candidate list at the database server, the transmission time of sending the candidate list from the database server to the *location anonymizer*, and the filtration time of computing an exact answer from the candidate list, and (b) *candidate list size*. However, the filtration time is much less than both the query processing time and the transmission time, so we do not show the filtration time in all the figures. For the experiments of performance tuning parameters, we vary the value of *refine*, and the number of *static query points* and the value of δ for the *shared execution paradigm*. These experiments are important to evaluate the performance trade-off of the three tuning parameters. Then, we perform experiments to show the scalability of the

proposed algorithms with respect to large numbers of users and data, and various data object sizes. Finally, we show the performance of the proposed algorithms with respect to various levels of k-anonymity which is the most commonly used privacy requirement.

We use two baseline algorithms to evaluate the performance of our algorithms. For *private queries over public data*, we compare our algorithms with an existing range nearest-neighbor algorithm that finds the minimal set of nearest objects of a rectangular query region [Hu and Lee 2006] (denoted as "RNN"). For private data, since none of existing approaches works for private data, we design a baseline algorithm for *private queries over private data* (denoted as "Base"). The basic idea of the Base algorithm is that we first find the nearest object t to the center of a cloaked area A and determine the maximum distance d_{max} between each vertex v of A and the furthest corner of t from v. Then, we determine a circular range search area centered at the center of A with a radius of the sum of the distance between the center of A and one of its vertex and d_{max}. The set of private data that intersects the range search area constitutes an answer set. Thus, the Base algorithm requires only one nearest neighbor search for t and one range search for the extended A. Since *public queries over private data* is very simple, we do not design any baseline algorithm for this query type. At the *location anonymizer*, we use a nearest neighbor search to compute the exact answer from a candidate list of answers.

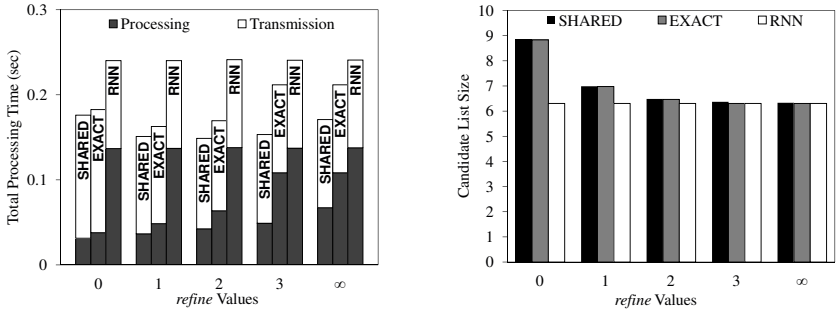
In all experiments, we generate a set of moving objects on the road map of Hennepin County in Minnesota, USA. The input road map is extracted from the Tiger/Line files that are publicly available [Bureau 2006]. Furthermore, the *location anonymizer* employs the *PrivacyGrid* algorithm that is the state-of-the-art location anonymization algorithm to blur users' locations into cloaked areas [Bamba et al. 2008]. These cloaked areas are the input of our *privacy-aware query processor*. Unless mentioned otherwise, the experiment considers 200K mobile users in which 100K users issue continuous privacy-aware queries, and 20K data objects with a size of 2 Kbytes. The continuous queries are simulated similar to the work [Mokbel et al. 2004], but we consider a more dynamic environment. Each moving object or query reports its new location information (if changed) every five seconds. *Casper** is adopted to refresh query results every five seconds. The communication bandwidth between the database server and the *location anonymizer* is 1 Mbits per second (Mbps). We generate a random k-anonymity privacy requirement for each user where k is assigned uniformly within a range $[10 - 50]$. For both the *basic* and *shared execution paradigms*, *refine* is set to one. For the *shared execution paradigm*, we consider $2^{10} \times 2^{10}$ *static query points* and δ is set to 60% of $\delta_{max} = 500$. Table II summarizes the parameter settings.

6.1 Effect of Performance Tuning Parameters

In this section, we study the effect of three performance tuning parameters, i.e., *refine*, the number of *static query points*, and δ, on our proposed *basic paradigm* (Exact) and *shared execution paradigm* (Shared) with respect to *total processing time* and *candidate list size*. In our framework, the tuning parameter *refine* is for both the *basic* and *shared execution paradigms*, while the number of *static query points* and the parameter δ are dedicated for the *shared execution paradigm*.

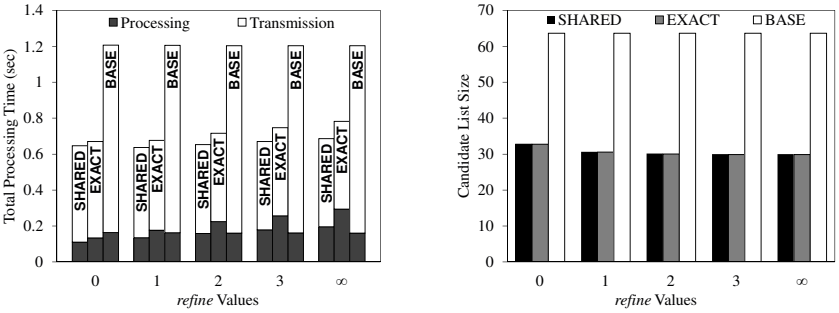
Table II. Summary of parameter settings

Parameters	Default Values	Ranges
Number of users	200K	100K to 500K
Number of data	20K	10K to 50K
Data size	2 Kbytes	2 to 10 Kbytes
Number of static query points	$2^{10} \times 2^{10}$	$2^6 \times 2^6$ to $2^{10} \times 2^{10}$
refine	1	0 to ∞
δ	60% of $\delta_{max} = 500$	20% to 100% of δ_{max}
k-anonymity privacy requirement	[10-50]	[10-50] to [10-250]



(a) Total Processing Time (b) Candidate List Size

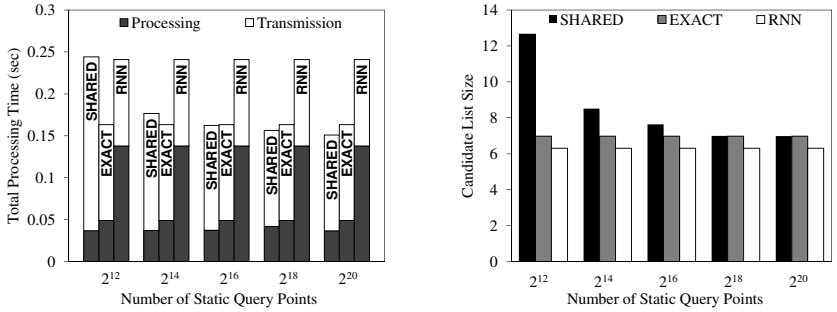
Fig. 11. *refine* Values (Private Queries over Public Data)



(a) Total Processing Time (b) Candidate List Size

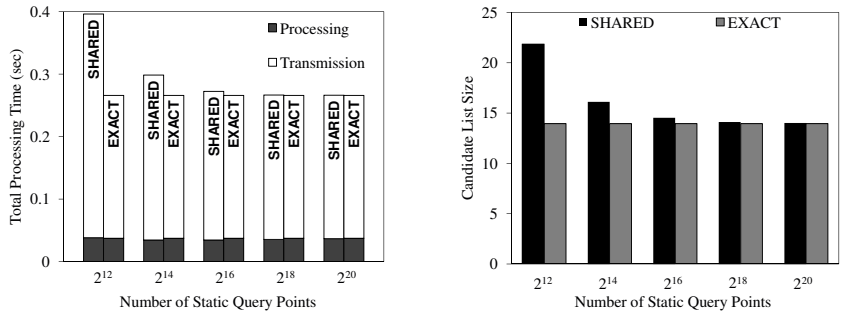
Fig. 12. *refine* Values (Private Queries over Private Data)

Figures 11 and 12 depict the performance of the Shared and Exact algorithms with respect to increasing the value of *refine* from zero to infinity. Figures 11a and 12a give the effect of *refine* on total processing time which includes the *processing time* of a candidate list of answers at the database server (represented by gray bars) and the *transmission time* of sending the candidate list to the location anonymizer (represented by white bars). Since the parameter *refine* has no effect on public queries over private data, we show only the performance of private queries over public and private data. The results show that the query processing time of our Shared and Exact algorithms increases when *refine* gets larger. The results also indicate that Shared effectively improves query processing time, i.e., Shared reduces the query processing time of Exact by from 17% to 55% for private queries over public data and by from 18% to 33% for private queries over private data, as depicted in Figures 11a and 12a, respectively. The main reason of the improvement



(a) Total Processing Time (b) Candidate List Size

Fig. 13. Number of Static Query Points (Private Queries over Public Data)

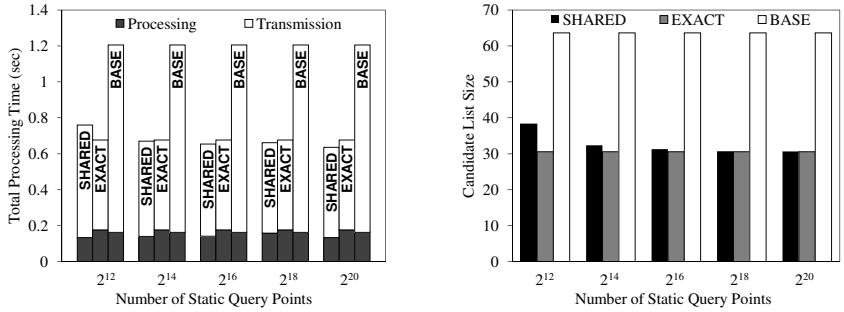


(a) Total Processing Time (b) Candidate List Size

Fig. 14. Number of Static Query Points (Public Queries over Private Data)

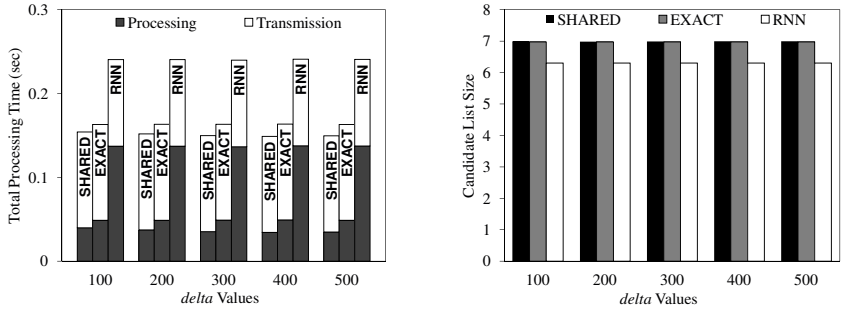
is that Shared significantly reduces the number of nearest-neighbor searches in Exact by sharing the answer of a set of *static query points* as approximate nearest-neighbor searches among all continuous queries. It is important to note that Shared only slightly increases the candidate list size (Figures 11b and 12b). The total processing time of Shared and Exact is better than the baseline algorithms RNN and Base for public and private data, respectively. The total processing time of Shared and Exact increases and the candidate list size slightly reduces when *refine* is larger than one. The main reason is that the first two iterations of refinements already prune the object set to a small set of objects which includes the exact answer to the user. Further refinements can only slight improve transmission time, but they incur higher query processing time. Thus, we set *refine* to one as a default value for other experiments.

Figures 13-15 depict the performance of Shared with respect to increasing the number of *static query points* from 2^{12} to 2^{20}. In general, when the number of *static query points* gets larger, Shared only slightly increases the query processing time (Figures 13a-15a) while significantly improving the candidate list size (Figures 13b-15b). The decrease of the candidate list size is due to the fact that Shared provides more accurate approximate nearest neighbor searches as there are more *static query points*, i.e., the vertex of a cloaked area A or the split point of the edge of A is closer to its nearest *static query point*. As Shared reduces the candidate list size, it also improves the transmission time; and thus, increasing the number of *static query points* results in better total processing time.



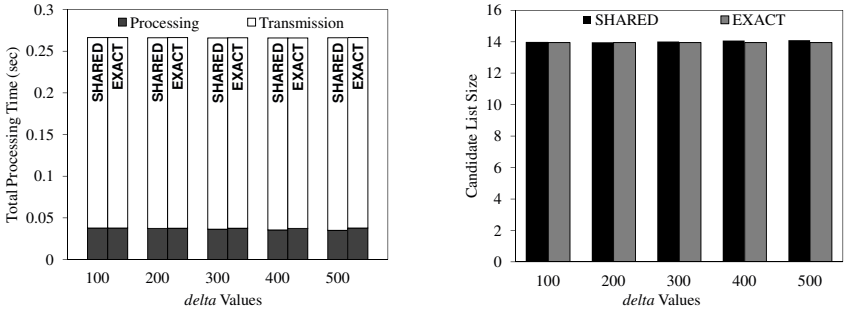
(a) Total Processing Time (b) Candidate List Size

Fig. 15. Number of Static Query Points (Private Queries over Private Data)



(a) Total Processing Time (b) Candidate List Size

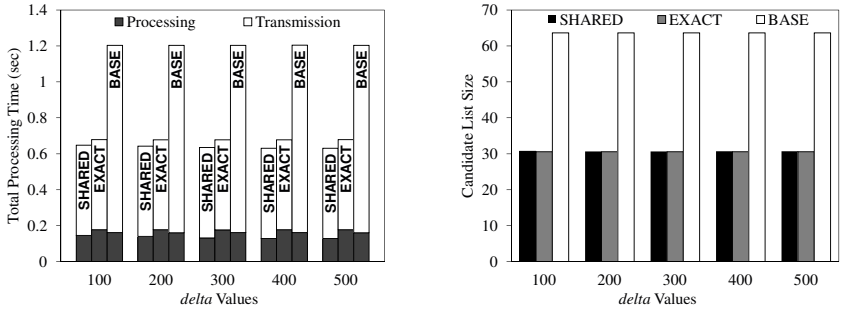
Fig. 16. δ Values (Private Queries over Public Data)



(a) Total Processing Time (b) Candidate List Size

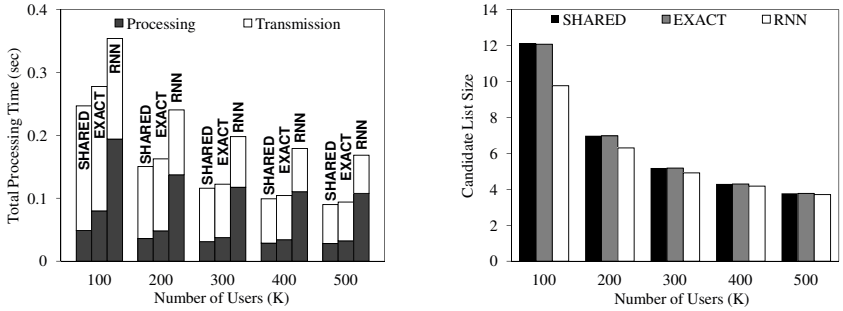
Fig. 17. δ Values (Public Queries over Private Data)

Figures 16-18 give the performance of Shared with respect to increasing δ from 20% to 100% of $\delta_{max} = 500$, i.e., from 100 to 500. In general, the results show that when δ gets larger, the query processing time of Shared decreases (Figures 16a-18a) while the candidate list size slightly increases (Figures 16b-18b). The reason for the improvement in query processing time is that the answer of static query points can be shared with more queries. However, as δ gets larger, Shared provides more inaccurate approximate nearest-neighbor searches for the query processor; and thus, the size of the range search areas in the range query set $\mathcal{R}$ gets larger. Larger range search areas would give larger candidate lists that lead to higher transmission time and filtration time. Therefore, δ can be served as a tuning parameter for a trade-off



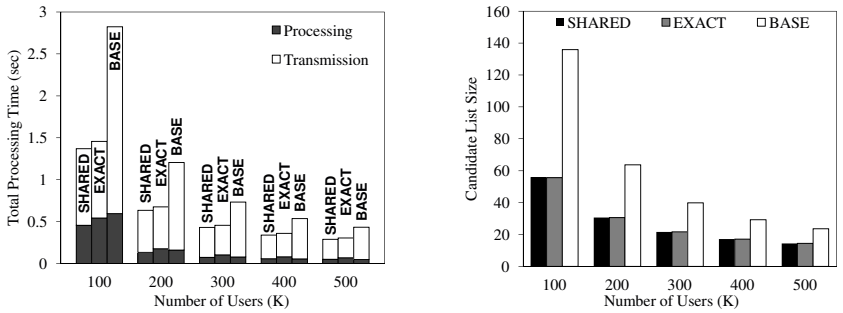
(a) Total Processing Time (b) Candidate List Size

Fig. 18. δ Values (Private Queries over Private Data)



(a) Total Processing Time (b) Candidate List Size

Fig. 19. Number of Users (Private Queries over Public Data)



(a) Total Processing Time (b) Candidate List Size

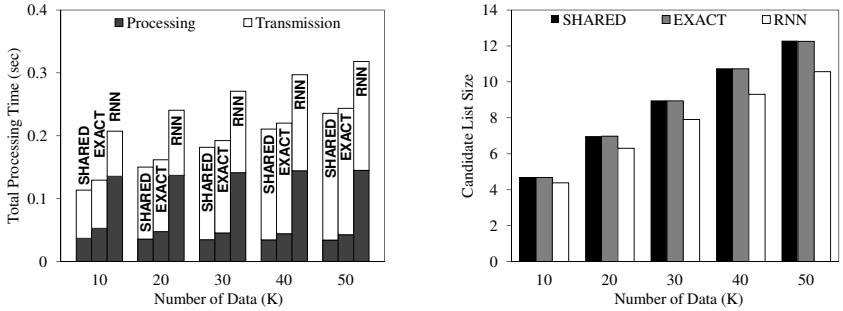
Fig. 20. Number of Users (Private Queries over Private Data)

between query processing time and candidate list size.

6.2 Scalability

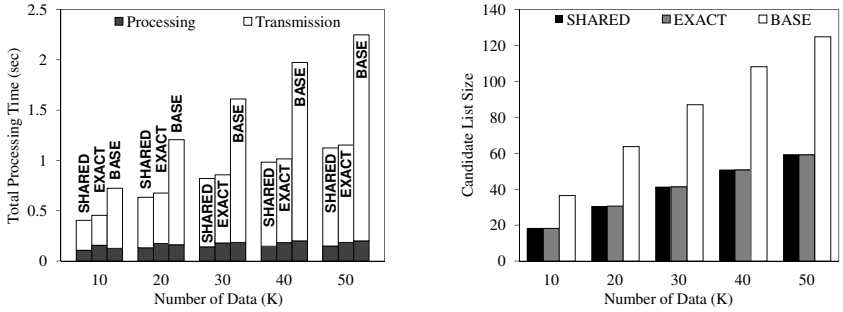
In this section, we evaluate the scalability of our algorithms with respect to three dimensions, i.e., the number of users, the number of data, and data size.

Figures 19 and 20 depict the scalability of our algorithms with respect to varying the number of mobile users from 100K to 500K. As there are more users, the total processing time of all approaches improves (Figures 19a and 20a). The main reasons for this are that (a) increasing the number of users results in smaller cloaked areas that incur lower query processing time; and (2) such smaller cloaked areas lead to



(a) Total Processing Time (b) Candidate List Size

Fig. 21. Number of Data (Private Queries over Public Data)



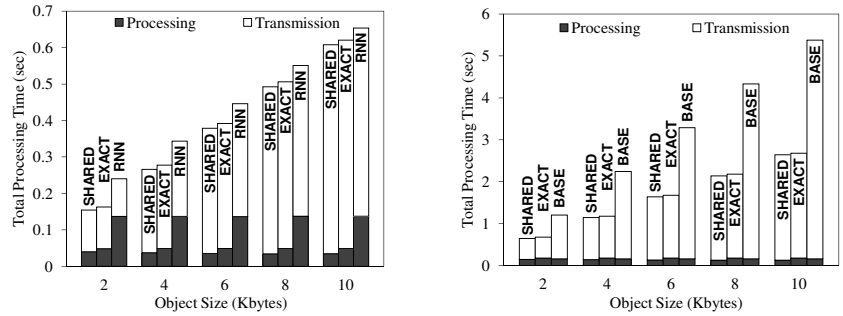
(a) Total Processing Time (b) Candidate List Size

Fig. 22. Number of Data (Private Queries over Private Data)

smaller candidate lists that reduce transmission time (Figures 19b and 20b).

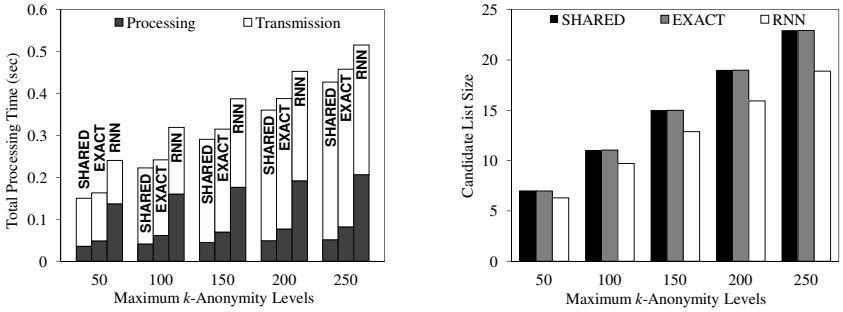
Figures 21 and 22 give the scalability of our algorithms with respect to increasing the number of public/private data from 10K to 50K. The results show that the total processing time and candidate list size of all approaches increase when the number of data gets larger, as depicted in Figures 21a-22a and Figures 21b-22b, respectively. The increase of the total processing time is due to higher transmission time of sending larger candidate lists from the database server to the *location anonymizer*. When the number of data increases, there are more objects within a cloaked area A and each edge of A has more nearest target objects; and hence, the candidate list size increases. Since the transmission time is higher than the query processing time and filtration time, when there are more data, we should increase the value *refine* to reduce candidate list size to improve the total processing time.

Figure 23 gives the effect of data object size on the total processing time of our algorithms with respect to increasing the object size from 2 to 10 Kbytes. Increasing the object size results in higher transmission time for all approaches, so the total processing time of all approaches increases. For public data, since our algorithms improve the query processing time by giving larger candidate lists of answers, i.e., $refine = 1$, the improvement in the total processing time becomes smaller when the object size gets larger (Figure 23a). For private data, our algorithms always (Shared and Exact) give smaller candidate lists than Base, so our algorithms perform much better than Base when the data object size increases (Figure 23b).



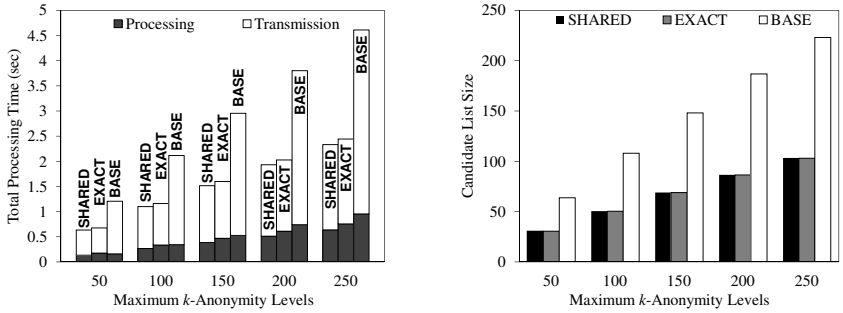
(a) Private Queries over Public Data (b) Private Queries over Private Data

Fig. 23. Data Object Size (Total Processing Time)



(a) Total Processing Time (b) Candidate List Size

Fig. 24. k-Anonymity Requirements (Private Queries over Public Data)



(a) Total Processing Time (b) Candidate List Size

Fig. 25. k-Anonymity Requirements (Private Queries over Private Data)

6.3 Effect of Privacy Requirements

Figures 24 and 25 depict the performance of our algorithms with respect to increasing the maximum k-anonymity level from 50 to 250 (the minimum k-anonymity level is 10). The results show that the query processing time of all approaches increases as k gets larger (Figures 24a and 25a). This is because increasing the k-anonymity level results in larger cloaked areas that lead to higher query processing time. Larger cloaked areas also pose larger candidate lists that lead to higher transmission time (Figures 24b and 25b). Thus, the total processing time of all approaches increases, as the k-anonymity level gets stricter.

7. CONCLUSION

This paper introduces a new framework *Casper** in which mobile users can obtain location-based services without the need to disclose their private location information. *Casper** has two main components, the *location anonymizer* and the *privacy-aware query processor*. The *location anonymizer* acts as a trusted third party that blurs the exact location information of each user into a *cloaked area* that satisfies the user specified privacy requirements. Since the *location anonymizer* part has been widely studied, we focus on only the *privacy-aware* query processing part. The *privacy-aware query processor* is embedded into traditional location-based database servers to tune their functionalities to be *privacy-aware* by dealing with *cloaked areas* rather than exact point information. Three new query types that are supported by *Casper** are identified, *private queries over public data*, *public queries over private data*, and *private queries over private data*. To deal with these three privacy-aware query types, the query processor provides a *candidate list* of answers rather than an exact answer for the user. We have proved that the returned candidate list contains the exact answer and is of minimal size. Then, we propose a *shared execution paradigm* that aims to share computational resources among *continuous* privacy-aware queries, in order to improve system scalability and computational efficiency of the query processor for *continuous* privacy-aware queries. In addition, the performance of the query processor can be tuned through several parameters to achieve a trade-off between system scalability, i.e., query processing time, and query answer optimality, i.e., candidate list size. Extensive experimental evaluation studies the *privacy-aware query processor* of *Casper** and shows its scalability and efficiency with a large number of mobile users, continuous queries, and data, various privacy requirements, and various performance tuning settings.

REFERENCES

Ackerman, L., Kempf, J., and Miki, T. 2003. Wireless location privacy: A report on law and policy in the united states, the europrean union, and japan. Tech. Rep. DCL-TR2003-001, DoCoMo Commuinication Laboratories, USA.

Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Mishra, N., Motwani, R., Srivastava, U., Thomas, D., Widom, J., and Xu, Y. 2004. Vision paper: Enabling privacy for the paranoids. In *Proc. of the International Conference on Very Large Data Bases, VLDB*.

Agrawal, R., Evfimievski, A. V., and Srikant, R. 2003. Information sharing across private databases. In *Proc. of the ACM International Conference on Management of Data, SIGMOD*.

Anonymizer. 1996-2008. Anonymous surfing. http://www.anonymizer.com.

Bamba, B., Liu, L., Pesti, P., and Wang, T. 2008. Supporting anonymous location queries in mobile environments with privacygrid. In *Proc. of the International World Wide Web Conference, WWW*.

Barkhuus, L. and Dey, A. K. 2003. Location-based services for mobile telephony: a study of users' privacy concerns. In *Proc. of the IFIP Conference on Human-Computer Interaction, INTERACT*.

Beresford, A. R. and Stajano, F. 2003. Location privacy in pervasive computing. *IEEE Pervasive Computing 2,* 1, 46–55.

Bureau, U. C. 2006. Tiger/line census files `http://www.census.gov/geo/www/tiger/`.

Cai, Y., Hua, K. A., and Cao, G. 2004. Processing range-monitoring queries on heterogeneous mobile objects. In *Proc. of the International Conference on Mobile Data Management, MDM*.

Cheng, R., Zhang, Y., Bertino, E., and Prabhakar, S. 2006. Preserving user location privacy in mobile data management infrastructures. In *Proc. of Privacy Enhancing Technology Workshop, PET*.

Chow, C.-Y. and Mokbel, M. F. 2007. Enabling private continuous queries for revealed user locations. In *Proc. of the International Symposium on Advances in Spatial and Temporal Databases, SSTD*.

Chow, C.-Y., Mokbel, M. F., and He, T. 2008. Tinycasper: A privacy-preserving aggregate location monitoring system in wireless sensor networks (Demo). In *Proc. of the ACM International Conference on Management of Data, SIGMOD*.

Chow, C.-Y., Mokbel, M. F., and Liu, X. 2006. A peer-to-peer spatial cloaking algorithm for anonymous location-based services. In *Proc. of the ACM Symposium on Advances in Geographic Information Systems, GIS*.

Du, W. and Atallah, M. J. 2001. Secure multi-party computation problems and their applications: A review and open problems. In *Proc. of the New Security Paradigms Workshop*.

Duckham, M. and Kulik, L. 2005. A formal model of obfuscation and negotiation for location privacy. In *Proc. of the International Conference on Pervasive Computing*.

Emekci, F., Agrawal, D., Abbadi, A. E., and Gulbeden, A. 2006. Privacy preserving query processing using third parties. In *Proc. of the International Conference on Data Engineering, ICDE*.

FoxNews. 2004. Man accused of stalking ex-girlfriend with GPS. http://www.foxnews.com/story/0,2933,131487,00.html. september 4.

Gedik, B. and Liu, L. 2004. Mobieyes: Distributed processing of continuously moving queries on moving objects in a mobile system. In *Proc. of the International Conference on Extending Database Technology, EDBT*.

Gedik, B. and Liu, L. 2005. A customizable k-anonymity model for protecting location privacy. In *Proc. of the International Conference on Distributed Computing Systems, ICDCS*.

Gedik, B. and Liu, L. 2008. Protecting location privacy with personalized k-anonymity: Architecture and algorithms. *IEEE Transactions on Mobile Computing 7,* 1, 1–18.

Ghinita, G., Kalnis, P., Khoshgozaran, A., Shahabi, C., and Tan, K.-L. 2008. Private queries in location based services: Anonymizers are not necessary. In *Proc. of the ACM International Conference on Management of Data, SIGMOD*.

Ghinita, G., Kalnis, P., and Skiadopoulos, S. 2007b. Mobihide : A mobile peer-to-peer system for anonymous location-based queries. In *Proc. of the International Symposium on Advances in Spatial and Temporal Databases, SSTD*.

Ghinita, G., Kalnis, P., and Skiadopoulos, S. 2007a. PrivÉ: Anonymous location-based queries in distributed mobile systems. In *Proc. of the International World Wide Web Conference, WWW*.

Gruteser, M. and Grunwald, D. 2003. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proc. of the International Conference on Mobile Systems, Applications, and Services, MOBISYS*.

Gruteser, M. and Liu, X. 2004. Protecting privacy in continuous location-tracking applications. *IEEE Security and Privacy 2,* 2, 28–34.

Gruteser, M., Schelle, G., Jain, A., Han, R., and Grunwald, D. 2003. Privacy-aware location sensor networks. In *Proc. of the Workshop on Hot Topics in Operating Systems, HotOS*.

Güting, R. H., de Almeida, V. T., Ansorge, D., Behr, T., Ding, Z., Höse, T., Hoffmann, F., Spiekermann, M., and Telle, U. 2005. Secondo: An extensible DBMS platform for research prototyping and teaching. In *Proc. of the International Conference on Data Engineering, ICDE*.

Haas, L. M., Miller, R. J., Niswonger, B., Roth, M. T., Schwarz, P. M., and Wimmers, E. L. 1999. Transforming heterogeneous data with database middleware: Beyond integration. *IEEE Data Engineering Bulletin 22,* 1, 31–36.

Hadjieleftheriou, M., Kollios, G., Bakalov, P., and Tsotras, V. J. 2005. Complex spatio-temporal pattern queries. In *Proc. of the International Conference on Very Large Data Bases, VLDB*.

Hashem, T. and Kulik, L. 2007. Safeguarding location privacy in wireless ad-hoc networks. In *Proc. of the International Conference on Ubiquitous Computing, UBICOMP*.

Hengartner, U. and Steenkiste, P. 2003. Protecting access to people location information. In *Proc. of the International Conference on Security in Pervasive Computing, SPC*.

Hong, J. I. and Landay, J. A. 2004. An architecture for privacy-sensitive ubiquitous computing. In *Proc. of the International Conference on Mobile Systems, Applications, and Services, MOBISYS*.

Hu, H. and Lee, D. L. 2006. Range nearest-neighbor query. *IEEE Transactions on Knowledge and Data Engineering, TKDE 18,* 1, 78–91.

Hu, H., Xu, J., and Lee, D. L. 2005. A generic framework for monitoring continuous spatial queries over moving objects. In *Proc. of the ACM International Conference on Management of Data, SIGMOD*.

Iwerks, G. S., Samet, H., and Smith, K. 2003. Continuous k-nearest neighbor queries for continuously moving points with updates. In *Proc. of the International Conference on Very Large Data Bases, VLDB*.

Jefferies, N., Mitchell, C. J., and Walker, M. 1995. A proposed architecture for trusted third party services. In *Proc. of the International Conference on Cryptography: Policy and Algorithms*.

Jensen, C. S. 2004. Database aspects of location-based services. In *Location-based Services*. Morgan Kaufmann, 115–148.

Kalnis, P., Ghinita, G., Mouratidis, K., and Papadias, D. 2007. Preventing location-based identity inference in anonymous spatial queries. *IEEE Transactions on Knowledge and Data Engineering, TKDE 19,* 12, 1719–1733.

Khoshgozaran, A. and Shahabi, C. 2007. Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In *Proc. of the International Symposium on Advances in Spatial and Temporal Databases, SSTD*.

Kido, H., Yanagisawa, Y., and Satoh, T. 2005. An anonymous communication technique using dummies for location-based services. In *Proc. of IEEE International Conference on Pervasive Services, ICPS*.

Kolahdouzan, M. R. and Shahabi, C. 2005. Alternative solutions for continuous k nearest neighbor queries in spatial network databases. *GeoInformatica 9,* 4, 321–341.

Lazaridis, I., Porkaew, K., and Mehrotra, S. 2002. Dynamic queries over mobile objects. In *Proc. of the International Conference on Extending Database Technology, EDBT*.

Li, P.-Y., Peng, W.-C., Wang, T.-W., Ku, W.-S., and Xu, J. 2008. A cloaking algorithm based on spatial networks for location privacy. In *Proceedingds of the International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, SUTC*.

Lin, B. and Su, J. 2005. Shapes based trajectory queries for moving objects. In *Proc. of the ACM Symposium on Advances in Geographic Information Systems, GIS*.

Mokbel, M. F. and Aref, W. G. 2005. Place: A scalable location-aware database server for spatio-temporal data streams. *IEEE Data Engineering Bulletin 28,* 3, 3–10.

Mokbel, M. F., Chow, C.-Y., and Aref, W. G. 2006. The new casper: Query processing for location services without compromising privacy. In *Proc. of the International Conference on Very Large Data Bases, VLDB*.

Mokbel, M. F., Xiong, X., and Aref, W. G. 2004. Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In *Proc. of the ACM International Conference on Management of Data, SIGMOD*.

Mokbel, M. F., Xiong, X., Aref, W. G., Hambrusch, S., Prabhakar, S., and Hammad, M. 2004. Place: A query processor for handling real-time spatio-temporal data streams (Demo). In *Proc. of the International Conference on Very Large Data Bases, VLDB*.

Mouratidis, K., Papadias, D., and Hadjieleftheriou, M. 2005. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In *Proc. of the ACM International Conference on Management of Data, SIGMOD*.

Mouratidis, K., Yiu, M. L., Papadias, D., and Mamoulis, N. 2006. Continuous nearest neighbor monitoring in road networks. In *Proc. of the International Conference on Very Large Data Bases, VLDB*.

Papadias, D., Shen, Q., Tao, Y., and Mouratidis, K. 2004. Group nearest neighbor queries. In *Proc. of the International Conference on Data Engineering, ICDE*.

Paypal. 1999-2008. Paypal. http://www.paypal.com/.

Pfitzmann, A. and Kohntopp, M. 2000. Anonymity, unobservability, and pseudonymity - a proposal for terminology. In *Proc. of the Workshop on Design Issues in Anonymity and Unobservability*.

Prabhakar, S., Xia, Y., Kalashnikov, D. V., Aref, W. G., and Hambrusch, S. E. 2002. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Transactions on Computers 51,* 10, 1124–1140.

Samarati, P. 2001. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering, TKDE 13,* 6, 1010–1027.

Smailagic, A. and Kogan, D. 2002. Location sensing and privacy in a context-aware computing environment. *IEEE Wireless Communication 9,* 5, 10–17.

Sun, J., Papadias, D., Tao, Y., and Liu, B. 2004. Querying about the past, the present and the future in spatio-temporal databases. In *Proc. of the International Conference on Data Engineering, ICDE*.

Sweeney, L. 2002a. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems 10,* 5, 571–588.

Sweeney, L. 2002b. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems 10,* 5, 557–570.

Tao, Y. and Papadias, D. 2005. Historical spatio-temporal aggregation. *ACM Transactions on Information Systems 23,* 1, 61–102.

Tao, Y., Papadias, D., and Shen, Q. 2002. Continuous nearest neighbor search. In *Proc. of the International Conference on Very Large Data Bases, VLDB*.

Tao, Y., Sun, J., and Papadias, D. 2003. Analysis of predictive spatio-temporal queries. *ACM Transactions on Database Systems , TODS 28,* 4, 295–336.

USAToday. 2002. Authorities: GPS system used to stalk woman. http://www.usatoday.com/tech/news/2002-12-30-gps-stalker_x.htm. december 30.

Warrior, J., McHenry, E., and McGee, K. 2003. They know where you are. *IEEE Spectrum 40,* 7, 20–25.

Wolfson, O., Cao, H., Lin, H., Trajcevski, G., Zhang, F., and Rishe, N. 2002. Management of dynamic location information in domino (Demo). In *Proc. of the International Conference on Extending Database Technology, EDBT*.

Wolfson, O., Xu, B., and Chamberlain, S. 2000. Location prediction and queries for tracking moving objects. In *Proc. of the International Conference on Data Engineering, ICDE*.

Xu, T. and Cai, Y. 2007. Location anonymity in continuous location-based services. In *Proc. of the ACM Symposium on Advances in Geographic Information Systems, GIS*.

Xu, T. and Cai, Y. 2008. Exploring historical location data for anonymity preservation in location-based services. In *Proc. of the International Conference of the Computer and Communications Societies, INFOCOM*.

Yiu, M. L., Jensen, C., Huang, X., and Lu, H. 2008. Spacetwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In *Proc. of the International Conference on Data Engineering, ICDE*.

Zhang, J., Zhu, M., Papadias, D., Tao, Y., and Lee, D. L. 2003. Location-based spatial queries. In *Proc. of the ACM International Conference on Management of Data, SIGMOD*.

Received Month Year; revised Month Year; accepted Month Year