

10-2011

Location-Dependent Spatial Query Containment

Ken C. K. LEE

University of Massachusetts - Dartmouth

Brandon UNGER

Microsoft Corporation

Baihua ZHENG


Singapore Management University, bhzheng@smu.edu.sg

Wang-Chien LEE

Pennsylvania State University - Main Campus

DOI: <https://doi.org/10.1016/j.datak.2011.06.001>

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the [Databases and Information Systems Commons](#), and the [Numerical Analysis and Scientific Computing Commons](#)

Citation

LEE, Ken C. K.; UNGER, Brandon; ZHENG, Baihua; and LEE, Wang-Chien. Location-Dependent Spatial Query Containment. (2011). *Data and Knowledge Engineering*. 70, (10), 842-865. Research Collection School Of Information Systems.

Available at: https://ink.library.smu.edu.sg/sis_research/1410

This Journal Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

Location-dependent spatial query containment

Ken C.K. Lee ^{a,*}, Brandon Unger ^b, Baihua Zheng ^c, Wang-Chien Lee ^d

^a Department of Computer and Information Science, University of Massachusetts Dartmouth, North Dartmouth, MA02747, USA

^b Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, USA

^c School of Information Systems, Singapore Management University, Singapore

^d Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA16802, USA

A B S T R A C T

Nowadays, location-related information is highly accessible to mobile users via issuing Location-Dependent Spatial Queries (LDSQs) with respect to their locations wirelessly to Location-Based Service (LBS) servers. Due to the limited mobile device battery energy, scarce wireless bandwidth, and heavy LBS server workload, the number of LDSQs submitted over wireless channels to LBS servers for evaluation should be minimized as appropriate. In this paper, we exploit query containment techniques for LDSQs (called LDSQ containment) to enable mobile clients to determine whether the result of a new LDSQ Q' is completely covered by that of another LDSQ Q previously answered by a server (denoted by $Q' \subseteq Q$) and to answer Q' locally if $Q' \subseteq Q$. Thus, many LDSQs can be reduced from server evaluation. To support LDSQ containment, we propose a notion of *containment scope*, which represents a spatial area corresponding to an LDSQ result wherein all semantically matched LDSQs are answerable with the result. Through a comprehensive simulation, our proposed approach significantly outperforms existing techniques.

Keywords:

Location-Dependent Spatial Query
Query containment
Containment scope

1. Introduction

The popularity of *Location-Based Services* (LBSs) has been rapidly increasing over the past decade [1]. Nowadays, location-related information (such as local news, traffic report, weather, tourist guides, etc.) becomes highly accessible at any place anytime. We refer to location-related information maintained and provided by LBS servers as a collection of *stationary spatial objects* (or *objects*, for short) and requests for location-related information as *Location-Dependent Spatial Queries* (LDSQs). Common LDSQs include range queries, window queries, nearest neighbor (NN) queries and k NN queries [32]. Then, a mobile client accesses nearby objects by submitting an LDSQ along with its location (as a query point) wirelessly to an LBS server. The server evaluates the LDSQ according to the query point and returns the result set (i.e., qualified objects for the query) to the client.

Since their locations are not always fixed, clients may need to reissue LDSQs to the server when they are relocated, so as to refresh corresponding LDSQ results according to their new locations. However, wireless environments are highly resource constrained. First, wireless bandwidth shared by mobile clients is scarce. High contention on the bandwidth leads to a long query latency. Second, short-life batteries and significant power consumption in wireless communications cannot afford mobile clients to frequently issue LDSQs and download query results. For instance, StrongARM SA-1100 processor [2] consumes only 200 mW. In contrast, RangeLAN2 PC wireless card consumes 750 mW and 1500 mW [26] when it receives and sends data, respectively. Third, LBS servers may be overloaded in serving a large mobile client population simultaneously. Thus, mobile clients cannot stay connected to LBS servers.

As we can observe, LDSQs issued at slightly different positions may receive similar or even equal results. Thus, the result set of an LDSQ, if maintained properly, is useful to answer subsequent LDSQs locally. As long as they can be completely answered by

* Corresponding author. Tel.: +1 508 910 6543; fax: +1 508 999 9144.
E-mail address: ken.ck.lee@umassd.edu (K.C.K. Lee).

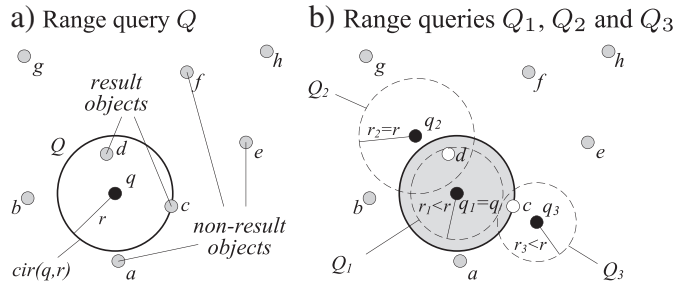


Fig. 1. Illustration of overlapped query results.

clients, some LDSQs can be saved from server evaluation, thereby reducing the energy and bandwidth contention and alleviating server workload. In fact, mobile clients are now powerful to locally evaluate LDSQs if they do not need additional data from servers. Nevertheless, how to effectively determine whether an LDSQ can be fully answered by a maintained query result is an important research problem; and we are addressing this problem in this paper.

1.1. Query containment for LDSQs

In this work, we exploit query containment techniques for LDSQs and name these techniques as *Location-Dependent Spatial Query Containment* (abbreviated as *LDSQ containment*). Different from conventional query containment extensively studied in database query optimizations [5,6,13], our LDSQ containment aims at determining whether the result of an LDSQ, Q' (denoted by R') is contained by that of another LDSQ, Q (denoted by R); formally, $R' \subseteq R$. In this research, we assume *no* predictable user movement; and thus we do not predetermine whether and what LDSQs would be issued in the future and do not preload their results in advance. This assumption is realistic to many daily applications. For instance, in a tourist information system, a tourist may first search for points of attraction within her vicinity. She may then move towards one selected point of attraction while occasionally issuing the LDSQ to ensure her moving path. The tourist's direction is not necessarily indicated to the LBS server for some reasons (e.g., location privacy [12]). Dealing with this scenario, we only consider that a mobile client can answer any LDSQ if its result R' is contained by another LDSQ result R , maintained by the client. By this means, some LDSQs can be locally evaluated by the clients and, on the other hand, the quantity of LDSQs submitted to a server for evaluation can be reduced.

To illustrate how an LDSQ can be answered with a previous LDSQ result, Fig. 1(a) shows a range query Q issued at a query point q upon a set of objects (i.e., $\{a, b, c, d, e, f, g, h\}$).¹ Let $cir(q, r)$ represent Q 's circular search region centered at q with radius r . The result set R is $\{c, d\}$ and the other objects (e.g. g and h) are non-result objects. For another range query Q' with a search area $cir(q', r')$, it is quite trivial to determine $R' \subseteq R$ if $cir(q', r') \subseteq cir(q, r)$. Indeed, there are some other cases, in which $R' \subseteq R$ though $cir(q', r') \not\subseteq cir(q, r)$. As a complete analysis, we list all six cases categorized according to possible relationships between q and q' and those between r and r' below:

- Case 1. $q' = q$ and $r' = r$. The radii and query points of Q' and Q are identical, which immediately implies that $cir(q', r') = cir(q, r)$. Here, it is certain that $R' = R$ as they cover exactly the same searched area.
- Case 2. $q' = q$ and $r' < r$. This case implies that the search area of Q' is fully contained by that of Q (i.e., $cir(q', r') \subset cir(q, r)$) and hence R' must be contained by R . As illustrated in Fig. 1(b), both Q_1 and Q are issued at the same query point, q , and $r_1 < r$, thus $R_1 \subseteq R$.
- Case 3. $q' = q$ and $r' > r$. The radius of Q' is larger than that of Q while both queries are issued at the same query point. Since only those objects inside $cir(q, r)$ are locally available to the client, it is possible that $cir(q', r')$ contains additional objects beyond $cir(q, r)$.
- Case 4. $q' \neq q$ and $r' = r$. Both Q and Q' have the same search area size but are at different query points. This case is common in mobile applications where a client issues the same range query while moving. In this case, the result of Q' is only contained by that of Q if the non-overlapped portion (i.e., $cir(q', r') - cir(q, r)$) contains zero objects. Fig. 1(b) demonstrates this case that Q_2 with $r_2 = r$ but $q_2 \neq q$ gets $R_2 = \{d\} \subset R$.
- Case 5. $q' \neq q$ and $r' < r$. This case considers both the change of search area sizes and the change of query points. This scenario happens, for instance, when a mobile client moves to a location and issues a query of a smaller search range. Consider that $cir(q', r') \not\subseteq cir(q, r)$. Then, R' is contained by R if and only if the area $cir(q', r') - cir(q, r)$ does not contain any object, similar to Case 4. Notice that $cir(q', r') \subseteq cir(q, r)$ is a special case of this condition. Q_3 depicted in Fig. 1(b) provides an example. Area $cir(q_3, r_3) - cir(q, r)$ contains zero object and hence $R_3 = \{c\} \subset R$.
- Case 6. $q' \neq q$ and $r' > r$. This case occurs when a mobile client moves to a location and issues a query of a larger search range. In this case, Q' may cover some additional objects beyond $cir(q, r)$.

This analysis inspires our proposed concept of LDSQ containment. It is noteworthy that some existing approaches (as will be reviewed in Section 2) can only handle a few of the above cases. For example, semantic region [8,33] only deals with Case 1, Case 2 and Case 5, while valid scope [24,41] handles Case 1 and Case 4. Our proposed LDSQ containment deals with all the cases (except

¹ The object distribution example is borrowed from [21].

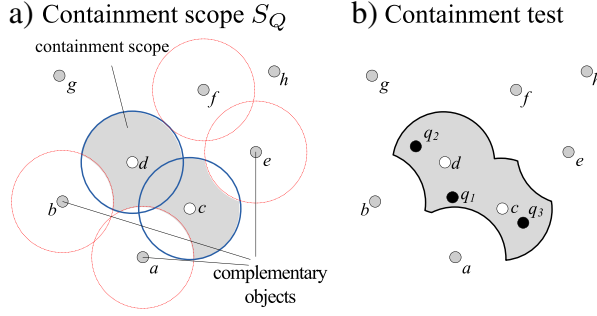


Fig. 2. Containment scope and containment test.

Case 3 and Case 6) at the same time, significantly improving the effectiveness in determining whether LDSQs can be locally answered and thus overall system performance.

On the other hand, when $r' > r$ (i.e., Case 3 and Case 6), the new search space $cir(q', r')$ definitely covers a larger area than $cir(q, r)$. Because only objects inside $cir(q, r)$ are available, the client needs to contact the server for possibly *additional* objects located inside $cir(q', r')$ but outside of $cir(q, r)$. After all objects with $cir(q', r')$ are collected, other range queries fully covered by $cir(q', r')$ can be answered. In many applications, initial queries usually cover broader areas in order to provide coarse views of queried areas. Then, subsequent queries are used to drill down certain areas with smaller scopes. As other research studies, these two cases are not addressed in this work.

1.2. Containment scope

To facilitate LDSQ containment determination, we introduce a notion of *containment scope*. Given a set of objects O distributed in an area S , a containment scope for the result R of an LDSQ Q , denoted by $S_Q (\subseteq S)$, represents a spatial area. If Q' is issued within S_Q and Q' 's search area size is not larger than Q 's, Q' is definitely answerable by R . Fig. 2(a) depicts a containment scope S_Q for the result set R of a range query Q of our example. Now, as shown in Fig. 2(b), because of conditions (1) $r_1 \leq r$ and (2) $q_1 \in S_Q$, the result set $R_1 = \{d\}$ for Q_1 can be completely derived by only evaluating the objects included in R . Similarly, as $r_2 \leq r$, $r_3 \leq r$, and $q_2, q_3 \in S_Q$ hold, Q_2 and Q_3 can also be answered with R locally, with $R_2 = \{d\}$ and $R_3 = \{c\}$, respectively.

On the other hand, the representation of a containment scope S_Q has a direct impact on (1) the wireless communication cost for transmitting S_Q from the server to a client, (2) computational overhead incurred by a client in deciding whether a new query point $q' \in S_Q$, and (3) local client storage cost. Thus, this representation issue is important and is worth our research effort. Intuitively, a containment scope can be represented as a polygon (i.e., a collection of edges and vertices). However, in some situations, a complex polygon that contains many vertices and edges is formed that incurs larger transmission and storage costs. Even worse, polygon-based representation does not provide an exact containment scope for certain LDSQs, e.g., range queries that have circular search areas.

Rather, we represent a containment scope by some object locations. Recall that given a query Q with a result set R , a new query whose search area does not include any non-result object is guaranteed to have its result fully contained by R . Here, we identify only a *representative subset* of non-result objects (called *complementary set*) that sufficiently constitutes the formation of a containment scope. A client preserves a result set R and a complementary set for R , both of which represent a containment scope of R . Referring to Fig. 2(a), the containment scope is represented by a result set $\{c, d\}$ and a complementary set $\{a, b, e, f\}$. Other non-result objects (e.g., g and h) are not downloaded to the client so that both communication and client storage costs can be saved.

1.3. Paper organization

Thus far, we have used range queries to illustrate the concept of LDSQ containment and the notion of containment scopes. In fact, the concept of LDSQ containment is broadly applicable to a variety of LDSQs. While the formulation of containment scopes is highly related to the *types* of LDSQs, we detail our approaches for various LDSQs in this paper. We conduct an extensive evaluation through simulations to validate the efficiency of the proposed approaches.

The remainder of the paper is organized as follows. Section 2 reviews related works and discusses the differences of LDSQ containment from the existing approaches. Section 3 provides the system model and outlines LDSQ processing algorithms as the basis of our discussion. Section 4 discusses LDSQ containment for range and window queries; and Section 5 discusses that for NN and kNN queries. Section 6 analyzes the performance of the LDSQ containment. Section 7 evaluates our proposed LDSQ containment in comparison with the existing approaches through simulations. Finally, as the conclusion of the paper, Section 8 lists our contributions presented and states our future research plans.

2. Related work

The idea of LDSQ containment is borrowed from query containment [5,6,13], which is developed to determine whether a query result (or a materialized database view) is sufficient to answer a query based only on their query expressions. Differently, LDSQ containment

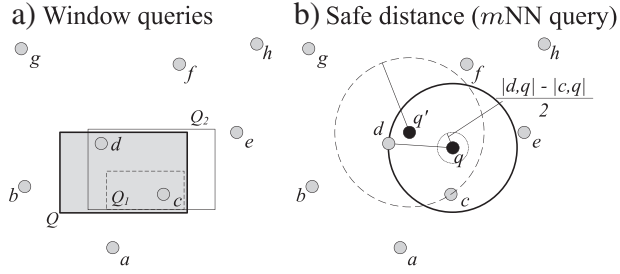


Fig. 3. Semantic regions.

introduced in this paper considers query types, query parameters, query points and more importantly, the spatial distribution of objects that leads to an important notion of containment scopes. Related to LDSQ containment are some research works. Continuous LDSQs are evaluated over time and their results are updated when query points are changed and/or objects move. Meanwhile, semantic regions [8,22,30,33] and valid scope [21,24,41] are designed to assert whether LDSQs produce the same previous query results. Both of them are the most related to our LDSQ containment, though they follow different design principles and assumptions. We review all of them in the following. Besides [18] provides a comprehensive review of location-dependent query processing.

2.1. Continuous queries

In general, there are two major types of approaches for continuous queries. The first type of approaches assumes that the movements of clients and objects are not predictable [7,10,11,14,16,28,29,34,39,40,42,44]. Thus, LDSQs are reevaluated whenever their results become invalidated when a query point changes or some object locations change. Usually LBS servers need to keep track of all queries; and mobile clients stay connected to the servers for query result updates. To avoid unneeded location updates, some techniques, like *safe regions* [16,29] that are areas wherein objects are located and object locations would not affect any queries, are explored. Usually the size and shape of safe regions are predefined. Differently, our containment scope is formulated and computed based on queries and object locations. The second type of approaches assumes that queries and objects are moving in predictable ways [17]. Various algorithms [3,19,20,23,35,36] for continuous queries were proposed to compute query results according to client and object trajectories. Our work does not assume any given client trajectories.

2.2. Semantic region

The main idea of semantic regions is to determine whether the queried area of an LDSQ Q' (denoted by $A_{Q'}$) is covered by that of another LDSQ Q (denoted by A_Q). As long as $A_{Q'} \subseteq A_Q$, the result for Q' , should be fully contained by that for Q . This semantic region technique for window queries was first explored in [8]. As shown in Fig. 3(a), the result for a window query Q , $R = \{c, d\}$ is preserved together with the description A_Q as the semantic region for R . As another window query Q_1 has searched area A_{Q_1} , fully covered by A_Q , Q_1 is guaranteed to be fully answered with R only. The same idea for k NN was studied in [33]. In [33], a *safe distance*, based on triangular inequality, has been developed to examine if a k NN query result is subsumed by existing m NN objects. Given m NN objects, o_1, o_2, \dots, o_m , obtained with respect to a query point q , k NN query issued at q' can be answered if the displacement from q to q' is bounded by a safe distance, i.e., $(|o_m, q| - |o_k, q|)/2$ where $k \leq m$.² Fig. 3(b) shows a safe distance, i.e., $(|d, q| - |c, q|)/2$, for an NN query over 2NN objects, i.e., c and d .

However, the idea and technique of semantic regions are overly conservative. In Fig. 3(a), another window query Q_2 with search area only partially covered by A_Q is strictly considered to be not fully answered by the query result. In fact, it covers the same Q 's result set. The safe distance for k NN queries is purely based on the result objects, without considering any non-result object. In Fig. 3(b), another 2NN query, issued at q' which is more than $(|d, q| - |c, q|)/2$ away from q , is considered to be not covered by the 2NN query result. It actually covers the same two result objects c and d . In contrast, LDSQ containment can handle all these cases better.

2.3. Valid scope

A valid scope corresponding to the result set of a query Q represents an area such that an *identical* LDSQ issued inside the area should have R as its result set. A valid scope can be determined by an intuitive approach [41] that simulates the client movement for all possible directions from the query point to probe those non-result objects which affect result validity. Based on this idea, a number of time-parameterized (TP) queries [35] are issued to identify non-result objects right after a query result is determined. We refer to these as *TP query based approaches*. For window query, the initial valid scope for its result is formed as the intersection of Minkowski regions [9] among all result objects. Here, a Minkowski region is a rectangular region centered at an object with its extent equal to that of the query window. Then, a number of TP-window queries with identical window size are issued from the current query point towards all vertices of the current form of valid scope. If any TP-window query touches a non-result object o'

² We use $|p, q|$ to represent Euclidean distance between points p and q .

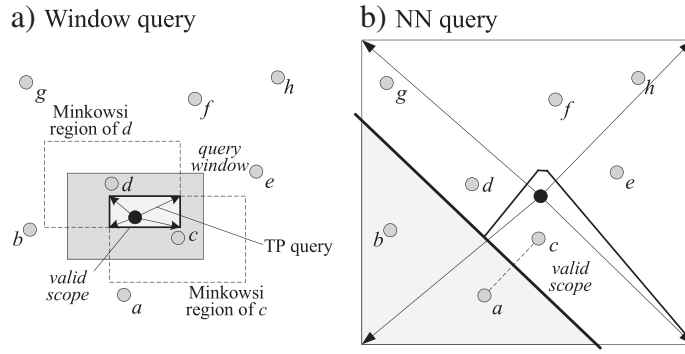


Fig. 4. TP query based valid scope computation.

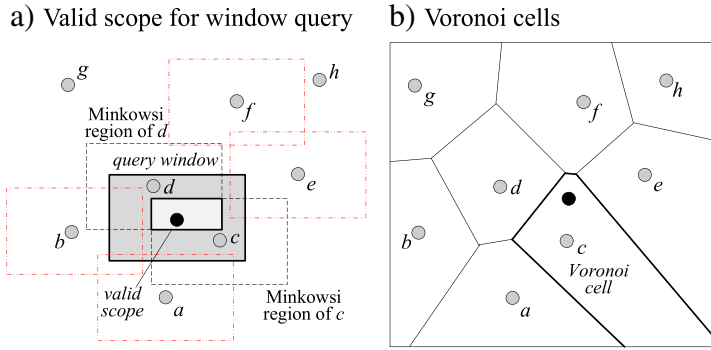


Fig. 5. Geometry-based approach and Voronoi cell.

before reaching its target vertex, the valid scope is trimmed by o' Minkowski window. Fig. 4(a) shows a formation of the valid scope for a window query, in which arrows represent the directions of TP-window queries towards the current valid scope, i.e., intersection of c 's and d 's Minkowski windows. This valid scope refinement repeats until no more non-result object can be probed to further refine the valid scope. For NN query, TP-NN queries are issued to formulate the valid scope of an NN query result. A number of TP-NN queries are issued towards all vertices of a valid scope, which is initialized as an entire area. If any non-result object is probed before a TP-NN query reaches the vertex of the current form of valid scope, the valid scope is refined. Fig. 4(b) shows the valid scope for an NN query derived by TP-NN queries. Due to exhaustive TP query invocations, the TP query based approaches always incur a long processing time and high I/O cost.

Alternatively, efficient *geometry-based* valid scope computation algorithms [21,24] have recently been proposed. Different from TP query based approaches, these approaches exploit the geometrical relationship between result object and non-result objects to formulate a valid scope. The main idea of these approaches is to explore a search area gradually from a given query point. The objects collected in an initial search space are result objects. Later, non-result objects located in the rest of the search space are checked against the result objects. Those non-result objects constituting the formation of a valid scope are collected while the others are discarded. Fig. 5(a) provides an example of valid scope computation for a window query. The geometry-based approaches can also support range queries that the TP query based approaches cannot. Additional to online computation, for NN query, the valid scope of an NN query (i.e., Voronoi cell [9]) can be pre-calculated, and each object is stored with a Voronoi cell, as shown in Fig. 5(b). When an NN query is evaluated, the Voronoi cell of the result object is provided to the client [43]. As will be discussed later, our online containment scope computation algorithms follow the same design principles as geometry-based approaches. However, since the definition of containment scope differs from that of valid scope, the algorithms for containment scope computation presented in this paper are not the same as those for valid scopes.

Table 1

Summary of existing approaches and LDSQ containment (where Q' is a new LDSQ and Q is an old LDSQ).

| | Range query | Window query | kNN query |
|-----------------------------------|-----------------------------------------------------------------|-----------------------------------------------------|------------------------------------------------------------------------------|
| Semantic region | $A_Q \subseteq A_Q \Rightarrow R' \subseteq R$ [8] | | $ q, q' \leq \frac{ O_m, q - O_k, q }{2} \Rightarrow R' \subseteq R$ [33] |
| Valid scope (TP query based) [41] | Not supported | $Q' = Q^{(\#)} \wedge q \in V_R \Rightarrow R' = R$ | |
| Valid scope (geometry-based) [21] | $Q' = Q^{(\#)} \wedge q \in V_R \Rightarrow R' = R$ | | |
| LDSQ | Q' matches $Q \wedge q \in S_Q(q) \Rightarrow R' \subseteq R$ | | |

(#) where Q' and Q are of the same type and with the same query parameters.

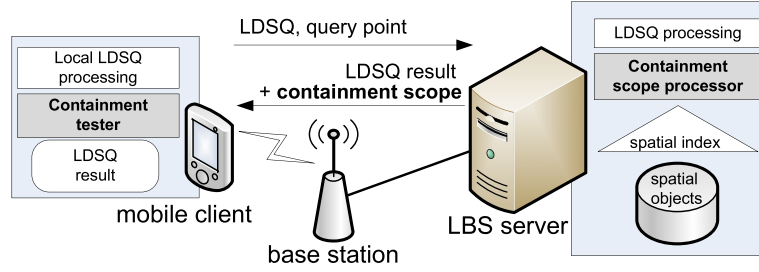


Fig. 6. System model for supporting LDSQ containment.

Finally, we summarize the differences among those most related approaches, namely, semantic regions and valid scopes, in terms of the main ideas and the supported types of queries in Table 1.

3. Preliminaries

To exploit LDSQ containment, we extend a conventional LBS client/server system model, as shown in Fig. 6, with three add-on components: (1) a *containment scope processor* at the server side, (2) a *containment tester* and (3) a *local LDSQ processor* both at the client side.

Since the computation of a containment scope (as collecting complementary objects) requires the knowledge of both result objects and all non-result objects, the containment scope processor at an LBS server computes containment scopes for LDSQs. We implement the containment scope processor with efficient containment scope computation algorithms and integrate them seamlessly with the LDSQ processor to optimize the overall processing cost. Without loss of generality, all objects in an LBS server are indexed on their spatial coordinates by an R-tree [25], for its wide acceptance and efficiency. In R-tree, objects are recursively grouped and indexed by index nodes. Next, a best-first traversal algorithm [15] on R-tree is adopted to answer LDSQs. The best-first algorithm efficiently retrieves objects for LDSQs in an ascending order of their mindists [31] to given query points. This is efficient for both LDSQ processing and containment scope computation because of two reasons. First, when a search terminates, the remaining priority queue preserves all non-result objects, based on which a containment scope for the query result can be derived. Next, the processing of LDSQs and the formation of their corresponding containment scopes can be seamlessly integrated into a single index traversal, as will be discussed later.

Thereafter, the server returns R together with a corresponding containment scope S_Q to the client as the response. Later, when the client issues a new LDSQ Q' , it first consults the containment tester to test against the result set of previously issued LDSQ Q and its containment scope, i.e., R and S_Q . The containment tester implements containment test algorithm designed for different types of LDSQs. Then, it submits Q' to the server *only* when the containment tester indicates $R' \not\subseteq R$ and downloads additional queried objects from the server. Otherwise, the local LDSQ processor returns required objects for Q' from the result set that passes the containment test.

4. LDSQ containment for range and window queries

Since range queries and window queries share a lot of similarities, we, in this section, first formulate containment scope, develop containment scope computation algorithm and containment test algorithm, and propose optimization techniques for range queries. Later, we extend those techniques to handle window queries.

4.1. Formulation of containment scope for range query

Every range query $Q_{range}(q, r)$ searches for objects located inside a specified circle $cir(q, r)$ with query point q as its center and radius r . Formally, the result set R of $Q_{range}(q, r)$ equals $\{o | o \in O, o \in cir(q, r)\}$. Alternatively, we can see that a set of objects $\{o\}$, with their Minkowski circles $cir(o, r)$ covering q , form the same query result. Fig. 7(b) illustrates the Minkowski circles of all the objects. As q is only located inside $cir(c, r)$ and $cir(d, r)$, objects c and d form the result set.

Recall that the *valid scope* V_R for a result set R is the area wherein R remains valid for a same LDSQ issued within V_R (see Section 2). That is exactly an area covered by the Minkowski circles of all the result objects but not covered by those of any non-result objects. Formally, V_R can be expressed as follows:

$$V_R = \bigcap_{o \in R} cir(o, r) - \bigcup_{o' \in O - R} cir(o', r)$$

where the first term $\bigcap_{o \in R} cir(o, r)$ represents the intersection of all the Minkowski circles of the result objects, i.e., a common area where all the result objects are covered by a query; and the second term $\bigcup_{o' \in O - R} cir(o', r)$ refers to an area where at least one non-result object is included as a result object. Revisit our example. The valid scope for $R_{range}(q, r) = \{c, d\}$ is depicted in Fig. 7(b). For any $q' \in V_R$, $R_{range}(q' r) = \{c, d\}$.

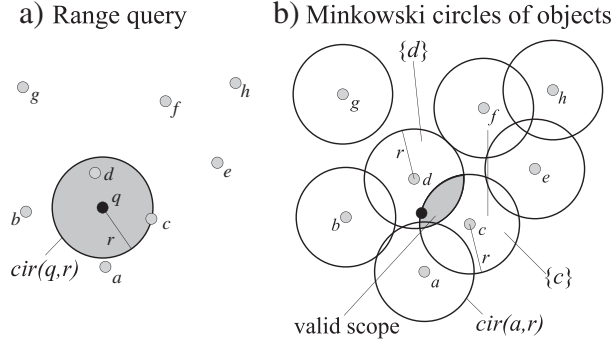


Fig. 7. Range query circle and Minkowski circles of objects.

Unlike valid scope, containment scope maximizes the reusability of individual result objects of $R_{range}(q, r)$ by considering not only the same range queries issued at different query point with the same range distance r , but also those semantically contained by $Q_{range}(q, r)$. Let $R' (\subseteq R_{range}(q, r))$ denote the result set of an LDSQ that can be answered by $R_{range}(q, r)$. The containment scope, denoted by $S_{range}(q, r)$, is derived in Eq. (1):

$$\begin{aligned}
S_{range}(q, r) &= \bigcup_{\forall R' \subseteq R_{range}(q, r)} V_{R'} = \bigcup_{\forall R' \subseteq R_{range}(q, r)} \left(\bigcap_{o \in R'} cir(o, r) - \bigcup_{o' \in O - R'} cir(o', r) \right) \\
&= \bigcup_{\forall R' \subseteq R_{range}(q, r)} \bigcap_{o \in R'} cir(o, r) - \bigcup_{\forall R' \subseteq R_{range}(q, r)} \bigcup_{o' \in O - R'} cir(o', r) \\
&= \bigcup_{o \in R_{range}(q, r)} cir(o, r) - \bigcup_{o' \in O - R_{range}(q, r)} cir(o', r).
\end{aligned} \tag{1}$$

Here, the first term $\bigcup_{o \in R_{range}(q, r)} cir(o, r)$ denotes a total area covered by *any* result object, whereas the second term $\bigcup_{o' \in O - R_{range}(q, r)} cir(o', r)$ denotes an area covered by non-result objects. This implies more range queries issued at various locations can be answered using $R_{range}(q, r)$ and thus more savings are expectedly attained. In our example, the containment scope for $R_{range}(q, r)$ (see Fig. 2(b)) is larger than the valid scope (as shown in Fig. 7(b)).

As we can observe from Eq. (1), the derivation of $S_{range}(q, r)$ requires a complete evaluation of all the non-result objects against individual result objects, which inevitably incurs non-negligible overhead. To reduce the number of non-result objects accessed, we re-formulate the calculation of $S_{range}(q, r)$ in Eq. (1) based on the set relationship between any two sets $A = \{a_1 \dots a_{|A|}\}$ and $B = \{b_1 \dots b_{|B|}\}$, i.e., (1) $A - B \equiv A - (A \cap B)$, and (2) $A \cap B \equiv \bigcup_{i=1}^{|A|} (\{a_i\} \cap B) \equiv \bigcup_{j=1}^{|B|} \bigcap_{i=1}^{|A|} (\{a_i\} \cap \{b_j\})$. The improved formation of $S_{range}(q, r)$ is expressed in Eq. (2).

$$S_{range}(q, r) = \bigcup_{o \in R_{range}(q, r)} cir(o, r) - \left(\bigcup_{o' \in O - R_{range}(q, r)} \left(\bigcup_{o \in R_{range}(q, r)} cir(o, r) \cap cir(o', r) \right) \right) \tag{2}$$

Algorithm RangeQueryContainmentScope(q, r, R, P)

Input. query point (q), radius (r), result set (R),
priority queue (P) from the search algorithm
Output. complementary set (C)

Begin

1. **while** (P is not empty) **do**
2. $(\epsilon, d) \leftarrow P.dequeue()$;
3. **if** ($\exists o \in R, cir(o, r)$ overlap $cir(\epsilon, r)$) **then**
4. **if** (ϵ is an index node) **then**
5. **foreach** child c of ϵ **do**
6. $P.enqueue((c, |c, q|))$;
7. **else** /* ϵ is an object */
8. $C \leftarrow C \cup \{\epsilon\}$;
9. **output** C ;

End.

Fig. 8. Algorithm RangeQueryContainmentScope.

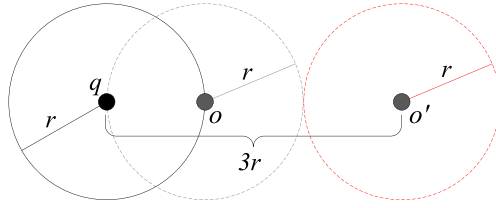


Fig. 9. Maximum search space $cir(q, 3r)$ for complementary objects.

From Eq. (2), we can see that only those non-result objects o' having Minkowski circles overlapping at least one result object's Minkowski circle are needed. Those non-result objects are termed *complementary objects*.

4.2. Containment scope computation for range query

Based on Eq. (2), we derive online containment scope computation algorithm, namely, *RangeQueryContainmentScope*, as outlined in Fig. 8. While the algorithm can be seamlessly integrated with the best-first search algorithm, for simplicity, we discuss it as a stand-alone algorithm. After a search completes, a result set, R , and a priority queue, P , which holds all the non-result objects in a non-decreasing order of mindists to q , are taken as inputs to our algorithm.

The algorithm iteratively examines the head entry (ϵ, d) from P with ϵ and d representing an R-tree index node or an object and its distance from q , respectively. When ϵ (i.e., a node or an object) has its Minkowski circle overlapping with the Minkowski circle of any result object, a detailed examination is performed (lines 4–8). If an entry is a node, it is explored and all its children are inserted to P for further examination (lines 4–6). Otherwise, it must be an object, and is added to a complementary set C (lines 7–8) if it constitutes a part of the containment scope. When P becomes empty, the algorithm outputs C and terminates. As in Lemma 1, the entire search area for all complementary objects and result objects is bounded by $cir(q, 3r)$ for $Q_{range}(q, r)$.

Lemma 1. The largest search space for complementary objects for a given range query $Q_{range}(q, r)$ is bounded by a circle $cir(q, 3r)$. \square

Proof. As illustrated in Fig. 9, for a range query, $Q_{range}(q, r)$, the maximum distance between any result object and the query point q does not exceed r . As two circles $cir(o, r)$ and $cir(o', r)$ for two objects o and o' overlap only when $|o, o'| \leq 2r$, the longest distance between the query point and a complementary object for a range query $Q_{range}(q, r)$ must be $3r$. \blacksquare

Fig. 10 illustrates how the algorithm runs for a range query $Q_{range}(q, |q, c|)$. Right after $Q_{range}(q, |q, c|)$ is evaluated, the result set R contains c and d and the priority queue P maintains (a, b, N_3, g) . Those objects are depicted in Fig. 10(a). First, a , the head entry of P is examined. Since $cir(a, r)$ overlaps $cir(c, r)$, a is included in C . Next, N_3 's Minkowski range overlaps the result objects' circles as shown in Fig. 1. Its children, e, f , and h are put into P to update P to (b, e, f, g, h) , as depicted in Fig. 1. Later, b is dequeued and its circle overlaps with $cir(d, r)$. Then, b is included in C . Subsequently, e and f are dequeued. As their Minkowski circles overlap with those of result objects, C is updated to $\{a, b, e, f\}$. Finally, g and h are dequeued to empty P . As the Minkowski circles of g and h do not overlap with that of any result object, the algorithm finishes and returns the complementary objects (i.e., a, b, e , and f). The containment scope is shown in Fig. 10(d).

4.3. Removable complementary objects

By identifying complementary objects, non-result objects can be ignored from presenting a containment scope. However, some of collected complementary objects (called *removable complementary objects*) may not have an impact on the shape of a containment scope, although they satisfy Eq. (2). This is because their impact is entirely hidden by that of other complementary objects. Fig. 3

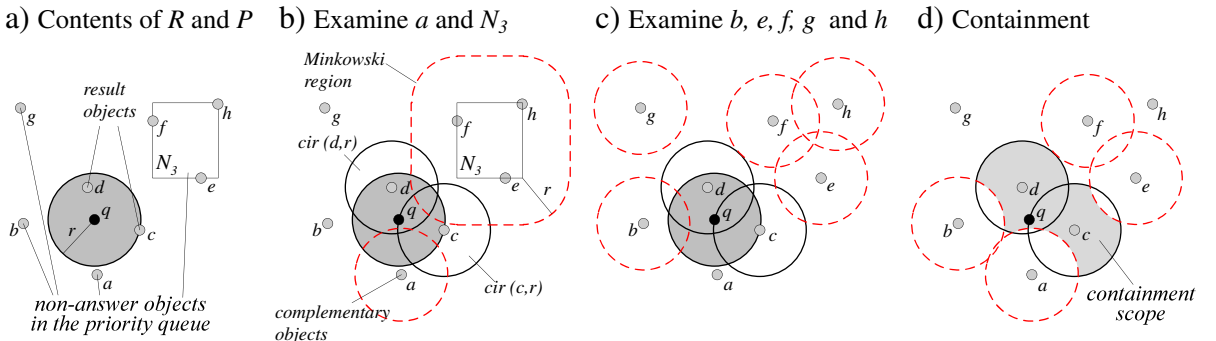


Fig. 10. Determining containment scope (range query).

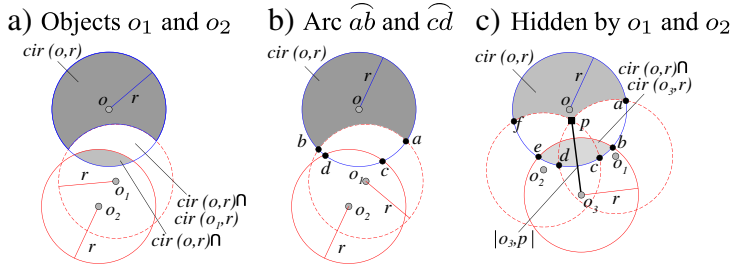


Fig. 11. Detection of removable complementary objects.

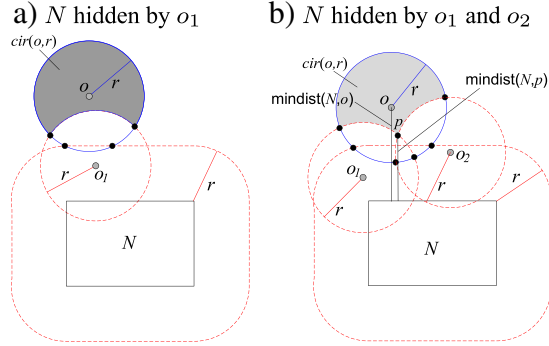


Fig. 12. Detection of redundant complementary objects in R-tree index nodes.

depicts two complementary objects o_1 and o_2 . With respect to a result object o , the overlap between $cir(o, r)$ and $cir(o_1, r)$ completely covers that between $cir(o, r)$ and $cir(o_2, r)$. In this case, o_2 is a removable complementary object. We can ignore o_2 while the same containment scope is formed. Removal of those removable complementary objects can reduce the transmission and storage costs for the complementary objects and some index I/O costs in containment scope computation. In this subsection, we explore the issue of removable complementary object identification.

Basically, a complementary object o' is removable if overlaps between $cir(o', r)$ and $cir(o, r)$ for all result objects o are fully covered by some other complementary objects. Here, we discuss two possible cases where complementary object o' is removable. For the first case that o is the result object and o_1 and o_2 are two complementary objects, that an overlap between $cir(o, r)$ and $cir(o_2, r)$ is fully covered by that between $cir(o, r)$ and $cir(o_1, r)$ can be determined with their arcs on $cir(o, r)$. As shown in Fig. 11(b), the arc \widehat{ab} of o_1 entirely covers \widehat{cd} of o_2 on the same $cir(o, r)$. Thus, o_2 is asserted to be removable. The second case happens when a complementary object is jointly covered by more than one other complementary objects. Fig. 11(c) shows an example. To determine if o_3 is removable due to other complementary objects o_1 and o_2 , we exploit another property as follows. We determine an intersection point p between the perimeter of $cir(o_1, r)$ and $cir(o_2, r)$ inside $cir(o, r)$. If $|o_3, p| > r$, o_3 is certainly hidden by the union of $cir(o_1, r)$ and $cir(o_2, r)$ with respect to o .

Similarly, complementary objects enclosed in an index node can be examined. Fig. 12(a) illustrates the intersection arc for an index node N on a result object o fully covered by that for a complementary object o_1 . Fig. 12(b) exemplifies that the arc for a node N on a result object o is covered by those for two complementary objects o_1 and o_2 and the distance from N to p , the intersection point of the overlaps with respect to o_1 and o_2 , is longer than r . In these two examples, all N 's enclosed objects are removable and thus N is ignored.

Based on the discussed idea, we devise Algorithm *RangeQueryRemovableCheck*, which can be incorporated into Algorithm *RangeQueryContainmentScope*, to identify removable complementary objects. The pseudo-code is outlined in Fig. 13. It takes an entry ϵ , which can be an index node or object, and examines it against all existing complementary objects stored in C for each result object. If ϵ is not removable, it reports false; otherwise true. The algorithm examines ϵ against each result object o iteratively and conducts two checks. The first check (line 3) examines the existence of a complementary object c whose arc $\widehat{c_o}$ fully covers that for ϵ , i.e., $\widehat{\epsilon_o}$. If such c exists, ϵ may be removable. The second check (lines 4–6) tests ϵ against two other complementary objects whose union of arcs fully covers $\widehat{\epsilon_o}$. If ϵ has its distance to an intersection point p (of the perimeters of the complementary objects) not longer than r , ϵ is not removable (line 6).³ If *hidden* it remains false after the two checks, ϵ is determined to be not removable and the algorithm terminates without examining all the remaining result objects (line 7). Otherwise, ϵ is determined to be hidden after examining all result objects, and it can be removed.

³ There should be two intersection points on the perimeters of two circles and here, we use one inside the result object.

Algorithm *RangeQueryRemovalCheck*(q, r, R, C, ϵ)

Input. query point (q), radius (r), result set (R),
complementary set (C),
an entry (node or object) being examined (ϵ),
Output. true when ϵ is removable, else false

Begin

1. **foreach** $o \in R, |o, \epsilon| \leq 2r$ **do**
2. $hidden \leftarrow \text{false};$
3. **if** $\forall c \in C \widehat{c}_o \subseteq \widehat{c}_\epsilon$ **then** $hidden \leftarrow \text{true};$
4. **if** $\exists C' \subseteq C, \forall c \in C' \widehat{c}_o \cap \widehat{c}_\epsilon \neq \emptyset$ **then**
5. **foreach** $c, c' \in C'$ **do**
 /* where the joint arc of c_o and c'_o fully covers o 's */
6. **if** $\exists_p |p, c| = r, |p, c'| = r, |p, o| < r \wedge |\epsilon, p| > r$
 then $hidden \leftarrow \text{true};$
7. **if** $hidden = \text{false}$ **then** **output** false;
8. **output** true;

End.

Fig. 13. Algorithm *RangeQueryRemovalCheck*.

The filtering of removable complementary objects is, however, time consuming due to exhaustive object comparisons. In our performance evaluation, we will discuss its performance, compared with the previously presented containment scope computation algorithm that simply collects all non-result objects as long as their Minkowski circles overlap with those of result objects.

4.4. Containment test for range query

Given a range query result and its containment scope that is represented by result objects and complementary objects, the containment test algorithm determines whether a new range query can be answered by the result. The pseudo-code is listed in Fig. 14. Whenever a client receives a new range query Q' , it needs to conduct two checks before sending Q' to the server. The first check is a semantic check to see whether Q' is semantically contained by the previous query via comparing their radii (line 1). The second one is to check whether its query point is located inside the Minkowski circles of any result objects and outside those of all complementary objects (lines 2–3). The query that passes these two checks will be answered locally.

4.5. LDSQ containment for window query

Due to the similarity between range and window queries, we directly extend the developed concepts and techniques for range query result to window query result. Given a window query $Q_{window}(q, l, w)$, $rect(q, l, w)$ represents a rectangular window which is centered at q , with length and width of $2l$ and $2w$, respectively. Hence, the result set of $Q_{window}(q, l, w)$ is $\{o | o \in O, o \in rect(q, l, w)\}$. Modifying Eq. (2) for the context of window queries, we express the containment scope for a window query result $R_{window}(q, l, w)$ denoted by $S_{window}(q, l, w)$, as in Eq. (3).

$$S_{window}(q, l, w) = \bigcup_{o \in R_{window}(q, l, w)} rect(o, l, w) - \left(\bigcup_{o' \in O - R_{window}(q, l, w)} \left(\bigcup_{o \in R_{window}(q, l, w)} rect(o, l, w) \cap rect(o', l, w) \right) \right) \quad (3)$$

The best-first traversal can also be utilized to process window query and to compute the containment scope for the result together. An example window query is depicted in Fig. 15(a), with $rect(q, l, w)$ as the search area, $\{c, d\}$ as the result set, and the

Algorithm *RangeQueryContainmentTest*($Q_{range}(q', r')$,
 $Q_{range}(q, r), R_{range}(q), C$)

Input. a new range query $Q_{range}(q', r')$,
a previous range query $Q_{range}(q, r)$
result set $R_{range}(q)$, complementary set (C)

Output. boolean (true: $Q_{range}(q', r')$ is fully covered)

Begin

1. **if** ($r' > r$) **then** **output** false;
2. **else if** ($\nexists o \in R_{range}(q), q \in cir(o, r)$) **then** **output** false;
3. **else if** ($\exists o' \in C, q \in cir(o', r')$) **then** **output** false;
4. **else** **output** true;

End.

Fig. 14. Algorithm *RangeQueryContainmentTest*.

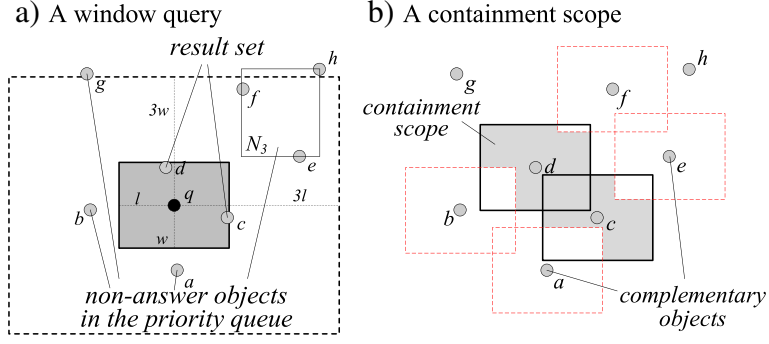


Fig. 15. Containment scope for window query.

dash line rectangle showing the upper bound of the search space for complementary objects. As stated in Lemma 2, the complementary objects and the final containment scope are depicted in Fig. 15(b).

Lemma 2. The maximum search space for complementary objects for a given window query $Q_{\text{window}}(q, l, w)$ is bounded by a rectangle $\text{rect}(q, 3l, 3w)$. \square

Proof. For a window query, $Q_{\text{window}}(q, l, w)$, the possibly farthest result object, o will be l (or w) from q on x - (y -) dimension. Centering at o , $\text{rect}(o, l, w)$ can touch a complementary object, o' , $2l$ ($2w$) away from it. Hence, the longest distance from o' to q is $3l$ ($3w$). \blacksquare

Further, some removable complementary objects with Minkowski rectangles hidden by those of other complementary objects can be ignored. The idea of how to determine if a complementary object is removable is illustrated in Fig. 16(a) and (b). In Fig. 16(a), whenever the overlapped area between a result object o and a complementary object o_2 is fully covered by that between o and any other complementary object o_1, o_2 can be removed. In some cases as shown in Fig. 16(b), removable complementary objects are detected when their Minkowski rectangles are covered by complementary objects in part but are entirely covered by the union of other complementary objects.

Finally, given a new window query $Q_{\text{window}}(q', l', w')$ whose search area is $\text{rect}(q', l', w')$, the containment test examines if $l' \leq l$ and $w' \leq w$, and if no complementary object is covered by $\text{rect}(q', l', w')$. The query is sent to the server when any check in the containment test fails.

5. LDSQ containment for k nearest neighbor query

In this section, we turn our focus to LDSQ containment for k NN queries. We formulate the containment scope for k NN query result, and we derive the corresponding algorithms for on-line containment scope computation and containment test.

5.1. Formulation of containment scope for k NN query

A k NN query $Q_{nn}(q, k)$ returns k objects nearest to q . Formally, the result set of $Q_{nn}(q, k)$, $R_{nn}(q, k)$, is $\{o | o \in O_{nn} \subseteq O, |O_{nn}| = k \wedge \forall o' \in O_{nn} \forall o'' \in O - O_{nn} |o| \leq |o', o''|\}$. Now, suppose that an NN query (i.e., $k = 1$) is issued at a query point q and its result object is o . With respect to any non-result object, o' , the entire search space S can be partitioned into two disjointed half-planes $HP_{o,o'}$ and $HP_{o',o}$ along the perpendicular bisector $\perp_{o,o'}$ between o and o' , where $HP_{o,o'}$ covers o and $HP_{o',o}$ covers o' . Fig. 17(a) shows two half-planes $HP_{d,f}$ and $HP_{f,d}$ formed based on the perpendicular bisector $\perp_{d,f}$. Any point located inside the half-plane $HP_{d,f}$ should have d closer to it than f . Thus, an area where o is guaranteed to be the closest to q among all the objects can be

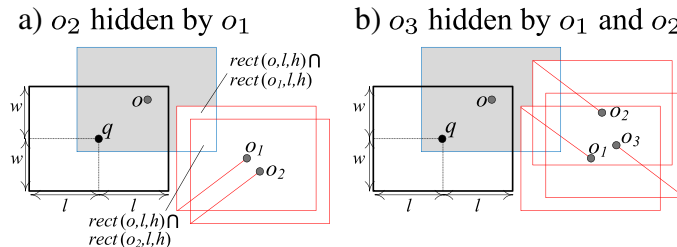


Fig. 16. Redundant complementary objects.

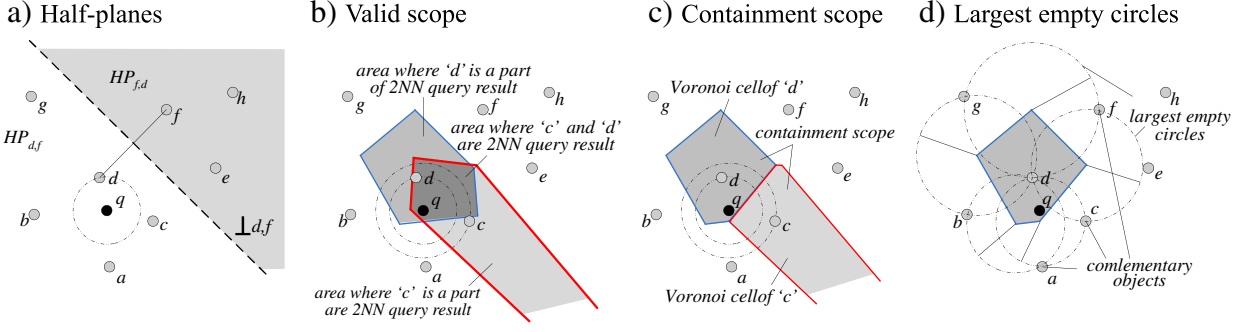


Fig. 17. Containment scope for kNN query result.

determined as an intersection area of all half-planes of o formed against all the other objects, i.e., $S \cap (\bigcap_{o' \in O - \{o\}} HP_{o,o'})$. This area is known as *Voronoi cell* [9,27], which is also a containment scope and a valid scope for the NN query result.

Generalizing this idea, we consider a k NN query issued at q and its result set R . We obtain half-planes formed by each result object $o \in R$ and each non-result object $o' \in O - R$. The valid scope V_R wherein R remains valid is formulated as:

$$V_R = S \cap \bigcap_{o \in R} \bigcap_{o' \in O - R} HP_{o,o'},$$

or

$$V_R = S - \bigcup_{o \in R} \bigcup_{o' \in O - R} HP_{o',o}.$$

Fig. 17(b) shows the valid scope for the result set (i.e., $\{c,d\}$) for a 2NN query, that is an intersection between the Voronoi cell of c and that of d . Differently, the containment scope is formed as a collection of valid scopes covering a k NN result $R_{nn}(q,k)$ and all possible subsets R' where $R' \subseteq R_{nn}(q,k)$. Hence, the containment scope, $S_{nn}(q,k)$, is formulated in Eq. (4).

$$S_{nn}(q,k) = \bigcup_{R' \subseteq R_{nn}(q,k)} V_{R'} = \bigcup_{R' \subseteq R_{nn}(q,k)} \left(S - \bigcup_{o \in R'} \bigcup_{o' \in O - R'} HP_{o',o} \right) = \bigcup_{o \in R_{nn}(q,k)} \left(S - \bigcup_{o' \in O - \{o\}} HP_{o',o} \right). \quad (4)$$

As Eq. (4) indicates, the containment scope is expressed as a subtraction of a union of half-planes formed between individual result objects and other non-result objects from an entire search space. In other words, this is the union of Voronoi cells of individual result objects. As shown in Fig. 17(c), the containment scope for a 2NN query result is composed of Voronoi cells of c and d , that is observably larger than the corresponding valid scope.

Despite all the half-planes are considered in Eq. (4), many of them can be safely discarded without affecting the formation of the containment scope. That means not all non-result objects are needed. To identify and eliminate those removable half-planes that have no impact on a containment scope, we exploit the largest empty circle property of Voronoi cell. Every circumcircle $cir(v, |v, o|)$ that centers at a vertex v of a Voronoi cell for an object o and has $|v, o|$ as its radius must touch o and other two objects on its perimeter. Then, v is a *true* vertex if and only if its circumcircle $cir(v, |v, o|)$ encloses no objects. Based on this largest empty circle property, we develop an efficient containment scope calculation algorithm for k NN query.

Algorithm k NNQueryContainmentScope(q, R, P)

Input. query point (q), result set ($R = \{o_1, \dots, o_k\}$), priority queue (P)

Local. k vertices sets ($V[1 \dots k]$), k complementary sets ($C[1 \dots k]$)

Output. union of k complementary sets

Begin

1. **while** (P is not empty) **do**
2. $(\epsilon, d) \leftarrow P.dequeue()$;
3. **if** (ϵ is a node) **then**
4. **if** ($\exists o_i \in R \wedge \exists v \in V[i], |o_i, v| > |\epsilon, v|$) **then**
5. **foreach** child c of ϵ **do**
6. $P.enqueue((c, |c, q|))$;
7. **else**
8. **foreach** $o_i \in R$ **do**
9. **if** ($\epsilon \neq o_i \wedge \exists v \in V[i], |o_i, v| > |\epsilon, v|$) **then**
10. update $V[i]$ and $C[i]$ with respect to ϵ ;
11. **output** $\bigcup_{i \in [1, k]} C[i] - R$;

End.

Fig. 18. Algorithm k NNQueryContainmentScope.

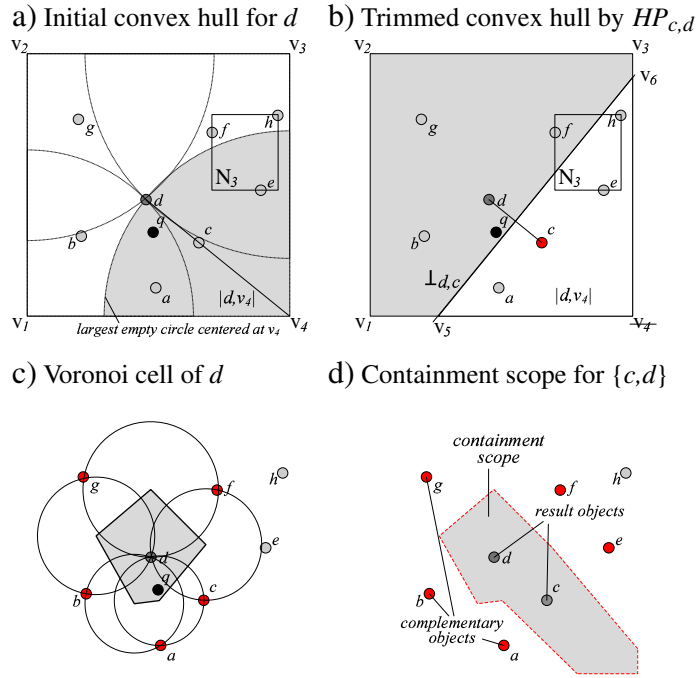


Fig. 19. Determining the containment scope (2NN query).

5.2. Containment scope computation for k NN query

With the best-first based NN search algorithm, all entries remained in the priority queue P are non-result objects, after k NN objects are found. The containment scope computation algorithm whose pseudo-code is depicted in Fig. 18 takes a query point q , a result set R and a priority queue P as inputs. The basic idea is to construct the Voronoi cell for each result object o_i represented by a set of vertices maintained in an array element $V[i]$ that in turn are contributed by complementary objects kept in an array element $C[i]$. All the Voronoi cells for result objects are initialized as convex hulls to cover an entire search space, and get refined when objects are examined in the process.

Since result objects might contribute to the Voronoi cells of other result objects, we put all the result objects back to the priority queue P before the processing. First, we initialize a convex hull for each result object o_i ($1 \leq i \leq k$) as the entire rectangular search space with four vertices. Then, the algorithm repeatedly examines each entry ϵ that could be a node or an object from P . If it is a node and it is covered by a largest empty circle formed by a vertex of any convex hull, ϵ is explored and all its child nodes are enqueued for later examination (lines 4–6). If ϵ is an object, it is checked against all the convex hulls. For those convex hulls with any largest empty circle covering the object, they, together with corresponding complementary objects, are updated (lines 8–10). To examine if ϵ is inside a largest empty circle v of a convex hull with respect to a result object o , we compare the distance between ϵ and v , (i.e., $|\epsilon, v|$) and that between o and v (i.e., $|o, v|$). The process continues until P becomes empty or all the pending entries are out of all the existing largest circles.

To illustrate how the algorithm derives a containment scope for a k NN query result, Fig. 19 gives an example in which a 2NN query is issued at point q and its result set contains objects c and d . To simplify the discussion, we only explain the formation of the Voronoi cell of d , and that of c can be formed in the same fashion. Initially, the convex hull for d is set to the entire area S with four vertices v_1, v_2, v_3 , and v_4 , as shown in Fig. 19(a), and the priority queue $P = \{d, c, a, b, N_3, g\}$.⁴ Next, the algorithm examines objects and index nodes in a priority queue P according to the ascending order of their mindist to q . First, d is dequeued. As it is the result object, we skip the detailed examination step. Second, c is dequeued and it is covered by the largest empty circle centered at v_4 . Then, the convex hull is trimmed by the half-plane $HP_{c,d}$ and is refined to a new set of vertices v_1, v_2, v_3, v_6 , and v_5 , with c inserted into C as a tentative complementary object (see Fig. 19(b)). Afterwards, other objects are examined in the same fashion, and the final convex hull (i.e., Voronoi cell) of d is shown in Fig. 18. Objects a, b, c, f and g are the complementary objects for d . The Voronoi cell of c is derived simultaneously. The final complementary objects are a, b, e, f and g . Notice that c and d are excluded as they are result objects.

⁴ Different from range query and window query, all the result objects are inserted back to P as each of them might affect the formation of the Voronoi cell of some other result object.

| | |
|------------------|------------------------------------------------------------------------------------------------------------|
| Algorithm | $k\text{NNQueryContainmentTest}(Q_{nn}(q', k'), Q_{nn}(q, k), R_{nn}(q, k), C)$ |
| Input. | query point (q'), number of NNs (k'), result set ($R_{nn}(q, k)$), complementary objects (C) |
| Local. | ordered list of objects (L); |
| Output. | boolean (true: $Q_{nn}(q', k')$ is fully covered) |
| Begin | |
| 1. | if ($k' > k$) then output false; |
| 2. | put objects in both $R_{nn}(q, k)$ and C to L ; |
| 3. | sort all objects in L in ascending distance order from q' ; |
| 4. | if $\exists o \in \text{head}(L, k')$, $o \in C$ then output false; |
| 5. | else output true; |
| End. | |

Fig. 20. Algorithm $k\text{NNQueryContainmentTest}$.

5.3. Containment test for $k\text{NN}$ query

With a complementary set C and a $k\text{NN}$ query result set maintained by a client, it is straightforward to determine whether $k'\text{NN}$ can be answered locally. The logic of the test as depicted in Fig. 20 is to examine if $k' \leq k$ (line 1) and if no complementary object would be covered (lines 3–4). In the test, we put result objects and complementary objects into an ordered list based on an ascending order of their distances to a query point, q' . If all of the first k' objects are result objects, the test is satisfied and those collected objects are the result set of $k'\text{NN}$ query issued at q' . Otherwise, the $k'\text{NN}$ query needs to be sent to the server for evaluation.

6. Performance analysis

In this section, we conduct a theoretical analysis to quantify the overhead incurred for LDSQ containment in terms of containment scope computation and its transmission for range, window and $k\text{NN}$ queries, and determine the benefit gained. To facilitate our discussion, Table 2 summarizes the notations used hereafter.

First, we denote ρ as the probability that a new LDSQ *cannot* be answered with a previous query result by a client; in other words, a new LDSQ has a probability $(1-\rho)$ to be answered locally. Since a containment scope is derived for every new query result, the server processing cost and communication cost can be estimated as $\rho \cdot \text{proc}_{Q+C}$ and $\rho \cdot (\text{comm}_R + \text{comm}_C)$, respectively. Compared with a conventional system (i.e., bare query processing) that evaluates every LDSQ, supporting LDSQ containment only incurs additional containment scope computation cost (i.e., $\rho \cdot \text{proc}_C$) and extra communication cost for delivering complementary objects to a client (i.e., $\rho \cdot \text{comm}_C$). LDSQ containment is beneficial if $\rho \cdot \text{proc}_C$ and $\rho \cdot \text{comm}_C$ are smaller than proc_Q and comm_R , respectively.

To facilitate our analysis, we assume that all objects O are in fixed storage size s_o and they are evenly distributed in a two-dimensional unit space (i.e., [1]). The average distance between any two adjacent objects, denoted by δ , can be approximated as

$\sqrt{\frac{1}{|O|}}$, according to [37]. With respect to a query point q , we assume that all the objects $o_i \in O$ are in a non-descending order of d_i ,

with $d_i = |q, o_i| \approx \sqrt{\frac{i}{\pi|O|}}$ according to [4]. Besides, all the objects are indexed by an R-tree with fan out f on their spatial coordinate of size s_c .

6.1. Performance analysis for range and window queries

In this section, we focus our analysis on range queries followed by a brief discussion on window queries. Here, we estimate (i) ρ_{range} , (ii) $\text{proc}_{Q,\text{range}}$ and $\text{proc}_{C,\text{range}}$, and (iii) $\text{comm}_{R,\text{range}}$ and $\text{comm}_{C,\text{range}}$.

Table 2
Summary of notations.

| Notations | Description |
|---------------------|-------------------------------------------------------------------------------------------------------------|
| O | A set of objects and its cardinality is $ O $. |
| R | A result set and its cardinality is $ R $. |
| C | A complementary set and its cardinality is $ C $. |
| s_o | The size of an object that includes both data content and spatial coordinate. |
| s_c | The size of a complementary object (with only a spatial coordinate) (8 bytes). |
| ρ | The probability that an LDSQ cannot be answered locally. |
| proc_Q | The processing cost of evaluating an LDSQ only. |
| proc_{Q+C} | The processing cost of evaluating an LDSQ and a computing corresponding containment scope. |
| proc_C | The processing cost of computing a containment scope (i.e., $\text{proc}_{Q+C} - \text{proc}_Q$). |
| comm_R | The communication cost (in terms of bytes) of transferring result objects to the client (i.e., $s_o R $). |
| comm_C | The communication cost of transmitting complementary objects to the client (i.e., $s_c C $). |

6.1.1. Estimation of ρ_{range}

Assume that a client initially submits a range query $Q_{range}(q, r)$ with a search area $cir(q, r)$, and it obtains a result $R_{range}(q, r)$ (or R_{range} for simplicity in the following) and a containment scope $S_{range}(q, r)$ in the client locally. When the client submits a new range query $Q_{range}(q', r')$ with $r' = r$, the result of $Q_{range}(q', r')$ will be contained by $R_{range}(q, r)$ if $q' \in S_{range}(q, r)$. Without loss of generality, we assume that q' is uniformly distributed in the space and hence $\rho_{range}(q', r) = (1 - |S_{range}(q, r)|)$ (i.e., the area outside of $S_{range}(q, r)$).

Suppose that r is reasonably large, and so $|R_{range}|$ covers a number of result objects which are evenly distributed within $cir(q, r)$. The area covered by the Minkowski circles of all the result objects is approximately $cir(q, 2r)$, i.e., the maximum size of a containment scope for R if no complementary object exists. Then, the containment scope is formed by trimming this area with the Minkowski circles of complementary objects. As depicted in Fig. 21(a), a containment scope can be considered as a fan of "arrowhead-like" shapes formed individually by the query point and partial perimeters of Minkowski circles of complementary objects. Then, determining the size of a containment scope involves two tasks, namely, (A) determining the number of complementary objects $|C_{range}|$ and (B) estimating the areas of individual "arrowhead-like" shapes, which constitute the containment scope. For simplicity, we do not consider those removable complementary objects in this analysis.

(A) Estimation of $|C_{range}|$. As shown in Fig. 21(b), each non-result object o_i forms an angular range Θ_i with respect to q (i.e., $2 \cdot \arcsin \frac{r}{d_i} = 2 \cdot \arcsin \frac{r \sqrt{\pi |O|}}{\sqrt{i}}$). Due to the fact that Θ_i formed by object o_i may overlap with Θ_j formed by object o_j , we denote the angular range that is *uniquely* contributed by a non-result object o_i as Θ'_i where $\Theta_i \leq \Theta'_i$. With the assumption that non-result objects are uniformly distributed around q and the generated angular ranges also follow the uniform distribution, we estimate Θ'_i as $\left(1 - \frac{\sum_{j=|R_{range}|+1}^{|R_{range}|+i-1} \Theta_j}{2\pi}\right) \Theta_i$. Given the fact that complementary objects are those non-result objects with their Minkowski circles intersecting with those of result objects (e.g., $cir(q, 2r)$ in this example), the union of all Θ'_i s (each formed by a complementary object) covers $(0, 2\pi)$ with respect to q .

As C_{range} is $\{o_{|R_{range}|+1}, \dots, o_{|R_{range}|+|C_{range}|}\}$, where $\sum_{i=1}^{|C_{range}|} \Theta_i = 2\pi$, the size of the complementary set (i.e., $|C_{range}|$) can be determined as in Eq. (5).

$$\begin{aligned} |C_{range}| &= \min \left(n \left(1 \leq n \leq |O| - |R_{range}| \right) \wedge \sum_{i=1}^n \Theta'_i = 2\pi \right) \\ &= \min \left(n \left(1 \leq n \leq |O| - |R_{range}| \right) \wedge \left[\sum_{i=1}^n \left(1 - \frac{\sum_{j=|R_{range}|+1}^{|R_{range}|+i-1} \Theta_j}{2\pi} \right) \left(2 \cdot \arcsin \frac{r \sqrt{\pi |O|}}{\sqrt{i}} \right) \right] = 2\pi \right) \end{aligned} \quad (5)$$

When r grows, Θ_i of each complementary object o_i increases; and thus, fewer complementary objects are resulted.

(B) Estimation of containment scope area. Now, we determine the area of an "arrowhead-like" shape. Referring Fig. 21(b), $A(o_i)$, which represents the area of the "arrowhead-like" shape formed by object o_i , can be computed as the difference between the area of two triangles and that of a sector. Let $\alpha_i + \beta_i$ be a range of angles at q towards the boundary of the Minkowski circle of a complementary object o_i and let $\Delta(d_i, \alpha_i)$, $\nabla(d_i, \beta_i)$ and $\sphericalangle(d_i, \alpha_i, \beta_i)$ represent the area of an upper triangle that bounds α_i , the area of a lower triangle that bounds β_i , and the area of a sector, respectively as stated in Eq. (6).

$$A(o_i) = \Delta(d_i, \alpha_i) + \nabla(d_i, \beta_i) - \sphericalangle(d_i, \alpha_i, \beta_i) \quad (6)$$

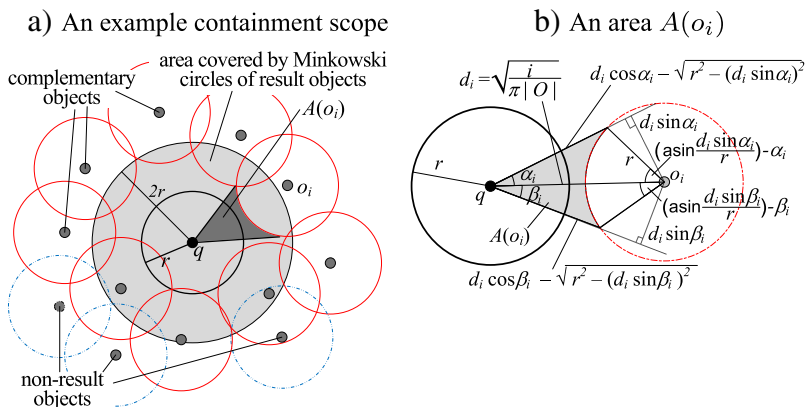


Fig. 21. Estimation of $|S_{range}(q, r)|$.

where

$$\begin{aligned}\Delta(d_i, \alpha_i) &= \frac{1}{2} \cdot d_i \sin \alpha_i \cdot (d_i \cos \alpha_i - \sqrt{r^2 - (d_i \sin \alpha_i)^2}), \\ \nabla(d_i, \beta_i) &= \frac{1}{2} \cdot d_i \sin \beta_i \cdot (d_i \cos \beta_i - \sqrt{r^2 - (d_i \sin \beta_i)^2}), \text{ and} \\ \sphericalangle(d_i, \alpha_i, \beta_i) &= \left(\operatorname{asin}\left(\frac{d_i \sin \alpha_i}{r}\right) + \operatorname{asin}\left(\frac{d_i \sin \beta_i}{r}\right) - (\alpha_i + \beta_i) \right) \cdot \frac{r^2}{2}\end{aligned}$$

Based on both Eqs. (5) and (6), we can now compute the probability that queries are sent to the server for processing, ρ_{range} , i.e., $1 - |S_{range}(q, r)|$ as stated in Eq. (7).

$$\rho_{range} = 1 - |S_{range}(q, r)| = 1 - \sum_{o_i \in C_{range}} A(o_i) = 1 - \sum_{o_i \in C_{range}} (\Delta(d_i, \alpha_i) + \nabla(d_i, \beta_i) - \sphericalangle(d_i, \alpha_i, \beta_i)) \quad (7)$$

6.1.2. Estimation of $proc_{Q,range}$ and $proc_{C,range}$

We estimate the total processing cost of a range query and corresponding containment scope computation, i.e., $proc_{Q+C,range}$ in terms of number of index page accesses, since it is the major processing cost. While the search needs to cover both $|R_{range}|$ and $|C_{range}|$, the search area becomes $circ(q, \omega)$ where $\omega = \sqrt{\frac{|R_{range}| + |C_{range}|}{\pi|O|}}$. According to the R-tree cost model [37], the number of R-tree node accesses equals the sum of the expected node accesses at all levels, while the expected node access is the product of number of index nodes times node access probabilities (that is the expanded MBB's area). Here, the node accesses for base query processing and query processing integrated with containment scope computation are stated in Eqs. (8) and (9), respectively.

$$proc_{Q,range} = \sum_{t=1}^{\lceil \log_f^N \rceil} \left[N_t \cdot (s_t^2 + 4rs_t + \pi \cdot r^2) \right] = \sum_{t=1}^{\lceil \log_f^N \rceil} \left[N_t \cdot \left(\frac{D_t}{N_t^2} + \frac{4r\sqrt{D_t}}{N_t} + \pi \cdot r^2 \right) \right] \quad (8)$$

$$proc_{Q+C,range} = \sum_{t=1}^{\lceil \log_f^N \rceil} \left[N_t \cdot \left(\frac{D_t}{N_t^2} + \frac{4 \cdot \omega \sqrt{D_t}}{N_t} + \pi \cdot \omega^2 \right) \right] \quad (9)$$

where s_t is the expected side length of each node's MBB at level t and it equals $\sqrt{D_t} = 1 + \frac{\sqrt{D_{t-1}} - 1}{\sqrt{f}}$ while $\sqrt{D_1} = 1 - \frac{1}{\sqrt{f}}$; $N_t = \frac{N_{t-1}}{f}$ while $N_1 = \frac{N}{f}$; and finally $\lceil \log_f^N \rceil$ determines the height of an R-tree index.

Then, the net cost incurred by the containment scope computation (i.e., $proc_{C,range}$) is $\sum_{t=1}^{\lceil \log_f^N \rceil} \left[N_t \cdot \left(\frac{4(\omega-r)\sqrt{D_t}}{N_t} + \pi \cdot (\omega-r)^2 \right) \right]$ (i.e., Eqs. (9)–(8)). For a large r , $|C_{range}|$ is expected to be small and thus ω is close to r ; and in this case, $proc_{C,range}$ is expected to be very small.

6.1.3. Estimation of $comm_{R,range}$ and $comm_{C,range}$

Further, we examine the communication cost incurred by LDSQ containment. Compared with bare query processing, LDSQ containment consumes extra bandwidth to download the spatial coordinates of complementary objects, $comm_{C,range} = s_c \cdot |C_{range}|$. While $(1 - \rho_{range})$ LDSQs can be answered locally, the communication cost saved is $(1 - \rho_{range}) \cdot comm_{R,range} \approx |S_{range}(q, r)| \cdot |R_{range}| \cdot s_o \approx |S_{range}(q, r)| \cdot (\pi r^2 |O|) \cdot s_o$. Given the fact that the size of a result object is larger than that of object coordinate (i.e., $s_o \gg s_c$) and the number of complementary objects (i.e., $|C_{range}|$) is small, the extra communication cost caused by LDSQ containment is reasonably insignificant, i.e., $comm_{C,range} \ll (1 - \rho_{range}) \cdot comm_{R,range}$.

6.1.4. Analysis for window queries

Since window queries are very similar to range queries except that window queries adopt rectangular search areas rather than circles, the performance analysis for window queries can be conducted based on the above methodology for range queries. To facilitate the analysis, we consider window queries $Q_{window}(q, l, w)$ whose search areas are squares, i.e., $l=w$. Then, we can consider that window queries with square search areas are equivalent to range queries with $r = \sqrt{\frac{21 \cdot 2w}{\pi}}$. Hence, the probability that a window query needs to be submitted to the server, ρ_{window} , the server processing overhead in computing a containment scope, $proc_{C,window}$, and the communication overhead for transferring complementary objects, $comm_{C,window}$, can all be derived based on the above techniques. To save space, we omit the detailed discussion.

6.2. Performance analysis for kNN queries

Next, we analyze the performance for kNN queries. We consider a kNN query $Q_{knn}(q, k)$ whose result $R_{knn}(q, k)$ (or R_{knn} for simplicity in the following) has a containment scope $S_{knn}(q, k)$. In the following, we estimate (1) the probability ρ_{knn} that a query

needs to be sent to the server, (2) the processing cost $proc_{Q,knn}$ incurred in evaluating a k NN query $Q_{knn}(q, k)$, and the processing cost $proc_{C,knn}$ for formulating its containment scope $S_{knn}(q, k)$, (3) the communication cost $comm_{R,knn}$ for transmitting a query result to the client, and the communication cost $comm_{C,knn}$ for downloading the complementary objects to the client.

As objects are uniformly distributed in the space, each object has an equal likelihood to be included into the result set of a k NN query $Q_{q,knn}$. Hence, the size of the containment scope, i.e., $|S_{knn}(q, k)|$, can be approximated as $\frac{1}{k|O|}$ and the probability that a new $Q_{q,knn}$ is not contained by a preserved containment scope and needs to be sent to server for processing can be estimated as $(1 - |S_{knn}(q, k)|)$, as stated in Eq. (10).

$$\rho_{knn} = 1 - S_{knn}(q, k) = 1 - \frac{1}{k|O|} \quad (10)$$

It is observed that ρ_{knn} increases as $|O|$ and/or k increase which is consistent with our expectation. Thus, the containment scope for a larger k is expected to be smaller which forces more queries to be sent to the server for processing.

Then, we estimate the total processing cost in evaluating $Q_{knn}(q, k)$ and computing $S_{knn}(q, k)$ in terms of node accesses. The search area that covers k NN objects with respect to q is expected to be a circle $cir(q, d_k)$ with a radius d_k (i.e., $\sqrt{\frac{k}{\pi|O|}}$). Further, since complementary objects are those neighbors around the result objects and as previously discussed, the expected distance between objects is δ (i.e., $\sqrt{\frac{1}{|O|}}$), the expected search area for both result objects and complementary objects is $cir(q, d_k + \delta)$, i.e., equivalent to $cir\left(q, \sqrt{\frac{k}{\pi|O|}} + \sqrt{\frac{1}{|O|}}\right)$. With the R-tree cost model [37], we determine $proc_{Q,knn}$ and $proc_{Q+C,knn}$ as the number of node accesses in Eqs. (11) and (12), respectively.

$$proc_{Q,knn} = \sum_{l=1}^{\lceil \log_f \frac{|N|}{f} \rceil} \left[N_l \cdot \left(\frac{D_l}{N_l^2} + 4 \frac{\sqrt{D_l}}{N_l} d_k + \pi \cdot d_k^2 \right) \right] \quad (11)$$

$$proc_{Q+C,knn} = \sum_{l=1}^{\lceil \log_f \frac{|N|}{f} \rceil} \left[N_l \cdot \left(\frac{D_l}{N_l^2} + 4 \frac{\sqrt{D_l}}{N_l} (d_k + \delta) + \pi \cdot (d_k + \delta)^2 \right) \right] \quad (12)$$

The number of node accesses needed for deriving containment scope $S_{knn}(q, k)$, i.e., $proc_{C,knn}$, takes only $\sum_{l=1}^{\lceil \log_f \frac{|N|}{f} \rceil} \left[4 \frac{\sqrt{D_l}}{N_l} (\delta) + 2 \cdot d_k \cdot \delta + \pi \cdot \delta^2 \right]$. This in fact is not significant, compared with $proc_{Q,knn}$.

Last, based on a fact that k result objects are located inside the circle $cir(q, d_k)$ and the overall search range for complementary objects and result objects is $cir(q, d_k + \delta)$, the number of complementary objects, $|C_{knn}|$, can be approximated as $\pi \cdot |O| \cdot ((d_k + \delta)^2 - d_k^2)$. Then, the transmission overhead for complementary objects, $comm_{C,knn} = |C_{knn}| \cdot s_c$. Notice that this containment scope can save the communication overhead of $|S_{knn}(q, k)| \cdot k \cdot s_o$.

6.3. Discussion

As we analyzed, the incurred overhead is not significant compared with bare query processing that processes every query at the server. Further, as the effectiveness of LDSQ containment is dependent on certain query parameters (e.g., q' , r , l , w , or k), the presented theoretical analysis is useful for a system to decide whether to adopt the bare LDSQ processing or LDSQ containment scope. In the next section, we validate our cost model through simulations.

7. Experiments

This section evaluates the effectiveness of our proposed LDSQ containment through simulations, with the primary focus on (1) overall system performance improvement, and (2) the processing overhead at the server when different approaches are used. The experiment results all indicate that LDSQ containment produces outstanding performance in comparison with those mostly related approaches reviewed in Section 2.

7.1. Experiment setup

In our evaluation, we implemented different approaches for comparison, as summarized in Table 1. They include bare query processing, semantic scope (for range and window query [8,22,30] and for extitkNN query [33]), valid scope (TP query based approaches) (for window and k NN queries only) [35], valid scope (geometry based approach) [21], additional to LDSQ containment introduced in this paper. We label them as Bare, Semantic, Valid (TPQ), Valid (Geo) and LDSQC, respectively, in the following discussion. In some circumstances, we use Valid to refer both Valid (TPQ) and Valid (Geo) if they generate the same

Table 3
Experiment parameters.

| Parameter | Value |
|--------------|-----------------------------------------------------------------------------------------------------------------------|
| Approaches | Bare, Semantic, Valid (TPQ), Valid (Geo), LDSQC, and LDSQC Opt |
| Service area | [1000, 1000] |
| Object sets | Uni (1 k, 10 k, 100 k), Gau (10 k) and Real |
| Query types | Range (radius $r = 10, 15, 20$), Window (square, side length $l = 10, 15, 20$), k NN ($k = 1, 4, 16, 64$) |
| Client | Maximum distance D moved per step (1, 5, 10) |
| Index cache | 5%, of index size for Valid (TPQ) |

results and the context is clear. Except Bare, Semantic and Valid as well as LDSQC respectively define semantic scope, valid scope and containment scope, are collectively called *auxiliary scopes* in this evaluation.

In detail, Bare is a baseline approach that clients submit all their queries to the server for processing and it involves no auxiliary scope computation. Semantic checks whether the new query area is completely covered by the previous one for range and window queries. For k NN queries, Semantic collects m (equal to $k+1$) nearest objects (i.e., k result objects plus the spatial coordinate of the nearest non-result object). Both Valid (TPQ) and Valid (Geo) form a valid scope for an LDSQ result, but they adopt different valid scope computation algorithms. Finally, for LDSQC, we evaluate the removable complementary object filtering logic for range and window queries (see Section 4.3) and label this approach as LDSQCopt. Then, LDSQC is referred to as LDSQ containment with non-optimized containment scopes, which might include some removable complementary objects. For k NN queries, there are no removable complementary objects.

Our evaluations use both synthetic and real object sets. Synthetic object sets are used to test the sensitivity of our approaches to various object cardinalities and distributions as well as query parameters. Those synthetic object sets are produced with object locations based on *Uniform* distribution and *Gaussian* distribution, in which the mean and the standard deviation are fixed at 500 and 100, respectively. Realistic object sets are used to examine the practicality of our approaches in real environments. The real object set that is obtained from the United States Census Bureau TIGER/Line dataset [38] includes the locations of ~ 110 k *shopping malls* across the United States. All of the object sets are normalized to a two-dimensional service area of 1000×1000 units. Further, we fix the storage sizes of an object content and an object location (spatial coordinates) at 256 bytes and 8 bytes, respectively, in the experiments. To save space, we present the representative set of results based on synthetic object sets with 1 k, 10 k, and 100 k objects in Uniform distribution (denoted by Uni), 10 k objects in Gaussian distribution (denoted by Gau) and real object set (denoted by Real).

Following the object distributions, we generate client positions at which queries are issued and processed. We include 10 clients initially placed at different random positions and they navigate in 100 steps in the area based on a random walk model. In every step, a client chooses a random direction to move towards and proceeds with the distance varying between 0 and D units. The value of D varies from 1, to 5 and to 10. After the completion of one movement step, the client issues one query, so 100 queries in total. The three discussed types of queries, namely, range, window, and k NN queries are evaluated. The radii of range queries r are varied from 10, 15, up to 20 (units). The search area of window queries is set to be square-shaped and its side length l is increased from 10, 15, up to 20 (units). The k of k NN queries is set to 1, 4, 16 and 64. We implemented all the approaches with GNU C++. We ran our simulations (each consisting one LBS server and some clients) on 30 Solaris Blade-1000 Workstations equipped with 750 MHz CPUs and 1 GB RAM each running the SunOS 5.10 operating system. This configuration enabled us to measure the server performance, bandwidth consumption and number of server (client) answered queries. All the experimented object sets are indexed by R-tree [25] with a disk page size of 4 KB. Particularly, the maximum capacities for non-leaf nodes and leaf nodes are 204 and 340, respectively. In addition, a cache with its size equal to 5% of the R-tree index size managed with LRU replacement policy is used to alleviate server I/O costs for Valid (TPQ) that needs repeated index accesses.

Here, we consider the four commonly used performance metrics: (1) *query submission rate*, (2) *bandwidth consumption*, (3) *I/O cost*, and (4) *server execution time*. Query submission rate is the ratio of the number of server processed LDSQs to the total amount of experimented LDSQs. Bandwidth consumption measures the amount of data (in kilobytes) transmitted over a downlink wireless channel from the server to the client, assuming that the bandwidth consumed in submitting client queries on an uplink channel is negligible. I/O cost counts the number of accessed R-tree indexed nodes that the server has to read from disk to answer a query and to form a corresponding auxiliary region if needed. Further, server execution time (in unit of milliseconds) measures the duration from the time an LDSQ is started to be processed to the time when a result and a corresponding auxiliary scope if needed is computed. We have evaluated the client processing time taken in determining whether LDSQs can be answered locally. In general, the average processing time about all the approaches ranges around 0.1 ms to 0.25 ms. To save space, we do not include these measurements in the discussion.

In what follows, we examine the performance of clients who issue LDSQs while moving when different approaches are adopted. Then, we evaluate the impact of object densities on the overall system performance. In general, we shall see that LDSQ containment can effectively enable clients to assert if queries can be answered with a previous result, thus saving overall system processing costs.

7.2. Exp. 1. Performance evaluation on fixed query parameters

The first experiment set studies the overall system performance for clients issuing queries with identical parameters, while the distance D moved between steps is varied. First, Fig. 22 plots the performance for range queries with radii fixed at 15. Valid (TPQ) that

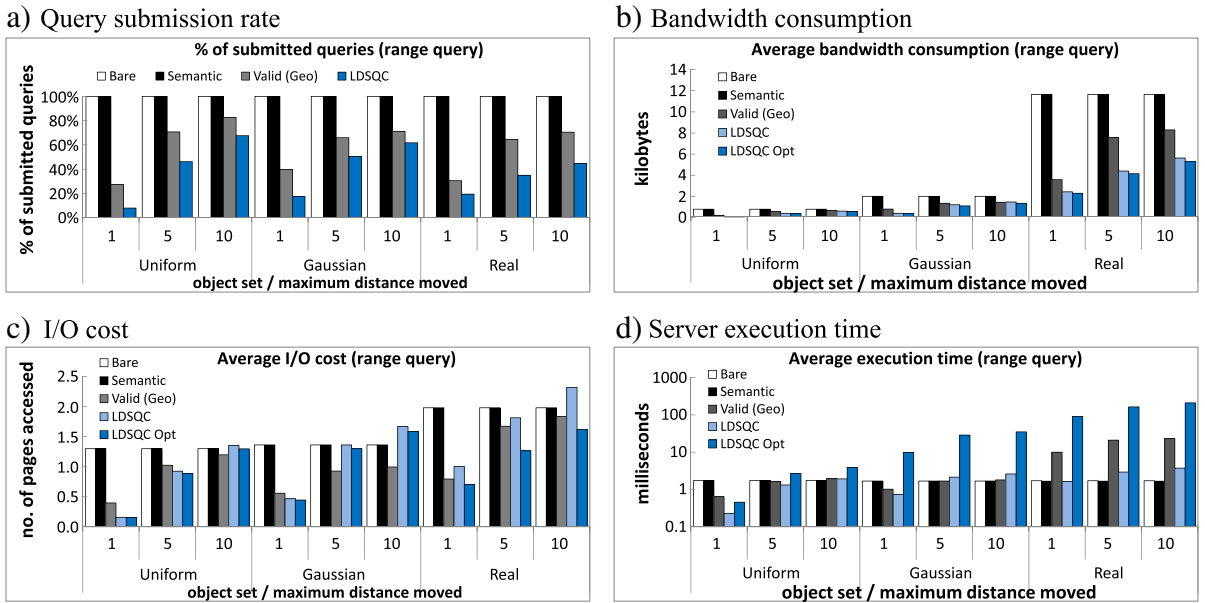


Fig. 22. Performance evaluation on fixed range queries ($r=15$).

does not support range queries is not included. In general, as shown in Fig. 22(a), Bare submits all queries to the server so its query submission rate reaches 100%. Semantic asserts the reuse of an LDSQ result only when the new search area is bounded by a semantic region. However, in our evaluation, clients are moving and they issue queries at different locations. As the query range is fixed, no LDSQ result is reusable which explains why it results in 100% query submission rate. Consequently, neither Bare nor Semantic can help moving clients to reuse previous LDSQ results. On the contrary, Valid (Geo) and LDSQC enable a better reuse of a previous LDSQ result to answer new LDSQs. As shown in the figure, they significantly reduce the query submission rates. When the maximum distance moved, tD , increases, the likelihood that new LDSQs are covered by previous LDSQs reduces, and hence the query submission rate increases.

The result of bandwidth consumption is shown in Fig. 22(b). Due to their high query submission rates, both Bare and Semantic incur the largest bandwidth consumption among all the evaluated approaches. LDSQC consumes less bandwidth than Valid (Geo) as a direct result of its lower query submission rate. Besides, Fig. 22(c) and (d) shows the result in terms of average I/O cost and execution time incurred by evaluation of server received queries. As Bare and Semantic do not derive any auxiliary scopes at the

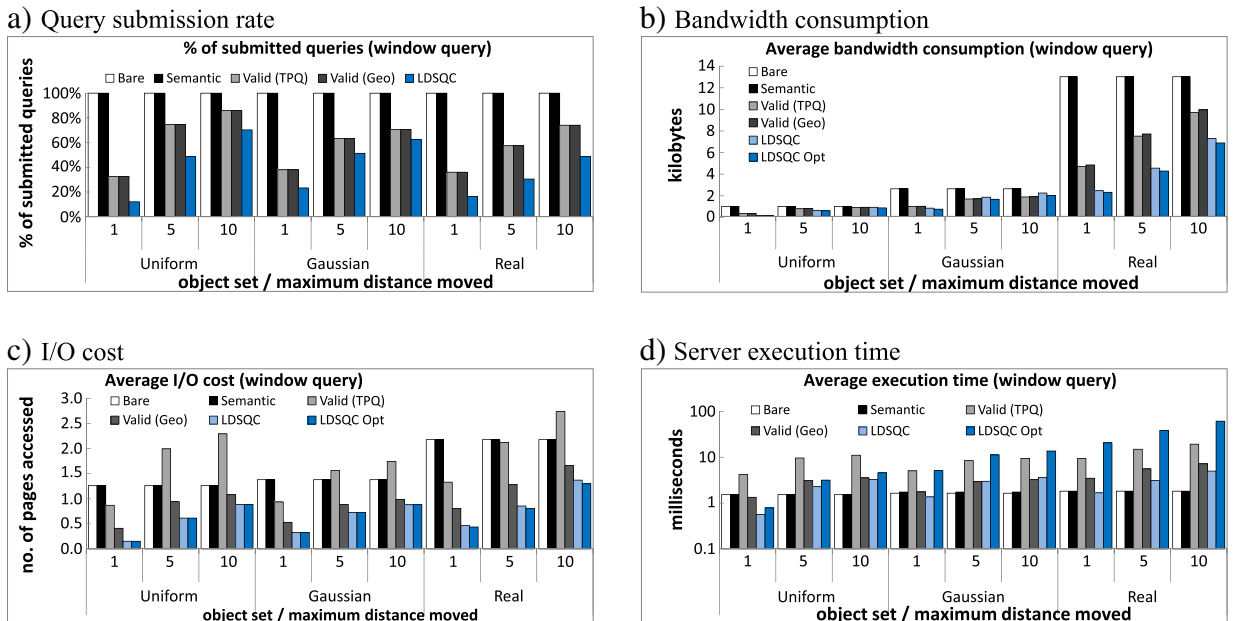


Fig. 23. Performance evaluation on fixed window queries ($l=15$).

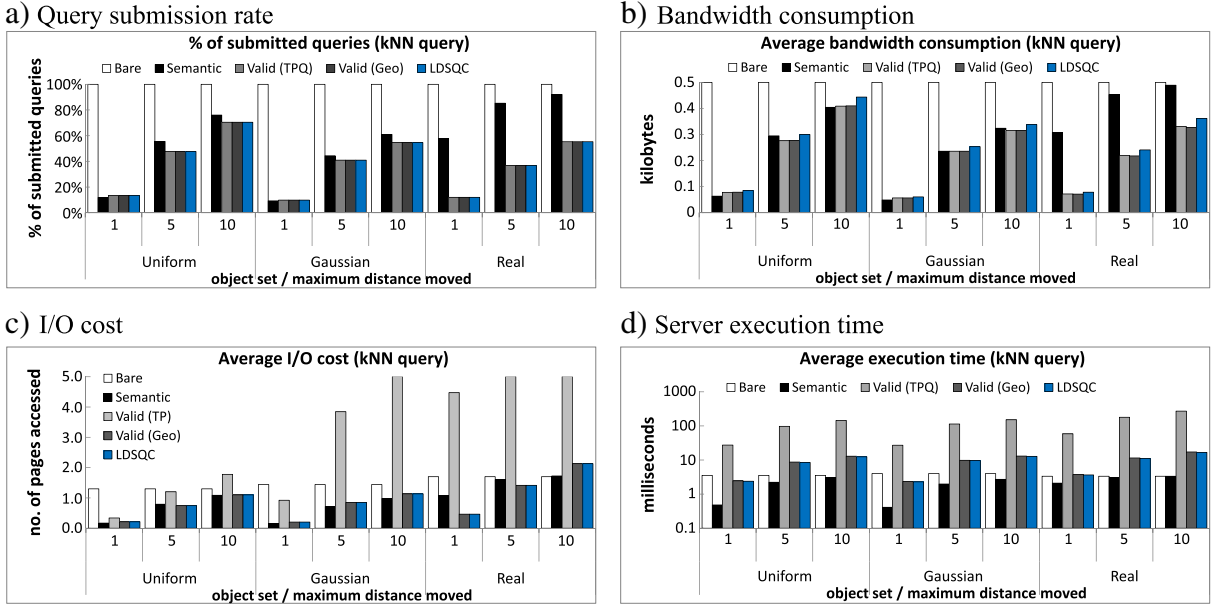


Fig. 24. Performance evaluation on fixed kNN queries ($k=4$).

server, their processing and I/O costs are the minimal as anticipated. On the other hand, Valid (Geo) and LDSQC incur longer execution times and higher I/O costs. In addition, LDSQCOpt incurs even longer execution time because of computationally expensive removable complementary object detection. To identify the extra cost incurred by filtering removable complementary objects, we can compare it with LDSQC. As shown in Fig. 22(d), LDSQC can considerably shorten the execution time while incurring only a few (about 1) extra page accesses and slightly extra bandwidth consumed (due to the small spatial coordinate sizes of non-result objects). As such, we can see that no removable complementary object filtering is an efficient alternative to determine containment scopes. Although more extra I/O costs incurred for each single query, LDSQC reduces server processing time. This improvement becomes huge for large populations of mobile clients.

Second, we investigate the system performance improvement for window queries, with the window side length l fixed at 15. Here, Valid (TPQ) is included. Basically, the experiment results as shown in Fig. 23 are very similar to those of range query and can be explained with similar reasons. We only highlight that despite Valid (TPQ) and Valid (Geo) provide identical valid scopes, thus resulting in the same query submission rate, they result in totally different processing and I/O costs. Valid (TPQ) evaluates a large number of TP window queries, leading to higher processing and I/O costs than Valid (Geo), which needs only one index lookup. Again, for window query, LDSQCOpt is the best in terms of query submission rate and bandwidth consumption. LDSQC speeds up the processing cost at a small expense of I/O cost and bandwidth consumption.

The last part of the experiment set evaluates the performance gain for kNN query where k is set to 4. The results are shown in Fig. 24. Semantic for kNN query is based on mNN query (where $m = k + 1$), so in this experiment it supports the reuse of mNN query results for new kNN queries issued nearby, and it shows an improvement in the query submission rate in Fig. 24(a). However, it is still worse than Valid (TPQ), Valid (Geo) and LDSQC, which derive more precise auxiliary scopes. As explained, Valid (TPQ) incurs very high processing and I/O costs. With a fixed extitk, containment scopes and valid scopes offer the same basic functionality and consequently provide the same query submission rate. When query parameters are varied, a lot more new queries can reuse the information maintained in containment scope. Thus, LDSQC can outperform both Valid (TPQ) and Valid (Geo) as will be presented next.

7.3. Exp. 2. Performance evaluation on varied query parameters

The second experiment set investigates the effect of varied query parameters on the system performance for all the evaluated approaches. Here, clients randomly pick query parameters (see Table 3) independently of previous LDSQs. As anticipated and in contrast to the previous experiment set, the results of LDSQs with different query parameters may be covered by a previous LDSQ result. Valid (TPQ) and Valid (Geo), which only support the reuse of results based on result equality, suffer in this experiment. Besides, Semantic and LDSQC, which can detect if the result of one LDSQ is contained by a previous one, are expected to outperform the others when query parameters can vary.

From Fig. 25(a), we can observe that Valid (TPQ) (for window and kNN queries only) and Valid (Geo) incur higher submission rates than Semantic and LDSQC as expected. Semantic in general incurs a higher submission rate than LDSQC (including LDSQCOpt). That implies LDSQC and LDSQCOpt are relatively more effective. As the consequence of having the lowest query submission rate, LDSQC and LDSQCOpt incur the lowest bandwidth consumption as shown in Fig. 25(b). On the other hand, LDSQC and LDSQCOpt have high

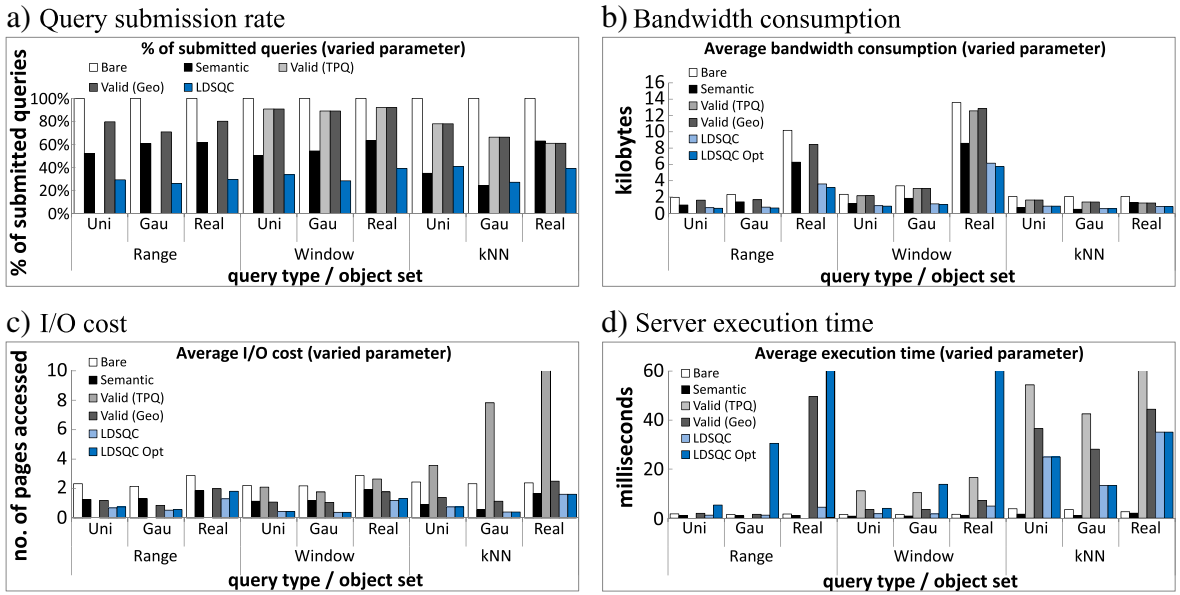


Fig. 25. Impact of client mobility with varied query parameters.

computational and I/O costs as indicated in Fig. 25(c) and (d). Since LDSQC incurs shorter processing time with slightly more bandwidth than LDSQCOpt, we consider LDSQC though including removable complementary objects as the better alternative to LDSQCOpt.

7.4. Exp. 3. Performance evaluation on object density

In the third and final experiment set, we investigate the system performance against various object densities. As the object density has a direct impact on the average distances between objects, it in turn affects the auxiliary scope size. In these experiments, we vary the cardinalities of objects that are uniformly distributed on the same service area between 1 k, 10 k and 100 k. The experiment results shown in Fig. 26 are obtained by the same settings as Exp. 2(a) while the maximum distance that clients can move is fixed at 5.

While the object density increases, the effectiveness of all the approaches is reduced as reflected by their query submission rate (as shown in Fig. 4). This is caused by the diminished auxiliary scope sizes. The bandwidth consumption increases as depicted in

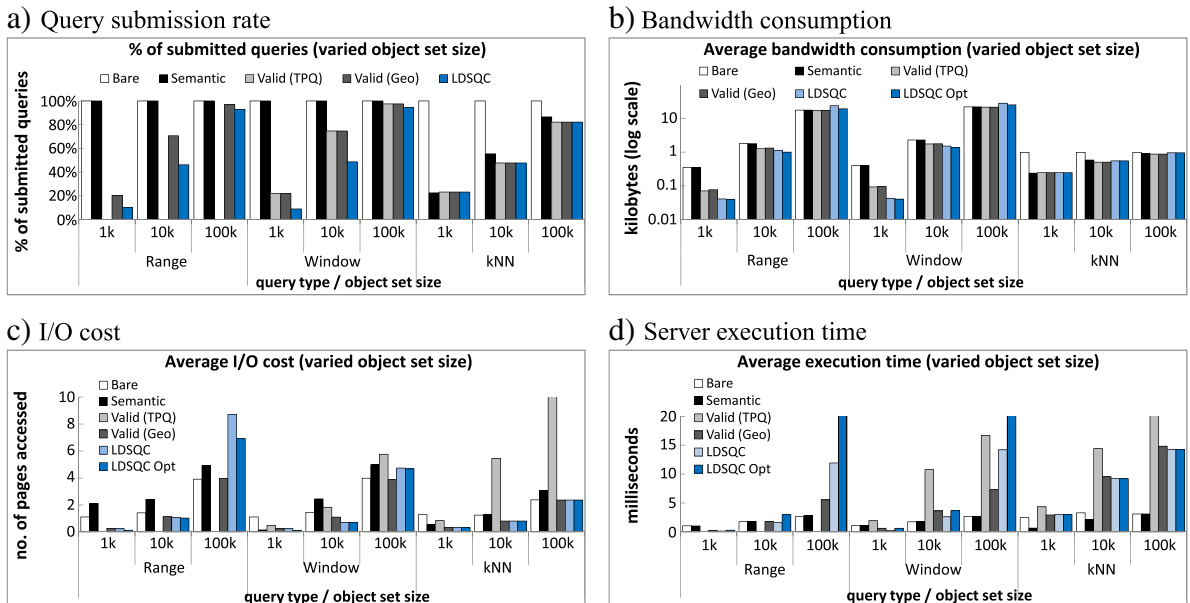


Fig. 26. The impact of object density on the query performance.

Fig. 26(b). On the server side, the increased number of objects causes a rise of both the I/O cost and the execution time as plotted in Fig. 26(c) and (d), respectively. Finally, in general, LDSQC still outperforms all the other approaches.

In summary, LDSQC performs the best among all the evaluated approaches for both fixed and varied query parameters. This is primarily because it makes every client capable of determining if a new LDSQ is covered by the result of previous LDSQs. Also, LDSQC can clearly improve server execution time while incurring only a little extra bandwidth consumption and I/O cost, compared with LDSQOpt. As a result, LDSQC is determined to be the best approach of supporting LDSQs on stationary objects in mobile and wireless environments.

8. Conclusion

LBSs have been receiving much attention and fostering a large commercial market. Generally speaking, in an LBS system, mobile clients access location-related information (such as local news, traffic news, tourist guides, points of interest, etc.) as a collection of stationary objects from an LBS server through issuing LDSQs with respect to their locations. On the other hand, mobile environments are very resource limited. As such, consumption of client energy and wireless bandwidth, and LBS server workload should be alleviated. To achieve this, we, in this paper, have introduced a new concept named *LDSQ Containment* and developed the notion of *containment scope*, accordingly. In summary, we have made the following six original and important contributions:

1. As the first attempt to explore both the semantics of LDSQs and the knowledge of object distributions, we have developed and introduced the new concept of *Location-Dependent Spatial Query Containment* (or LDSQ containment), which can effectively determine whether an LDSQ can be answered with the results of previous queries, thereby eliminating unnecessary LDSQs to the server, shortening query response time, and reducing client energy consumption and bandwidth contention.
2. We have presented an enhanced LBS system model that supports LDSQ containment. We have also developed a notion of *containment scope* that represents a spatial area associated with an LDSQ result set R wherein any new LDSQ Q' has a result set R' fully covered by R . Further, we have devised LDSQ containment test algorithms based on containment scope (that is composed of result objects and complementary objects' spatial coordinates) to determine if any new LDSQ result is fully covered by the previous one for clients.
3. We have devised efficient on-line containment scope computation algorithms for range, window and k NN queries. Our computation algorithms integrate containment scope computations with LDSQ processing to optimize server processing overhead.
4. We have provided optimized containment scopes for range and window queries to eliminate removable complementary objects, so as to reduce the bandwidth consumption and client storage overhead.
5. We have analyzed the effectiveness of LDSQ containment in terms of the performance gains and the overheads.
6. We have conducted extensive empirical experiments to evaluate system performance in comparison with existing techniques designed for reusing previous LDSQ results. In general, LDSQ containment is shown to outperform existing approaches under many circumstances.

We believe that our proposed LDSQ containment concept will lead us to other advanced data management and query processing techniques for LDSQs and LBSs in the future. Currently, we are extending our effort to elaborate the concept to support client answering LDSQs with multiple LDSQ results and support LDSQ containment for other complex LDSQs.

References

- [1] ABIResearch, Mobile Location Based Services (Research Report), <http://www.abiresearch.com/research/1003335-Mobile+Location+Based+Services>.
- [2] ARM, SA-1100 Microprocessor Product Brief, http://bwrc.eecs.berkeley.edu/Research/Pico_Radio/Test_Bed/Hardware/Documentation/ARM/processor_arm_SA1100-productbrief.pdf.
- [3] R. Benetis, C.S. Jensen, G. Karcauskas, S. Saltenis, Nearest and reverse nearest neighbor queries for moving objects, *VLDB Journal* 15 (3) (2006) 229–249.
- [4] S. Berchtold, C. Böhm, D.A. Keim, F. Krebs, H.-P. Kriegel, On Optimizing Nearest Neighbor Queries in High-Dimensional Data Spaces, *Proceedings of 8th International Conference on Database Theory (ICDT'01)*, 2001, pp. 435–449.
- [5] D. Calvanese, G.D. Giacomo, M. Lenzerini, Conjunctive query containment and answering under description logic constraints, *ACM Transactions on Computational Logics* 9 (3) (2008) 1–31.
- [6] D. Calvanese, G.D. Giacomo, M. Lenzerini, M.Y. Vardi, View-Based Query Containment, *Proceedings of the 22nd ACM Symposium on Principles of Database Systems (PODS'03)*, 2003, pp. 56–67.
- [7] S. Chen, B.C. Ooi, Z. Zhang, An Adaptive Updating Protocol for Reducing Moving Object Databases Workload, *Proceedings of International Conference on Very Large Databases (VLDB'10)*, 2010, pp. 735–746.
- [8] S. Dar, M.J. Franklin, B.T. Jónsson, D. Srivastava, M. Tan, Semantic Data Caching and Replacement, *Proceedings of 22th International Conference on Very Large Data Bases (VLDB'96)*, 1996, pp. 330–341.
- [9] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer Verlag, 2000.
- [10] U. Demiryurek, F.B. Kashani, C. Shahabi, Efficient continuous nearest neighbor query in spatial networks using euclidean restriction, *Proceedings of the 11th International Symposium on Advances in Spatial and Temporal Databases (SSTD'09)*, 2009, pp. 25–43.
- [11] B. Gedik, L. Liu, *MobiEyes: a distributed location monitoring service using moving location queries*, *IEEE Transactions Mobile Computing* 5 (10) (2006) 1384–1402.
- [12] B. Gedik, L. Liu, *Protecting location privacy with personalized k-Anonymity: architecture and algorithms*, *IEEE Transactions on Mobile Computing* 7 (1) (2008) 1–18.
- [13] G. Grahne, A. Thomo, Query Containment and Rewriting using Views for Regular Path Queries under Constraints, *Proceedings of the 22nd ACM Symposium on Principles of Database Systems (PODS'03)*, 2003, pp. 111–122.
- [14] R.H. Güting, T. Behr, C. Düntgen, *SECONDO: a platform for moving objects database research and for publishing and integrating research implementations*, *IEEE Data Engineering Bulltin* 33 (2) (2010) 56–63.
- [15] G.R. Hjaltason, H. Samet, Distance browsing in spatial databases, *ACM Transactions on Database Systems* 24 (2) (1999) 265–318.

- [16] H. Hu, J. Xu, D.L. Lee, A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects, Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'05), 2005, pp. 479–490.
- [17] C.-C. Hung, W.-C. Peng, A regression-based approach for mining user movement patterns from random sample data, Data & Knowledge Engineering 70 (1) (2011) 1–20.
- [18] S. Ilarri, E. Mena, A. Illarramendi, Location-dependent query processing: where we are and where we are heading, ACM Computing Survey 42 (3) (2010).
- [19] G.S. Iwerks, H. Samet, K.P. Smith, Continuous K-Nearest Neighbor Queries for Continuously Moving Points with Updates, Proceedings of 29th International Conference on Very Large Data Bases (VLDB'03), 2003, pp. 512–523.
- [20] G. Kollios, D. Papadopoulos, D. Gunopulos, V.J. Tsotras, Indexing mobile objects using dual transformations, VLDB Journal 14 (2) (2005) 238–256.
- [21] K.C.K. Lee, W.-C. Lee, H.V. Leong, B. Unger, B. Zheng, Efficient Valid Scope Computation for Location-Dependent Spatial Query in Mobile and Wireless Environments, Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication (ICUIMC'09), 2009, pp. 131–140.
- [22] K.C.K. Lee, H.V. Leong, A. Si, Semantic query caching in a mobile environment, ACM SIGMOBILE Mobile Computing and Communications Review (MC2R) 3 (2) (1999) 28–36.
- [23] K.C.K. Lee, H.V. Leong, J. Zhou, A. Si, An Efficient Algorithm for Predictive Continuous Nearest Neighbor Query Processing and Result Maintenance, Proceedings of the 6th International Conference on Mobile Data Management (MDM'05), 2005, pp. 178–182.
- [24] K.C.K. Lee, J. Schiffman, B. Zheng, W.-C. Lee, Valid Scope Computation for Location-Dependent Spatial Query in Mobile Broadcast Environments, Proceedings of the 17th ACM International Conference on Information and Knowledge Management (CIKM'08), 2008, pp. 1231–1240.
- [25] Y. Manolopoulos, A. Nanopoulos, A. Papadopoulos, Y. Theodoridis, R-trees: Theory and Applications, Springer, 2005.
- [26] Oliver Kasten, Energy Consumption, http://www.inf.ethz.ch/personal/kasten/research/bathtub/energy_consumption.html.
- [27] J. O'Rourke, Computational Geometry in C, Cambridge University Press, 1998.
- [28] P. Pesti, L. Liu, B. Bamba, A. Iyengar, M. Weber, Roadtrack: scaling location updates for mobile clients on road networks with query awareness, PVLDB 3 (2010) 1493–1504.
- [29] S. Prabhakar, Y. Xia, D.V. Kalashnikov, W.G. Aref, S.E. Hambrusch, Query indexing and velocity constrained indexing: scalable techniques for continuous queries on moving objects, IEEE Transactions on Computers 51 (10) (2002) 1124–1140.
- [30] Q. Ren, M.H. Dunham, V. Kumar, Semantic caching and query processing, IEEE Transactions on Knowledge and Data Engineering (TKDE) 15 (1) (2003) 192–210.
- [31] N. Roussopoulos, S. Kelley, F. Vincent, Nearest Neighbor Queries, Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'95), 1995, pp. 71–79.
- [32] I. Sergio, M. Eduardo, I. Arantza, Location-dependent query processing: where we are and where we are heading, ACM Computing Surveys 42 (2010) 1–173.
- [33] Z. Song, N. Roussopoulos, K-Nearest Neighbor Search for Moving Query Point, Proceedings of 7th Symposium on Advances in Spatial and Temporal Databases (SSTD'01), 2001, pp. 79–96.
- [34] D. Stojanovic, A.N. Papadopoulos, B. Predic, S. Djordjevic-Kajan, A. Nanopoulos, Continuous range monitoring of mobile objects in road networks, Data & Knowledge Engineering 64 (1) (2008) 77–100.
- [35] Y. Tao, D. Papadias, Time-Parameterized Queries in Spatio-Temporal Databases, Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'02), 2002, pp. 334–345.
- [36] Y. Tao, D. Papadias, Q. Shen, Continuous Nearest Neighbor Search, Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02), 2002, pp. 287–298.
- [37] Y. Theodoridis, E. Stefanakis, T.K. Sellis, Efficient cost models for spatial queries using R-trees, IEEE Transactions on Knowledge and Data Engineering 12 (1) (2000) 19–32.
- [38] U.S. Census Bureau, Topologically Integrated Geographic Encoding and Referencing System - TIGER/Line[web], <http://www.census.gov/geo/www/tiger/> 2006.
- [39] H. Wang, R. Zimmermann, A novel dual-index design to efficiently support snapshot location-based query processing in mobile environments, IEEE Transactions on Mobile Computing 9 (9) (2010) 1280–1292.
- [40] Z. Xu, H.-A. Jacobsen, Expressive location-based continuous query evaluation with binary decision diagrams, Proceedings of the 25th International Conference on Data Engineering (ICDE'09), 2009, pp. 1155–1158.
- [41] J. Zhang, M. Zhu, D. Papadias, Y. Tao, D.L. Lee, Location-based Spatial Queries, Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'03), 2003, pp. 443–454.
- [42] L. Zhao, N. Jing, L. Chen, Z. Zhong, A Novel Framework for Processing Continuous Queries on Moving Objects, Proceedings of 11th International Conference on Web-Age Information Management (WAIM'10), 2010, pp. 321–332.
- [43] B. Zheng, W.-C. Lee, D.L. Lee, On semantic caching and query scheduling for mobile nearest-neighbor search, Wireless Networks 10 (6) (2004) 653–664.
- [44] B. Zheng, W.-C. Lee, K.C.K. Lee, J. Winter, M.-C. Chen, DISQO: A Distributed Framework for Spatial Queries over Moving Objects, Proceedings of International Conference on Parallel Processing (ICPP'10), 2010, pp. 414–423.



Ken C.K. Lee obtained his Ph.D. degree in Computer Science and Engineering from the Pennsylvania State University, University Park in 2009. Prior to starting his Ph.D. study in 2004, he earned his bachelor and master degrees both in Computing, in the Hong Kong Polytechnic University, Hong Kong. He is now an assistant professor of Computer and Information Science at the University of Massachusetts Dartmouth. His research interests include location-based services, mobile and wireless computing, spatial databases, spatial networks, query processing and index, etc. He is a member of the IEEE and the ACM.



Brandon Unger currently is a software development engineer on the Microsoft SQL Server team and is responsible for implementing various components in Microsoft's business intelligence platform. Most of his work is related to core storage engine and formula engine technology adopted in several Microsoft products that include Analysis Services and PowerPivot for Excel. Prior to joining Microsoft, Brandon completed a B.S. degree in Computer Science, a B.S. degree in Mathematics, and an M.S. degree in Computer Science and Engineering all at the Pennsylvania State University.



Baihua Zheng received a Ph.D. degree in Computer Science from Hong Kong University of Science and Technology. She is a member of the IEEE and the ACM. Currently, she is an associate professor in the School of Information Systems at Singapore Management University. Her research interests include mobile and pervasive computing and spatial databases.



Wang-Chien Lee received the BS degree from the Information Science Department, National Chiao Tung University, Taiwan, the MS degree from the Computer Science Department, Indiana University, and the Ph.D. degree from the Computer and Information Science Department, the Ohio State University. He is an associate professor of computer science and engineering at The Pennsylvania State University. Prior to joining Penn State, he was a principal member of the technical staff at Verizon/GTE Laboratories, Inc. He leads the Pervasive Data Access (PDA) Research Group at Penn State University to pursue cross-area research in database systems, pervasive/mobile computing, and networking. He is particularly interested in developing data management techniques (including accessing, indexing, caching, aggregation, dissemination, and query processing) for supporting complex queries in a wide spectrum of networking and mobile environments such as peer-to-peer networks, mobile ad hoc networks, wireless sensor networks, and wireless broadcast systems. Meanwhile, he has worked on XML, security, information integration/retrieval, and object-oriented databases. He has published more than 160 technical papers on these topics. His research has been supported by multiple US National Science Foundation (NSF) grants. Most of his research results have been published in prestigious journals and conferences in the fields of databases, mobile computing, and networking. He has been active in various IEEE/ACM conferences and has given tutorials for many major conferences. He was the founding program co-chair of the International Conference on Mobile Data Management (MDM). He

has also served as a guest editor for several journal special issues (e.g., IEEE Transactions on Computers) on mobile database related topics. He has served as the TPC chair or general chair for a number of conferences, including the Second International Conference on Scalable Information Systems (Infoscale07), the Sixth International ACM Workshop on Data Engineering for Wireless and Mobile Access (MobiDE07), the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC08). He is currently serving as the TPC chair for the Tenth International Conference on Mobile Data Management (MDM09). He is a member of the IEEE Computer Society.