Ordered Reverse *k* Nearest Neighbor Search via On-demand Broadcast

Li Li[1], Guohui Li[1], Quan Zhou[1] and Yanhong Li[2]
[1] School of Computer Sci. & Tech, Huazhong University of Sci. & Tech
Wuhan, Hubei - China
[e-mail: *xiangpenpende@gmail.com*]
[2] College of Computer Science, South-Central University for Nationalities
Wuhan, Hubei - China
[e-mail: *anddylee@163.com*]
*Corresponding author: Guohui Li

Received April 23, 2014; revised June 25, 2014; revised August 21, 2014; accepted September 13, 2014; published November 30, 2014

Abstract

The Reverse *k* Nearest Neighbor (R*k*NN) query is valuable for finding objects influenced by a specific object and is widely used in both scientific and commercial systems. However, the influence level of each object is unknown, information that is critical for some applications (e.g. target marketing). In this paper, we propose a new query type, Ordered Reverse *k* Nearest Neighbor (OR*k*NN), and make efforts to adapt it in an on-demand scenario. An Order-*k* Voronoi diagram based approach is used to answer OR*k*NN queries. In particular, for different values of *k*, we pre-construct only one Voronoi diagram. Algorithms on both the server and the clients are presented. We also present experimental results that suggest our proposed algorithms may have practical applications.

Keywords: on-demand broadcast, ordered reverse *k* nearest neighbor query, ordered order-*k* Voronoi diagram

http://dx.doi.org/10.3837/tiis.2014.11.013

1. Introduction

A reverse *k* nearest neighbor (R*k*NN) query [1-11] returns objects that take a query object as one of its top *k* closest neighbors. In the past decade, the R*k*NN query has received increasing research attention and has had a large impact on a broad range of applications, including strategic planning, resource allocation, online reality games, mobile navigation systems, decision support systems and profile-based marketing. The main drive behind many R*k*NN applications is determining the degree of influence that a query object has on some other objects. Generally, the query object bears an influence over every object in the result set of a R*k*NN query. However, in almost all existing studies, the answer objects in the R*k*NN result set are treated equally even though they are affected by the query object to various degrees.

With respect to a R*k*NN query, we identified various applications for which there is a need to obtain an influence rank for the answer objects. A representative example is targeted selling, which is becoming increasingly popular. Super markets that adopt the targeted selling strategy could increase promotional effectiveness by placing targeted advertisements in the most efficient locations. Consequently, it is of great importance to rank the influences of the different communities sourced by the super market. In addition, in order to schedule advertising campaigns accordingly, a gas station manager would need to identify the influence rank of the vehicle and whether or not it is located in the gas station's influence zone. Consider another scenario in which players of computer games tend to attack the enemy that is nearest to him. To avoid potential dangers, the best strategy would be to sequentially kill the enemies that take him as the 1^{st}, 2^{nd}, …… closest enemy neighbors, separately.

However, ordinary R*k*NN queries are not appropriate for such application because the answer objects are unordered. To tackle the problems arising from these practical applications, we introduce a new query type, namely ordered reverse *k* nearest neighbors (OR*k*NN) query. Compared with the original R*k*NN search, the OR*k*NN returns the same set of result objects but with more information. For each returned object, the OR*k*NN returns the influence of the query object. The ranked RNN query [1] retrieves *t* data objects most influenced by the query object. While this query is similar to the OR*k*NN search, it is not a substitute for OR*k*NN queries. For example, in the applications of targeted selling, a ranked query may return objects that are weakly influenced by the user, while an object that is strongly affected by the user may not be returned when *t* is not large enough.

Wireless technologies are widely used around the world. More and more, users prefer wireless networks to cable and fiber, owing to the flexible, cost-saving nature of wireless communication. Therefore, techniques toward wireless network are continuously proposed [12-16]. Using wireless broadcast, a broadcasted data item can be shared by a number of clients, leading to a more efficient use of shared bandwidth. There are two primary approaches to accessing data via wireless broadcast: on-demand broadcast and push-based broadcast. Push-based broadcasts are useful for applications that have relatively stable access patterns. However, on-demand broadcast is more efficient at handling time-critical queries with dynamic data access patterns, which makes on-demand broadcast quite appropriate for our proposed OR*k*NN applications.

In this paper, we focus on processing the OR*k*NN queries in on-demand broadcast environments for fixed query clients and moving objects of interest. To the best of our knowledge, we present the first effort toward studying the OR*k*NN search. We first present a method for computing OR*k*NN results by utilizing the pre-computed ordered order-*K* Voronoi

diagram (OO*K*VD) [17], where *K* is the maximal possible value of *k*. Only one Voronoi diagram needs to be pre-constructed, which greatly reduces the pre-construction overhead. We then present the algorithms to the server and clients. Data items may dynamically change in our model, preventing existing broadcast scheduling algorithms to be directly applied to deliver OR*k*NN results via on-demand broadcast to the clients. Consequently, we develop a novel scheduling method on the server to solve this problem. A priority computation technique is used to assign priorities to pending requests before scheduling them.

Our main contributions in this paper can be summarized as follows.

1. We propose a new query type termed ordered reverse *k* nearest neighbors (OR*k*NN) query, which has many practical applications (e.g. targeted selling). For each returned object, OR*k*NN ranks the influence of the query object.

2. We propose an order-*k* Voronoi diagram based approach to process OR*k*NN queries via on-demand broadcast. Although the OR*k*NN queries may be submitted with different values of *k*, we only pre-constructed one Voronoi diagram, greatly reducing the total computational burden.

3. We develop a novel broadcast scheduling method to deliver OR*k*NN results via on-demand broadcast to the clients. A priority computation technique is used to assign priorities to the pending requests and data items.

4. We conduct extensive experiments to evaluate the proposed algorithms and illustrate their affectivity and efficiency.

We organize the remainder of the paper as follows. Related work is provided in Section 2. In Section 3, we describe some preliminaries including our proposed OR*k*NN query and the broadcast model. Section 4 presents the algorithm running on the server, and Section 5 presents the client-side algorithm. Section 6 shows the performance evaluation of our algorithms via simulation experiments. Section 7 concludes the paper.

2. Related Work

In this section, we briefly review the previous studies related to our work in the RNN research area and in the field of on-demand broadcast scheduling.

2.1 RNN Queries and Variants

Since the first introduction of RNN in [2], the RNN query problem has been extensively studied [1-10] and a range of variants have been proposed. The RNN ranking query sorts the objects of the database according to the number of other, more similar objects in the database, without the need to restart the search from scratch [11]. In the continuous reverse nearest neighbor (CRNN) queries [7, 9], a monitoring region of a continuous query that enables the possibility of incremental processing is maintained. Previous studies have handled the RNN queries for moving objects, but not in wireless broadcast environments [10, 18]. A variant of RNN query, called ranked RNN query, is also proposed [1]. To the best of our knowledge, the ranked RNN query is the most similar method to the proposed OR*k*NN query, but has several distinct characteristics. The ranked RNN retrieves *t* data objects most influenced by the query object, where *t* is specified at the query time to limit the size of result sets, thus the value of *k* is indeterminate and the size of the result set is fixed. For an OR*k*NN query, the value of *k* is fixed and determined when the request is issued. However, the number of returned objects is variable. Further, the algorithm for processing ranked RNN queries in [19] is not assumed in wireless broadcast environments.

Several approaches have been proposed in previous works to processing RNN queries under wireless periodic broadcast environments. In this field, a preliminary study is conducted [20], where three air indexing techniques (naive air index, Rdnn-tree air index and D-tree air index) are employed and corresponding algorithms based on these techniques are devised. A structure called Jump Rdnn-tree is proposed to process RNN query in the broadcast environment [21]. A Jump Rdnn-tree is constructed by adding additional link pointers to the traditional Rdnn-tree. Using Jump Rdnn-tree, the backtracking problem is eliminated, and thus the air index is more suitable for linear access under a wireless broadcast environment. An RNN search protocol used in data broadcast environments is proposed [10, 18]. In this protocol, no index structure is needed.

2.2 Data Broadcast Algorithms

Data broadcast is a large research area, and we introduce a few of the most relevant studies here. Various scheduling algorithms have been proposed to determine the broadcast sequence of data items in on-demand broadcast environments [22-28]. A data selection and scheduling algorithm using a modified EDF is proposed in [27]. The scheduling algorithm termed SIN-α [22] takes into accounts both timelines and the number of data requests. A number of data productivity-based scheduling algorithms are extended in [29]. Additionally, their performances for scheduling multi-item requests in multi-channel broadcast environments are evaluated. However, these algorithms mentioned above are not approximate enough for application in the OR*k*NN query processing, where the requests are time-critical and require multiple items at the same time.

Algorithms presented in [24] and [25] (called DPA and DTIU, respectively) are well designed to schedule multi-item requests under time constraints. Although they are effective and efficient for scheduling static objects, direct applying of DPA and DTIU in our OR*k*NN applications would lead to request response errors because data items may change in our model. Wang et al. [30] investigated the broadcast scheduling problems associated with disseminating timely data to periodic continuous queries and proposed an efficient online scheduling algorithm, called RM-UO. They assume that data items accessed by a query would not vary during their lifetime and that the requests are invoked periodically. RM-UO is a systematic and highly efficient solution for multiple practical applications. Unfortunately, these assumptions, especially the former, make RM-UO inadequate for processing our proposed OR*k*NN queries.

3. Preliminaries

3.1 OOkVD

OO*k*VD, short for ordered order-*k* Voronoi diagram [17], is a variant of the ordinary Voronoi diagram. An OO*k*VD tessellates the space into mutually exclusive and collectively exhaustive Voronoi cells. Each cell is characterized by an ordered sequence of generator sites.

According to the definition of OO*k*VD [17], every OO*k*VD cell is characterized with an ordering of the *k* nearest neighboring generators associated to it. The *k* nearest neighboring generators are organized as a *k*-tuple. This arrangement can distinguish among the first, the second, and up to the k^{th} nearest neighboring generator for all locations in each cell. In this paper, we use the corresponding *k*-tuple to identify an OO*k*VD cell. When *k=1*, OO*k*VD is the ordinary Voronoi diagram.

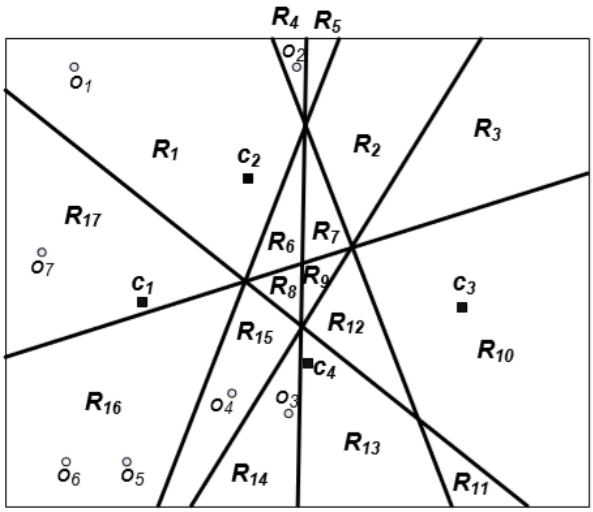


Fig. 1. An example of OO*k*VD

An example of OO*k*VD is shown in **Fig. 1**, where k is 3, $C=\{c_1, c_2, c_3, c_4\}$ is the set of generators (clients), $O=\{o_1, \ldots, o_7\}$ is the set of interest objects moving inside the OO*k*VD space and the symbols $R_1, \ldots, R_{17}$ are used to represent the OO3VD cells. The k-tuples used to identify the OO*k*VD cells (R_1-R_{17}) are shown in **Table 1**. The k-tuple (c_2, c_1, c_4) corresponding to R_1 means the first nearest neighboring generator to any point p in R_1 is c_2, the second nearest is c_1, and the third is c_4.

Table 1. Triples associated to the OO*k*VD cells

OO*k*VD Cell	*k*-tuple	OO*k*VD Cell	*k*-tuple	OO*k*VD Cell	*k*-tuple
R_1	(c_2, c_1, c_4)	R_7	(c_2, c_4, c_3)	R_{13}	(c_4, c_3, c_1)
R_2	(c_2, c_3, c_4)	R_8	(c_4, c_2, c_1)	R_{14}	(c_4, c_1, c_3)
R_3	(c_3, c_2, c_4)	R_9	(c_4, c_2, c_3)	R_{15}	(c_4, c_1, c_2)
R_4	(c_2, c_1, c_3)	R_{10}	(c_3, c_4, c_2)	R_{16}	(c_1, c_4, c_2)
R_5	(c_2, c_3, c_1)	R_{11}	(c_3, c_4, c_1)	R_{17}	(c_1, c_2, c_4)
R_6	(c_2, c_4, c_1)	R_{12}	(c_4, c_3, c_2)		

3.2 Definition of OR*k*NN Queries

An OR*k*NN query returns every object of interest that considers the query object as one of their top k closest neighbors, as well as the degree of influence.

Definition 1 (Influence degree): Let O be the set of the objects of interest. The influence degree of an interest object $o \in O$ indicates how much o is influenced by the query object. The influence degree of o is set to be i if the query object is the i^{th} closest neighboring site of o.

The smaller the influence degree is, the more influence the query object has over the object of interest, and vice versa. The concept of influence degree is also used in [1].

Definition 2 (OR*k*NN query): Let C be the set of clients, which may issue OR*k*NN queries. An OR*k*NN query is characterized by a client $c \in C$ and k. An OR*k*NN query returns a set of pairs (o, i), where o is an interest object and i is the corresponding influence degree. The result set of the OR*k*NN query issued by a client $c \in C$ is denoted as OR*k*NN(c).

3.3 Broadcast Model

In this section, we present the model for processing OR*k*NN queries via on-demand broadcast. The on-demand broadcast environment is suitable for handling OR*k*NN queries for the following two reasons: (1) A broadcasted data item can be shared by several different requests. For example, in **Fig. 1**, o_5 is requested by c_1 and c_4 at the same time; (2) The OR*k*NN queries are time-critical since the interest objects may move out of the influence region.

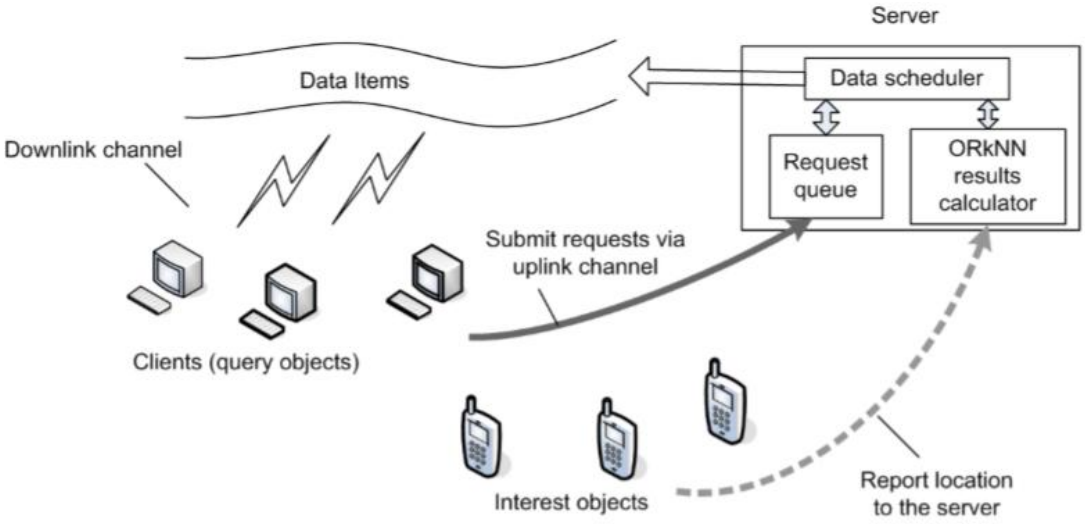


Fig. 2. Broadcast model

As illustrated in **Fig. 2**, the broadcast model consists of one server, a set of clients, a set of interest objects, and several wireless channels. The server calculates the OR*k*NN results and broadcasts them to the clients. The clients are assumed to be static (e.g. parking lot, gas station and shopping mall), and they submit OR*k*NN requests to the server through an uplink channel. Each request has a deadline beyond which the receipt of required data items has no value to the client. We assume that the clients would not issue a new request until the request being served is satisfied, or the deadline expires. Once a client sends a request to the server, the client listens to the download broadcast channel until the deadline expires or all needed data items have been received. The interest objects are assumed to have arbitrary movements and their positions and velocities are maintained in the server.

Table 2 lists the definitions of some symbols that will be frequently used later.

Table 2. Symbols and definitions

Symbol	Definition
O/C	The set of interest objects/clients
K	The order of the pre-computed Voronoi diagram
$o(c_{i1},\cdots,c_{iK})$	A data item
$(c_{i1},\cdots,c_{iK})$	A k-tuple contained in a data item used to identify an OO*k*VD cell
$S(q)$	The remaining lifetime of query q
OR*k*NN(c)	The result set of an OR*k*NN query issued by client c
q	An OR*k*NN request
RQ	The queue used for recording all pending requests

$BQ(q)$	The queue recording all broadcasted data items required by q
$UQ(q)$	The queue recording all un-broadcasted data items required by q
DR	The request drop ratio
SR	The scheduling efficiency ratio
N_q	The number of interest objects required by q
$broadcast\_slot$	The time used for broadcasting a single data item

4. Server Algorithm

In this section, we propose an OOkVD based method to process ORkNN queries. Using order-k Voronoi diagrams for processing RkNN queries has been considered infeasible due to the following limitations: the value of k is not known in advance and the computation of Voronoi diagrams for different k incurs high computational and spatial overhead costs. However, by further exploring new properties of OOkVD, it is only necessary to pre-compute one Voronoi diagram for the different k values, which greatly reduces the cost.

To pre-construct the Voronoi diagram, we first estimate the range of the values of k in ORkNN queries. Then we construct the OOKVD (k=K), taking the set of clients as generators, where K is the maximal possible value of k. Since the clients are assumed to be static, we do not consider the updates of OOKVD in this paper.

Generally, for each new submitted ORkNN query q, we first calculate the ORkNN results based on which data items needed by q are constructed. We then insert the obtained data items to be broadcasted into the Data Item Set. Finally, a broadcast scheduling method is adopted to broadcast the data items to clients. The process is shown in **Fig. 3**.

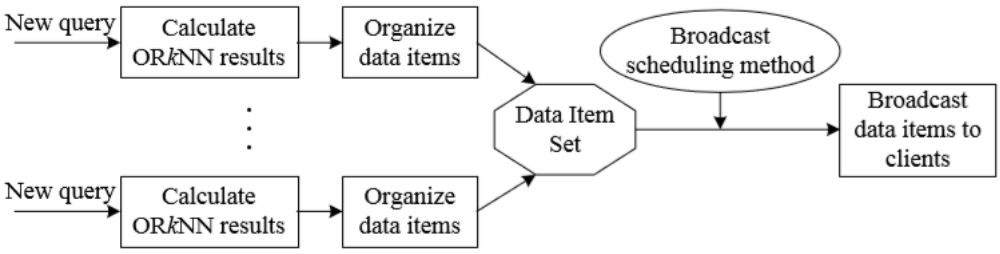


Fig. 3. Query processing on server

The maintenance of the Data Item Set involves many implementation details and the methodology used to broadcast data items through wireless channels is out of the scope of this paper. Instead, we focus on the ORkNN result calculation, data item organization and the broadcast scheduling method.

4.1 ORkNN Result Calculation

Before presenting how to calculate ORkNN results based on the pre-computed OOKVD, we first introduce two notions, $S(c_{i1},\cdots,c_{iK})$ and $S^j_{c_r}$.

Definition 3 ($S(c_{i1},\cdots,c_{iK})$): The set of interest objects that reside in the OOkVD cell with the K-tuple$(c_{i1},\cdots,c_{iK})$.

For example, in **Fig. 1**, where we assume K=3, we have $S(c_1, c_4, c_2)=\{o_5, o_6\}$, and $S(c_2, c_1, c_4)=\{o_1\}$.

Definition 4 ($S^j_{c_r}$): The set of interest objects, which take c_r as their j^{th} reverse nearest neighbors.

Based on Definition 3 and 4, we propose Lemma 1 as follows.

Lemma 1: $S^j_{c_r} = \bigcup_{c_{ij}=c_r} S(c_{i1}, \cdots, c_{iK})$.

Proof: This lemma can be proved directly according to the properties of OO*K*VD discussed in Section 3.2.

Based on Lemma 1 and the definition of OR*k*NN queries, we obtain OR*k*NN(c_r) for any $c_r \in C$, as shown in Lemma 2.

Lemma 2: OR*k*NN(c_r) = {($S^1_{c_r}$, *1*), $\cdots$, ($S^k_{c_r}$, *k*)}.

Proof: This lemma can be proved directly according to the definitions of $S^j_{c_r}$ and OR*k*NN queries.

According to Lemma 2, the set of interest objects required by the OR*k*NN query issued by $c_r \in C$ can be represented as $\bigcup_{j=1}^{k} S^j_{c_r}$.

Take **Fig. 1** as an example, OR*1*NN(c_1) = $\{(S^1_{c_1},1)\}$, and OR*2*NN(c_1) = $\{(S^1_{c_1},1), (S^2_{c_1},2)\}$, where by Lemma 1, $S^1_{c_1} = S(c_1, c_2, c_4) \cup S(c_1, c_4, c_2) = \{o_5, o_6, o_7\}$, and $S^2_{c_1} = S(c_2, c_1, c_4) \cup S(c_2, c_1, c_3) \cup S(c_4, c_1, c_3) \cup S(c_4, c_1, c_2) = \{o_1, o_2, o_3, o_4\}$.

Lemma 1 and Lemma 2 together make the Voronoi diagram based approach feasible, since we do not have to construct different Voronoi diagrams for different *k* values. In effect, we have created a mapping between the OO*K*VD cells and OR*k*NN(c_r). The OO*K*VD cells, in which the interest objects are required by $c_r \in C$, are called the influence cells of c_r. That means, every object of interest in the influence cells of c_r is in the result set of OR*k*NN(c_r).

Note that a query can be discarded directly after calculating the OR*k*NN results, without executing the following steps. Specifically, if there is not a required object of interest by *q*, or the remaining life of *q* is less than the time used to broadcast all un-broadcasted data items required by *q*, i.e., $S(q) < N_q \times broadcast\_slot$, then *q* is discarded immediately. Here, N_q is the number of interest objects required by *q*, and *broadcast_slot* is the time used for broadcasting a single data item.

4.2 Data Item Organization

In regular on-demand applications, a client is aware of the data items that are to be retrieved. However, in our model, the clients themselves are unaware of the OR*k*NN query results. Thus, the server should supply hints to guide the clients to fetch, or not fetch, a data item. To handle this issue, we organize data items in the form of $o(c_{i1}, \cdots, c_{iK})$, where $o \in O$ is an interest object and $(c_{i1}, \cdots, c_{iK})$ is the *K*-tuple associated with the OO*K*VD cell where *o* resides.

At each moment, a data item contains a unique interest object, since one interest object can be located in only one OO*k*VD cell at each moment. For presentation simplicity, we also use the symbol $\overline{o}$ to denote a data item $o(c_{i1}, \cdots, c_{iK})$.

The *K*-tuple in a data item carries two important messages: (1) which clients are requesting the data item; (2) the influence degree of the corresponding interest object. For example, data

item $o(c_{i1}, \cdots, c_{iK})$ is required by the clients $c_{i1}, \cdots, c_{iK}$, and for any client c_{ir} $(1 \le r \le K)$, the influence degree of o is r.

Since the clients do not know about the ORkNN results, one challenge is a client that all the wanted data items have been received, so that it can stop listening. A straightforward method would be to broadcast a special data item as the termination signal. This method is efficient in stopping the client, but inefficient for bandwidth utilization. We tackle this challenge by representing the client IDs in the K-tuples in various ways. Suppose each client ID consists of a set of English characters. We assign an integer number to each character without any repetition (e.g. a-z, A-Z correspond to *1-26*, *27-52*). If $o(c_{i1}, \cdots, c_{iK})$ is not the last un-broadcasted data item required by an ORkNN query issued by client c_{ir}, then each character of c_{ir} is represented by the corresponding number. Otherwise, the first character of c_{ir} is represented by adding *52* to its corresponding number. This means that a client c_{ir} could determine whether $o(c_{i1}, \cdots, c_{iK})$ is the last needed data item by checking the first character of c_{ir}. If the first character of c_{ir} is larger than *52*, $o(c_{i1}, \cdots, c_{iK})$ is the last needed data item. For example, *2* and *54* represent the same client ID 'b', and *54* also informs the client that it can stop listening after receiving the current data item.

Note that the data items can be out-of-date. Data items expire when they have been broadcasted and then the corresponding interest object moves from one OOkVD cell to another. A simple example is shown in **Fig. 4**, which is a part of **Fig. 1**. In **Fig. 4**, we assume that at time t the interest object o, which is inside R_{17}, is broadcasted, and at time $t+1$, o moves to R_1. Since when interest object o moves from R_{17} to R_1, data item $o(c_1, c_2, c_4)$ becomes $o(c_2, c_1, c_4)$, we say data item $o(c_1, c_2, c_4)$ is out-of-date at time $t+1$.

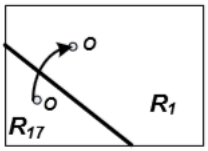


Fig. 4. An example of the out-of-date data item

4.3 Broadcast Scheduling Method

In this section, we propose a broadcast scheduling method based on the priorities of pengding requests and data items. Before presenting the priority computations, we first introduce some symbols that will be used.

$\overline{o}$: A data item containing interest object o.

UnservedSet(q): The set of un-broadcasted data items required by q. |*UnservedSet*(q)| indicates the number of data items in *UnservedSet*(q)..

$N(\overline{o})$:The total number of pending requests for $\overline{o}$. Once $\overline{o}$ is broadcasted, $N(\overline{o})$ becomes zero.

$V(o)$: The velocity of interest object o.

$L(o)$: The distance from o to the boundary of the OOkVD cell in which o resides along the moving direction of o. An example of $L(o)$ is shown in **Fig. 5**, where the arrow indicates the moving direction of o, and the length of the dotted line segment is $L(o)$.

Fig. 5. An example of $L(o)$

$S(q)$: The remaining lifetime of q, which can be obtained by subtracting current time instant from the deadline of q.

$P(\overline{o})$: The priority of data item $\overline{o}$.

$P(q)$: The priority of request q.

The priority computations should consider the following two issues: (1) the number of objects required by an ORkNN request can be arbitrary; (2) there are time limits, not only because each ORkNN query is submitted together with a deadline, but also because a broadcasted object of interest may become out-of-date due to its movement.

Naturally, given two requests, the one with closer deadline should be broadcasted first; given two requests, the one whose required data items are demanded by more other requests should be broadcasted first; given two data items, the one with the larger request frequency should be broadcasted first. Once an interest object moves out of the current OOkVD cell, the results of the corresponding ORkNN requests may change.

Motivated by these observations, we define $P(\overline{o})$ and $P(q)$ as follows.

$$P(\overline{o}) = \frac{L(o) \times N(\overline{o})}{V(o)} \tag{1}$$

$$P(q) = \frac{\sum_{o \in UnservedSet(q)} P(\overline{o})}{\left| UnservedSet(q) \times S(q) \right|} \tag{2}$$

Notice that in both equations, a larger value indicates a higher priority.

The existence of out-of-date data items could lead a client receiving incorrect query results. Therefore we should attempt to avoid generating out-of-date data items. If we broadcast a data item with a small value of $L(o)/V(o)$, the data item is more likely to expire. To some extent, the value of $L(o)/V(o)$ indicates the expected time o spends moving out of the current OOkVD cell. Generally, when o arrives to a new OOkVD cell, it has a larger value of $L(o)/V(o)$. Thus, we give data items with smaller $L(o)/V(o)$ low priority.

Below we show the basic idea of broadcast scheduling following a presentation of the priority calculations.

We say a request q is finished when all data items required by q have been broadcasted; q is terminated when $S(q) < N_q \times broadcast\_slot$. The request being served is called the current request.

At each decision point, the server selects a new data item to broadcast to clients, according to the following two rules:

(1) If the current request is finished or terminated, we choose a new request that has the highest priority, and the data item with the highest priority in the new current request is selected to broadcast;

(2) Otherwise, we choose the data item with the highest priority in the current request to broadcast.

Notice that we compute priorities for the pending requests before scheduling them for broadcasting to the clients, and then choose the most rewarding request to serve. To avoid the request starvation problem, data items within the same request are considered as a whole, meaning required data items of a request must be broadcasted consecutively before a new query is selected. We also assign a priority value to each data item in the requests. Consideration of data item popularity in scheduling can enhance the scheduling efficiency since broadcasting a hot data item can serve more requests at a time.

4.4 Summary of Server Algorithm

We use a request queue *RQ* to record the pending requests. For each pending request q, we use a broadcasted queue *BQ(q)* and an un-broadcasted queue *UQ(q)* to record the broadcasted and un-broadcasted data items required by q, respectively. The request being served is called the current request.

The entire algorithm on the server is briefly presented as follows.

(1) When a new request q arrives, we first calculate the interest objects required by q, based on Lemma 2. If there is no interest object required by q, q is discarded. Otherwise, we check if $S(q)$ is smaller than $N_q \times broadcast\_slot$. If the answer is positive, q is discarded; otherwise, we insert q into the request queue.

(2) When all needed data items for request q are broadcasted or $S(q)$ is smaller than $N_q \times broadcast\_slot$, we first remove q from *RQ*, and then compute the priority for each request in *RQ*. Finally, we chose the reqeust that has the highest priority.

(3) For each pending request q in the request queue, which is issued by client $c_r \in C$, we examine its result set periodically to see if it changes. If the answer is positive, we update the data items required by q. There exist three situations to be considered: (1) If a new interest object moves into the influence cells of c_r, we insert the corresponding data item into the un-broadcasted queue of q; (2) If an interest object $o \in O$ moves out of the influence cells of c_r, and $\overline{o}$ has not been broadcasted, we remove $\overline{o}$ from the un-broadcasted queue of q; (3) If an interest object $o \in O$ moves out of the influence cells of c_r, and $\overline{o}$ has been broadcasted, we insert the updated $\overline{o}$ containing the updated K-tuple into the un-broadcasted queue of q.

(4) At each decision point, we choose a new data item to broadcast. Let q be the current request, we compute the priority $P(\overline{o})$ for each data item $\overline{o}$ in $UQ(q)$, and choose the data item with the highest priority to broadcast.

5. Client Algorithm

As shown in **Fig. 6,** algorithms on clients work as follows: (1) Submit requests to the server; (2) Receive broadcasted data items; (3) Calculate query results based on received data items. We focus on the third step in this paper. Note that except for the time when a new request is submitted, there exists no interaction between the client and the server, which can save both wireless bandwidth and the power on the client.

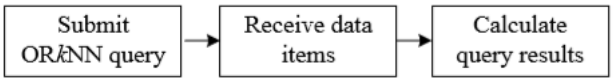


Fig. 6. Query processing on clients

Before presenting the client-side algorithm, we propose a lemma.

Lemma 3: Suppose the data items required by a client $c_r \in C$ are as follows:

$o_{11}(c_{i1}, \cdots, c_{iK})$, $\cdots$, $o_{1\alpha}(c_{i1}, \cdots, c_{iK})$ ($c_{i1} = c_r$);
$o_{21}(c_{i1}, \cdots, c_{iK})$, $\cdots$, $o_{2\beta}(c_{i1}, \cdots, c_{iK})$ ($c_{i2} = c_r$);
$\vdots$
$o_{K1}(c_{i1}, \cdots, c_{iK})$, $\cdots$, $o_{K\delta}(c_{i1}, \cdots, c_{iK})$ ($c_{iK} = c_r$).

Then we have ORkNN(c_r)={({o_{11}, $\cdots$, $o_{1\alpha}$ }, *1*), ({o_{21}, $\cdots$, $o_{2\beta}$ }, *2*), $\cdots$, ({o_{K1}, $\cdots$, $o_{K\delta}$ }, *K*)}.

Proof: This lemma can be proved according to Lemma 2 and the definition of the data item.

Below we present the algorithm executed on each client $c_r \in C$.

(1) c_r begins to listen to the downlink channel immediately after an ORkNN query is submitted.

(2) For each data item $o(c_{i1}, \cdots, c_{iK})$ that is received by c_r, we check if the identifier of client c_r is included in the *K*-tuple $(c_{i1}, \cdots, c_{iK})$, and if another data item $o(c_{j1}, \cdots, c_{jK})$, which contains the same interest object *o*, has already been received by c_r. Depending on the examination results, there exist four cases: (1) If c_r is included in the *K*-tuple $(c_{i1}, \cdots, c_{iK})$, and $o(c_{j1}, \cdots, c_{jK})$ exists, we insert the newly received $o(c_{i1}, \cdots, c_{iK})$ to the temporary result set and remove $o(c_{j1}, \cdots, c_{jK})$ from the temporary result set; (2) If c_r is not included in the *K*-tuple $(c_{i1}, \cdots, c_{iK})$ and $o(c_{j1}, \cdots, c_{jK})$ exists, we remove $o(c_{j1}, \cdots, c_{jK})$ from the temporary result set; (3) If c_r is included in the *K*-tuple $(c_{i1}, \cdots, c_{iK})$ and $o(c_{j1}, \cdots, c_{jK})$ does not exist, we insert $o(c_{i1}, \cdots, c_{iK})$ into the temporary result set; (4) If c_r is not included in the *K*-tuple $(c_{i1}, \cdots, c_{iK})$ and $o(c_{j1}, \cdots, c_{jK})$ does not exist, we discard $o(c_{i1}, \cdots, c_{iK})$ directly.

(3) c_r stops listening to the downlink channel when: i) The deadline expires; ii) All necessary data items have been received, i.e., the data item $o(c_{i1}, \cdots, c_{iK})$, which satisfies that c_r is included in $(c_{i1}, \cdots, c_{iK})$ and the first character of the identifier of c_r is represented by a number larger than 52, is received.

(4) Considering the data items in the temporary result set as the data items required by c_r, we calculate ORkNN(c_r) based on Lemma 2. Note that if no data item is received during the lifetime of the query request, we conclude that the query result set is empty.

6. Experimental Results and Analysis

6.1 Experimental Setup

The overall simulation model is based on the system architecture shown in **Fig. 2**, which consists of one base station, a number of clients and interest objects, and several wireless channels. The available bandwidth is set to be 84Mbps. The clients are generated as uniformly distributed points. For each interest object, we randomly chose two points as the starting point and ending point. The object of interest moves along the line segment between the two endpoints at a constant speed, which is also randomly generated between the maximum and minimum speeds. After arriving the ending point, the interest object takes the ending point as the new starting point and another point is chosen at random as the new ending point.

Meanwhile, we generate a random number between the maximum and minimum speed as the new speed. OR*k*NN query requests from different clients are independent. The lifetime of each request is randomly chosen between the maximal and minimal values. When a request deadline is reached, the client submits another request after a time interval, which is also a randomly selected value with the upper and lower limits. In our simulations, we obtain different workloads mainly by adjusting the time interval *T*.

The main parameters used in the simulations are summarized in **Table 3**. Note that, except when explicitly stated, the experiments are conducted under the default settings. For each group of settings, the program is run *100* times, and we take the average value as the output. Each simulation run lasts for *10,000* broadcast slots during which the number of clients stays unchanged and no new interest objects are added.

Table 3. Simulation parameters

Parameter	Default value
Client density	*0.1* per unit area
Interest object density	One per unit area
Minimal/Maximal interest object speed	*20*/*60* unit lengths per broadcast slot
Minimal / Maximal request lifetime	*10*/*30* broadcast slot
Minimal / Maximal time interval between two consecutive requests (abbreviated to *T*)	*5*/*15* broadcast slot
k	*3*

The algorithms are implemented by using *C*++ language and all the programs are tested on an Intel quad-core, *3.40*GHz, *8*GB-main-memory machine.

6.2 Experimental Results

6.2.1 Evaluation of Overhead

As mentioned in the broadcast scheduling method, at each scheduling decision point, the server chooses a new data item to be broadcasted by recalculating priorities. In this set of experiments, we examined the average overhead involved in making a scheduling decision, as well as the necessary time for calculating OR*k*NN results.

Fig. 7 shows the average time of making a scheduling decision when the number of data items to be broadcasted grows. Since we need to calculate data item priorities at each decision point, the time cost changes proportionally to the number of items needed to be broadcasted Some data items do not change and and do not need to be recalculated, so the time cost grows slowly.

We also investigated the time cost of calculating OR*k*NN results. As a result, the time used to calcute results of an OR*k*NN query is not strongly affected by the value of *k* (**Fig. 8**).

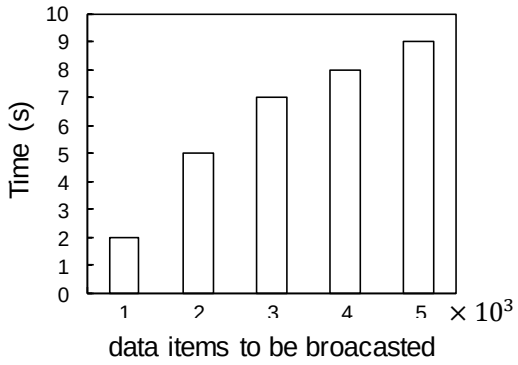


Fig. 7. Time cost of making a scheduling decision

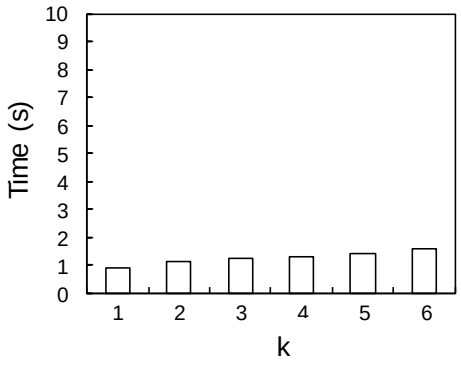
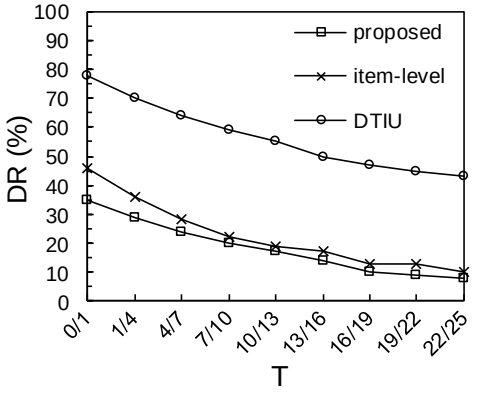


Fig. 8. Time cost of calculating OR*k*NN results

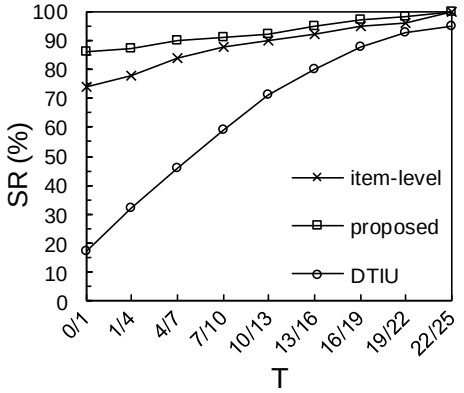
6.2.2 Evaluation of Broadcast Scheduling Method

To the best of our knowledge, we have not found any prior works that make the same assumptions as our present study. Therefore, we examine and compare our broadcast scheduling method (referred to as *proposed*) with algorithms *DTIU* [25], and *item-level*. In each of these experiments, we use the the proposed methods to calculate OR*k*NN results and construct data items. Then, one of *proposed*, *DTIU* and *item-level* is adopted to schedul the data items to be broadcasted. The assumptions of *DTIU* are the most similar to *proposed*, but it also assumes that the data items required by a request do not change during the lifetime of the request. *item-level* is the same as *proposed*, except that a query request is not scheduled as a whole, i.e., *item-level* schedules at the level of the item.

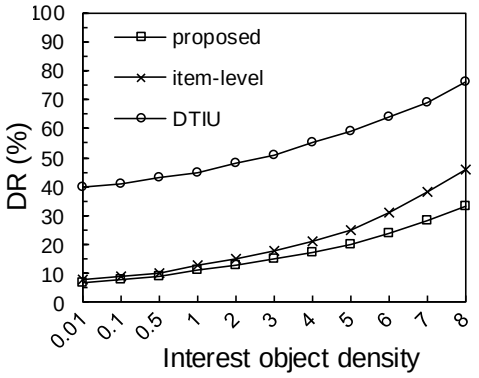
Two metrics, namely the request drop ratio (*DR*) and the scheduling efficiency ratio (*SR*), are adopted to evaluate and analyze the performance of our proposed algorithms. Let $|Q|$ be the total number of submitted requests and $|Q_{miss}|$ be the number of unsatisfied requests, then *DR* is defined as $|Q_{miss}|/|Q|$. *DR* measures the ability to satisfy requests. The ideal value of *DR* is zero, and means all requests are satisfied. In other words, a lower value of *DR* implies better performance. Let $|I|$ be the total number of broadcasted data items, and $|I_{useful}|$ represent the number of broadcasted data items that lead to at least one satisfied request, then *SR* is defined as $|I_{useful}|/|I|$. *SR* reflects the algorithm efficiency in satisfying requests. In particular, a higher scheduling efficiency ratio implies higher performance, meaning more requests are satisfied by the received data items. The maximum value of the scheduling efficiency ratio is one, and means all the broadcasted data items contribute to satisfied requests.


DR (%)
T
proposed
item-level
DTIU
0/1
1/4
4/7
7/10
10/13
13/16
16/19
19/22
22/25

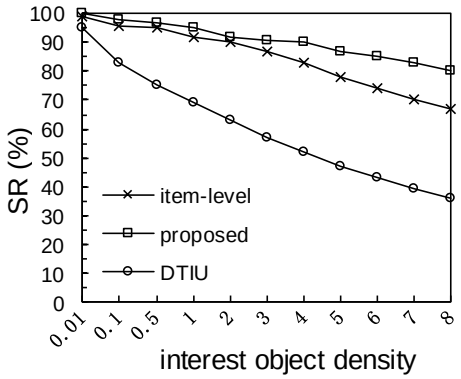
(a)


SR (%)
T
item-level
proposed
DTIU
0/1
1/4
4/7
7/10
10/13
13/16
16/19
19/22
22/25

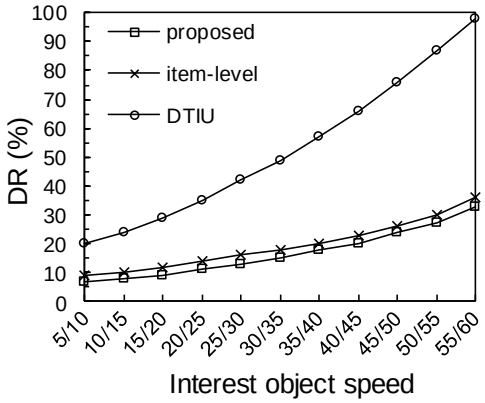
(b)


DR (%)
Interest object density
proposed
item-level
DTIU
0.01
0.1
0.5
1
2
3
4
5
6
7
8

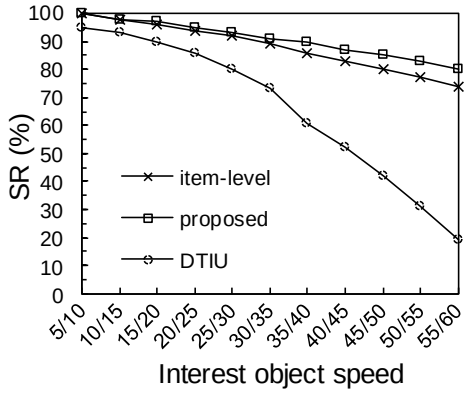
(c)


SR (%)
interest object density
item-level
proposed
DTIU
0.01
0.1
0.5
1
2
3
4
5
6
7
8

(d)


DR (%)
Interest object speed
proposed
item-level
DTIU
5/10
10/15
15/20
20/25
25/30
30/35
35/40
40/45
45/50
50/55
55/60

(e)


SR (%)
Interest object speed
item-level
proposed
DTIU
5/10
10/15
15/20
20/25
25/30
30/35
35/40
40/45
45/50
50/55
55/60

(f)

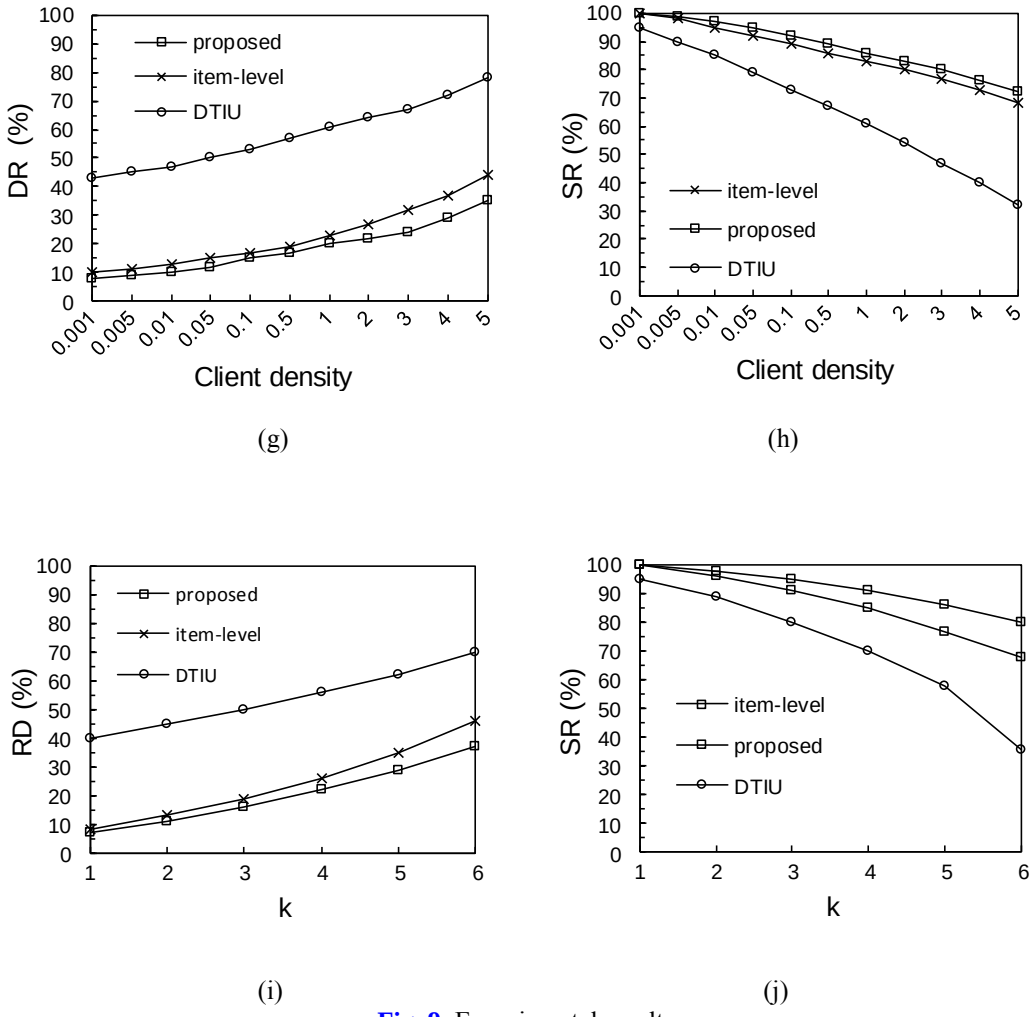


Fig. 9. Experimental results

Effect of *T*. In this set of experiments, the algorithms are evaluated under different *T* values. A lower value of *T* results in a larger request arrival rate and a heavier workload. It is reasonable to see that, in **Fig. 9** (a) and (b), all algorithms have increased performance as *T* grows. As can be seen, *proposed* outperforms the other algorithms. *DTIU* performs the worst because it does not take the changes of the data items into consideration. *item-level* deteriorates faster when query workload increases than *proposed*. This is due to its scheduling at the data item level leading to unsatisfied requests, even if only one data item is left broadcasted. Scheduling a request as a whole helps to alleviate this request starvation problem. The results are consistent with the results reported in previous study on multi-item request scheduling [24].

Effect of Interest Object Density. **Fig. 9** (c) and (d) show the effect of the interest object density on *proposed*, *DTIU* and *item-level*. As can be seen, the algorithm of *proposed* performs best. In general, when the value of the interest objects density becomes larger, it is more likely that an OR*k*NN query request asks for more data items and that the result set of a request changes during the lifetime. Thus, all algorithms deteriorate as the interest object density grows. In general, a larger density of interest objects implies a larger set of request

results and a larger probability of appearances by invalid data items. In such cases, *item-level* comes across more serious problems and *DTIU* becomes more prone to error.

Effect of Interest Object Velocity. In **Fig. 9** (e) and (f), we vary the speed of interest objects and study the effect on *proposed*, *DTIU* and *item-level*. In general, the larger the interest object's speed, the more likely a broadcasted data item is to be out-of-date. *DTIU* performance deteriorates sharply as the interest objects move faster, because the movements of the interest objects are not involved at all when *DTIU* makes broadcast scheduling decisions.

Effect of Client Density. **Fig. 9** (g) and (h) show the request drop ratio and scheduling efficiency ratio of the algorithms *proposed, DTIU* and *item-level*, when the client density increases. All else equal, the larger the client density, then the more requests submitted to the server within the same period of time, the less interest objects are requested by an OR*k*NN query, and the more likely the interest objects are to move out of their original OO*k*VD cells. Consequently, less empty-result requests appear. Thus, all algorithm performances deteriorate with the client density. *DTIU* deteriorates faster because it is not designed to schedule moving data items. As the average result size becomes small and the starvation problem is not that serious, *item-level* performs relatively better.

Effect of *k*. In **Fig. 9** (i) and (j), we varied the value of *k* and studied the effect of *k*. For any *k* value, the number of objects required by an OR*k*NN query does not exceed *k*. In particular, when *k*=1, the number of objects requested by an OR*k*NN query is either zero or one. In general, a larger *k* value means a request needs more data items, which also leads to a larger probability of the appearance of invalid data items. In such cases, *proposed* performs best, as shown in **Fig. 9** (i) and (j).

In brief, the proposed algorithms are demonstrated to be effective and efficient. The overhead of calculating OR*k*NN results, and the time cost of making a scheduling decision, suggest that the OR*k*NN result calculation and data item organization methods are efficient. Furthermore, the comparison result of the proposed broadcast scheduling method and other related methods suggest that the proposed broadcast scheduling method is efficient for handling OR*k*NN queries via on-demand broadcast. Above all, the final results (*DR* and *SR*) suggest that the entire processing method may have practical applications.

7. Conclusion

This paper processes ordered order *k* reverse nearest neighbor (OR*k*NN) queries in on-demand environments. Experimental results regarding the calculation of OR*k*NN results and the broadcast scheduling approach are presented. The experimental results suggest that our proposed algorithms may have practical applications.

8. Acknowledgment

The authors would like to thank Professor LihChyun Shu and Jingpeng Wu for polishing the writing. This work was substantially supported by National Natural Science Foundation of China under Grants No.61173049, No. 61332001 and No. 61309002.

References

[1] K. C. K. Lee, Baihua Zheng and Wang-Chien Lee, "Ranked Reverse Nearest Neighbor Search," *IEEE Trans. Knowl. Data Eng.,* vol. 20, no. 7, pp. 894. Article (CrossRef Link)

[2] Flip Korn and S. Muthukrishnan, "Influence Sets Based on Reverse Nearest Neighbor Queries," in *Proc. of the 2000 ACM SIGMOD International Conference on Management of Data*, pp. 201–212, 2000. Article (CrossRef Link)
[3] Congjun Yang and King-Ip Lin, "An index structure for efficient reverse nearest neighbor queries," in *Proc. of the 17th International Conference on Data Engineering*, pp. 485-92, 2001. Article (CrossRef Link)
[4] King-Ip Lin, M. Nolen, and Congjun Yang, "Applying bulk insertion techniques for dynamic reverse nearest neighbor problems," *IDEAS*, 2003. Article (CrossRef Link)
[5] I. Stanoi, D. Agrawal and A. El Abbadi, "Reverse Nearest Neighbor Queries for Dynamic Databases," in *Proc. of the ACM SIGMOD Workshop on Data Mining and Knowledge Discovery (DMKD)*, 2000. Article (CrossRef Link)
[6] J. M. Kang, M. F. Mokbel, S. Shekhar, Tian Xia and Donghui Zhang, "Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors," *ICDE*, 2007. Article (CrossRef Link)
[7] Guohui Li, Yanhong Li, Jianjun Li, LihChyun Shu, Fumin Yang, "Continuous reverse *k* nearest neighbor monitoring on moving objects in road networks," *Information Systems,* 2010. Article (CrossRef Link)
[8] Wei Wu, Fei Yang, Chee Yong Chan and K. -L. Tan, "Continuous Reverse k-Nearest-Neighbor Monitoring," *Mobile Data Management,* 2008. Article (CrossRef Link)
[9] Tian Xia, Donghui Zhang, "Continuous reverse nearest neighbor monitoring," *ICDE*, 2006. Article (CrossRef Link)
[10] Rimantas Benetis, Christian S. Jensen, Gytis Karčiauskas and Simonas Šaltenis, "Nearest and reverse nearest neighbor queries for moving objects," *VLDB Journal*, vol. 15, no. 3, pp. 229-250, 2006. Article (CrossRef Link)
[11] H. -P. Kriegel, P. Kroger, M. Renz, A. Zufle and A. Katzdobler, "Incremental reverse nearest neighbor ranking," in *ICDE*, 2009. Article (CrossRef Link)
[12] Doohee Song and Kwangjin Park, "An Efficient Adaptive Bitmap-based Selective Tuning Scheme for Spatial Queries in Broadcast Environments," *KSII Transactions on Internet & Information Systems,* 2011. Article (CrossRef Link)
[13] P. Pahlavani, V. Derhami and A. M. Z. Bidoki, "FENC: Fast and Efficient Opportunistic Network Coding in wireless networks," *KSII Transactions on Internet & Information Systems*, 2011. Article (CrossRef Link)
[14] S. A. Alomari and P. Sumari, "Effective Broadcasting and Caching Technique for Video on Demand over Wireless Network," *KSII Transactions on Internet & Information Systems,* 2012. Article (CrossRef Link)
[15] J. Chen, Y. Xu, L. Ma and Y. Wang, "Multi-agent Q-learning based Admission Control Mechanism in Heterogeneous Wireless Networks for Multiple Services," *KSII Transactions on Internet and Information Systems (TIIS),* 2013.
[16] C. X. Liu, Y. Zhang, E. Xu, Y. Q. Yang and X. H. Zhao, "A Novel Multi-Path Routing Algorithm Based on Clustering for Wireless Mesh Networks," *KSII Transactions on Internet & Information Systems,* 2014.
[17] A. Okabe, B. Boots, K. Sugihara and S. N. Chiu, "Spatial tessellations: concepts and applications of Voronoi diagrams," *John Wiley & Sons*, vol. 501, 2009.
[18] R. Benetis, Christian S. Jensen, G. Karciauskas and S. Saltenis, "Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects," in *Proc. of IDEAS*, pp. 44-53, 2002. Article (CrossRef Link)
[19] M. A. Cheema, Xuemin Lin, Wenjie Zhang and Ying Zhang, "Influence zone: Efficiently processing reverse k nearest neighbors queries," in *Proc. of ICDE*, pp. 577-588, 2011. Article (CrossRef Link)
[20] InHo Jang and SangKeun Lee, "Search Reverse Nearest Neighbor Query On Air," in *Proc. of International Conference on Information Technology*, pp. 291-296. Ariticle (CrossRef Link)
[21] Lien-Fa Lin, "Search RNN on Broadcast Environment," *Proc. of Third International Conference on Intelligent Information Hiding and Multimedia Signal*," vol. 2, pp. 361-364, 2007. Article (CrossRef Link)

[22] Jianliang Xu, Xueyan Tang and Wang-Chien Lee, "Time-critical on-demand data broadcast: Algorithms, analysis, and performance evaluation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 1, 2006. Article (CrossRef Link)

[23] D. Aksoy and M. Franklin, "R×W: a scheduling approach for large-scale on-demand data broadcast," *IEEE/ACM Transactions on Networking (ToN)*, vol. 7, no. 6, pp. 846-860, 1999. Article (CrossRef Link)

[24] Jun Chen, Victor C. S. Lee and Kai Liu, "On the performance of real-time multi-item request scheduling in data broadcast environments," *Journal of Systems and Software,* vol. 83, no.8, pp. 1337-45, 2010. Article (CrossRef Link)

[25] Jun Chen, Victor C. S. Lee and Edward Chan, "Scheduling real-time multi-item requests in wireless on-demand broadcast networks," in *Proc. of the 4th ACM international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology,* pp. 125-131, 2007. Article (CrossRef Link)

[26] H. D. Dykeman and J. W. Wong, "A performance study of broadcast information delivery systems," in *Proc. of IEEE INFOCOM*, 1988. Article (CrossRef Link)

[27] Ping Xuan, S. Sen, O. Gonzalez, J. Fernandez and K. Ramamritham, "Broadcast on demand: Efficient and timely dissemination of data in mobile environments," in *Proc. of 3rd IEEE Real-Time Technology Application Symposium*, 1997. Article (CrossRef Link)

[28] Chuan-Ming Liu and Kun-Feng Lin, "Efficient scheduling algorithms for disseminating dependent data in wireless mobile environments," in *Proc. of IEEE Conf. on Wireless Networks, Communications and Mobile Computing*, pp. 375-380, 2005. Article (CrossRef Link)

[29] Kai Liu and Victor C. S. Lee, "Performance analysis of data scheduling algorithms for multi−item requests in multi−channel broadcast environments," *International journal of communication systems*, vol. 23, nol. 4, pp. 529-542. Article (CrossRef Link)

[30] Hongya Wang, Yingyuan Xiao and LihChyun Shu, "Scheduling Periodic Continuous Queries in Real-Time Data Broadcast Environments," *IEEE Transactions on Computers*, vol. 61, no.9, pp. 1325-40, 2012. Article (CrossRef Link)

Li Li is a Phd candidate in School of Computer Science and Technology, Huazhong University of Science and Technology. She received her B.E. degree in Information Security from Chongqing University of Posts and Telecommunications, China, in 2009. Her reaearch interests mainly include spatial data management and data mining methodologies.

Guohui Li received his B.E. degree from Nanjing University of Aeronautics and Astronautics in 1994 and Ph.D. degree from Huazhong University of Science and Technology in 1999, where he is now a full professor. His research interests include location-based data management, big data processing and real-time computing.

Quan Zhou received the B.E. degree in software engineering from Heilongjiang University, Harbin, Heilongjiang, P.R. China, in 2009. He is currently a PhD candidate in Huazhong University of Science and Technology, Wuhan, Hubei, P.R. China. His research interests include real-time computing, mobile computing and database systems.

Yanhong Li received the B.E. degree in Mechanical Engineering from Central South University (CSU), China, in 1993, the M.S. degree in Computer Application from Chongqing University (CQU), China, 2004, and the Ph.D degree in Computer Softwareand Theory from Huazhong University of Science and Technology (HUST) in 2011. Since 2012, she has been on the faculty in computer school at South-Central University for Nationalities, and she is currently a associate professor in the Department of Computer Application. Her research interests include Spatial Information and Communication, and multimedia network communication technology.