Journal of Computer and System Sciences 77 (2011) 637–651

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Voronoi-based range and continuous range query processing in mobile databases

Kefeng Xuan [a], Geng Zhao [a], David Taniar [a,*], Wenny Rahayu [b], Maytham Safar [c], Bala Srinivasan [a]

[a] *Clayton School of Information Technology, Monash University, Australia*
[b] *Department of Computer Science and Computer Engineering, La Trobe University, Australia*
[c] *Computer Engineering Department, Kuwait University, Kuwait*

ARTICLE INFO

Article history:
Received 15 September 2009
Received in revised form 11 January 2010
Available online 1 March 2010

Keywords:
Mobile databases
Range search
Mobile query processing
Voronoi diagram

ABSTRACT

With the wide availability of mobile devices (smart phones, iPhones, etc.), mobile location-based queries are increasingly in demand. One of the most frequent queries is range search which returns objects of interest within a pre-defined area. Most of the existing methods are based on the road network expansion method – expanding all nodes (intersections and objects) and computing the distance of each node to the query point. Since road networks are extremely complex, node expansion approaches are inefficient. In this paper, we propose a method, *Voronoi Range Search* (*VRS*) based on the Voronoi diagram, to process range search queries efficiently and accurately by partitioning the road networks to some special polygons. Then we further propose *Voronoi Continuous Range* (*VCR*) to satisfy the requirement for continuous range search queries (moving queries) based on *VRS*. Our empirical experiments show that *VRS* and *VCR* surpass all their rivals for both static and moving queries.

1. Introduction

Mobile databases [31] have benefited through the rapid advancement of global positioning systems (GPS) [3] and geographic information system (GIS) [35,18], as well as the mobile device itself [12]. As a result, there has been a growing demand for mobile location-based service applications [13,9,4] and ubiquitous applications [34]. One of the most common queries in mobile, as well as in spatial, databases is range search [33] that finds all objects of interest within the given region or radius. It can be defined formally as: given a query point q, a user specified range e and a set of objects of interest $\wp$, find all objects of interest from $\wp$ within range e from q.

Range search on road networks has been applied to mobile navigation. The two most well-known methods are Range Euclidean Restriction (RER) and Range Network Expansion (RNE) [15], which calculate the shortest network distance between two objects located on the road networks. All objects of interest returned to users are evaluated according to their network distances to the query point, which can be calculated easily using the Dijkstra algorithm [2]. Therefore RER and RNE [15] provide more accurate results than the Euclidean range search approaches [8]. The disadvantages of these approaches [15] are also obvious, since RER and RNE are based on expansion methods and consequently, many false hits are retrieved during the expansion, which makes these methods time consuming. Moreover, if the density of the objects is very low, the performance of expansion-based approaches will decrease dramatically.

* Corresponding author.
E-mail addresses: Kefeng.Xuan@infotech.monash.edu.au (K. Xuan), Geng.Zhao@infotech.monash.edu.au (G. Zhao), David.Taniar@infotech.monash.edu.au (D. Taniar), W.Rahayu@latrobe.edu.au (W. Rahayu), Maytham.Safar@ku.edu.kw (M. Safar), Bala.Srinivasan@infotech.monash.edu.au (B. Srinivasan).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2010.02.005

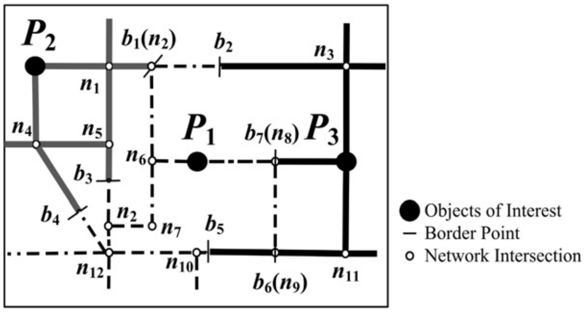


Fig. 1. Network Voronoi Diagram.

In mobile navigation, users who invoke the range search may also move, and therefore, range search query becomes continuous; therefore, the server needs to let the users know when the range query results change. Although continuous *k*NN query processing methods have long been addressed by many researchers [28,11,19,35], continuous range search is still an unexplored area. In our previous work, we proposed an algorithm named Continuous Range Search (CRS) [32] based on both Euclidean distance and network distance. But *CRS*, like other continuous *k*NN [28,11,19], needs to divide the traveling path into sub-segments according to the intersections of the road networks. This is certainly not suitable for the complex networks, where the performance will degrade dramatically.

In this paper, we propose two approaches, namely, *Voronoi-based Range Search* (*VRS*) and *Voronoi Continuous Range* (*VCR*) which are based on some natural properties of Network Voronoi Diagram (NVD) [14], dedicated to range search query processing for both static and moving queries. *VRS* can reduce the expansion area to promote the efficiency by using some pre-calculated components. Furthermore, it can reduce the rate of false hits and keep it at a lower level. We then extend *VRS* to include continuous query, subsequently called VCR. Because *VCR* no longer needs to divide the moving path into some segments, the performance and the applicability of the continuous range search have been improved considerably. Since the details of connections in networks can be ignored during the moving of the query point, *VCR* can also solve the undefined-trajectory continuous query properly.

2. Network Voronoi Diagram: a background

Our proposed algorithms, both *VRS* and *VCR* use the Voronoi diagram as their underlying framework. A Voronoi diagram has numerous geometric properties that can immensely improvement the performance of range search queries processing. A Voronoi diagram is a decomposition of a plane space according to the position of a set of discrete points (site). Each site generates a Voronoi Polygon (VP), which involves all points closer to its site than to any other. A VP is formed by a set of Voronoi edges that are subsets of a locus of points equidistant from two adjacent sites. The intersection of these edges for a site is called the Voronoi vertex. The definition of a Voronoi diagram is:

Definition 1. Given a set of discrete objects $\wp = (P_1, \ldots, P_n)$ $(n > 1)$ in road networks, $VP(P_i) = \{\forall p \mid d_n(p, P_i) \leqslant d_n(p, P_j)\}$ $(i, j \in I_n$ and $i \neq j)$. The Voronoi Diagram for $\wp$ is:

$$VD(\wp) = \bigcup_{i=1}^{n} VP(P_i)$$

This definition indicates that for any point inside the $VP(P_i)$, its distance to P_i must be smaller than to others generators, then $VP(P_i)$ is called a Voronoi polygon associated with P_i.

An NVD is a special kind of Voronoi diagram constructed on spatial networks [14]. The decomposition is based on the connection of the discrete objects rather than Euclidean distance. In the NVD, the Voronoi polygon changes to a set of road segments termed Network Voronoi polygon (NVP) and the edges of the polygons also shrink to some midpoints, termed border points, of the road network connection between two objects of interest.

Fig. 1 shows an example of NVD. Besides the objects of interest (P), an NVD also includes some road network intersections (n) and border points (b). According to the properties of a Voronoi diagram, from border points to a pair of adjacent objects is equidistant (e.g. $dis(b_7, P_1) = dis(b_7, P_3)$), then we just need to use Dijkstra's algorithm within one Voronoi polygon to obtain the distance from a generator to its borders. The distance between objects can be calculated by selecting the minimum distance to their shared borders and doubling this value (e.g. $MIN(dis(P_3, P_1)) = 2 * dis(b_7, P_3) = 2 * dis(b_7, P_1)$).

As all distances from border points to generators/other border points can be precalculated using Dijkstra's algorithm and stored in databases, then when a range query is issued, all required distances can be retrieved from databases rather than being calculated online. So with the assistance of NVD, *VRS* and *VCR* can process range queries more efficiently than the traditional range search methods.

3. Related work

This section will briefly introduce some existing approaches that focus on network range search, Voronoi-based methods on *k*NN and C*k*NN, and a continuous range search approach. We conclude with a summary of their limitations.

3.1. Network range search methods

Range search on network distance is very different from the previous methods. The most efficient algorithms for range search on network distance are known as RER and RNE [15].

RER applies a range search on Euclidean distance e from query point q to retrieve all possible candidates, as the Euclidean distance can be seen as the lower boundary for network distances $d_e(q,p) \leqslant d_{net}(q,p)$, which ensures that all objects of interest within the searching range will not be missed. But a large number of false hits, which have $d_e(q,p) < e$, and $d_{net}(q,p) > e$, are involved in this procedure. These false hits have to be filtered from the candidates list by performing network expansion in the next step until all the candidates tested or all the segments in the range have been exhausted.

According to the algorithm of RER, if only a few objects are located in complex networks, the processing time will increase sharply, because most of the operation will be wasted on the network expansion to retrieve only a small number of objects. In another scenario where the network distances are quite different from the corresponding Euclidean distance, the candidates sets will include numerous false hits which need to be crossed out one by one in the filter step.

RNE outperforms RER by using an expansion method whereby a set of segments within the network range e filters the false hits first. Then RNE uses R-tree indexing to find the objects of interest falling on the valid segments, which intersect with minimum boundary rectangles (MBRs) [8]. Finally, when R-tree indexing is finished, the results are sorted to remove the duplicates.

However, RNE still cannot solve the problem of some unnecessary expansions being performed. If the range area is enormous, RNE will need to scan every segment even if there are just a few objects needing to be retrieved in the searching range. So we try to seek a new approach which can avoid false hits or keep it at a reasonable level, whilst at the same time, minimizing the expansion.

3.2. Continuous range search method

Our previous work on continuous range search, which we call *CRS* [32], processes continuous range query by dividing the predefined trajectory into sub-segments according to the intersections of the networks. For each sub-segment, CRS uses the network expansion range search technique, such as RER/RNE [15] at two ends to retrieve all objects of interest within the predefined range. Then with the movement of the query point, objects that become accessible or inaccessible will be added or deleted respectively from the candidate list at the split point.

The disadvantages of CRS are also obvious. Firstly, the trajectory needs to be segmented according to the intersections. If the number of intersections is massive, the number of the sub-segments will be extremely large, which will occupy not only huge storage space, but also unnecessary repeats processes. Secondly, since CRS uses network expansion, all nodes on road networks need to be accessed and compared with the searching range. Thirdly, CRS can process only pre-defined trajectory continuous queries. If the moving path is changed on the way to the destination, CRS cannot be applied.

3.3. Voronoi-based methods for kNN and CkNN

Numerous algorithms for *k*NN [10,18] and continuous *k*NN [28,11,19] based on the Voronoi diagram have been proposed in recent years. All of them utilize the geometry properties of the Voronoi diagram to provide a better performance than other traditional network expansion methods.

The Voronoi diagram was first adopted in *k*NN by Kolahdouzan and Shahabi [10] in whose performance results showed that their approach VN^3 outperformed its main competitor INE [15] by up to one order of magnitude. One year later, they proposed two new approaches for C*k*NN query, named IE and UBA [11]. They used these approaches to find the different variation trends of the network distance to the query point between adjacent objects in the candidates set when the query point is moving. The only difference between IE and UBA is that the former compares *k*NN results of the two ends of the road segment, while UBA just extends the *k*NN results at one end to enhance the execution performance.

At the same time, another group proposed an improved algorithm of VN^3, called PINE [18]. After that, DAR and *e*DAR [19] were proposed for continuous *k*NN queries. Voronoi diagram has also been utilized in other newer query types, such as multiple type objects *k*NN [36] and Reverse Nearest Neighbour (RNN) [20].

The limitations of the above approaches are summarized in Table 1.

4. Voronoi-based Range Search (VRS)

In this section, we present our static range search based on NVD, which we name *Voronoi-based Range Search* (*VRS*).

Table 1
Existing methods.

Query classification	Methods	Limitations
Continuous Range Search	CRS	Inefficient performance
Network range search	RER/RNE VN^3/PINE	High number of false hits and large expansion area
Voronoi based *k*NN/C*k*NN	IE/UBA DAR/*e*DAR	Work on *k*NN/C*k*NN, not on range search

Data structure of object of interest P_i	***Example of P_1 in Figure 9***:
Object P_i	P_1
(	(
Adjacent Components,	$P_2, P_3, P_4, P_5, P_7, P_8$
Border points,	$b_1, b_2, b_3, b_4, b_5, b_6, b_7$
NVP(P_i),	NVP(P_1)
$dis_{net}(q, P_i)$,	$dis_{net}(q, P_1)$ = NULL
Inside/outside	True
)	)

Fig. 2. Data structure of object of interest.

4.1. Data structure and storage schema

To demonstrate our approach, the data structure of the key objects, the NVD generators and Q_{pre} used to store candidates should be clearly defined as well as the network distance calculation techniques (see Fig. 2).

- *Adjacent Components*: Store the generators of adjacent NVPs for P_i.
- *Border points*: Store all border points that belong to NVP(P_i), each border point b_x that stores the pre-calculated components relevant to itself, $dis_{net}(b_x, P_j)$ and $dis_{net}(b_x, b_y)$, $j \neq i$, $x \neq y$, $b_x, b_y \in BoP(\text{NVP}(P_i))$.
- NVP(P_i): A network Voronoi polygon associates with P_i. If $\text{NVP}(P_i) = \text{NVP}(P_q)$, then NVP($P_i$) contains the query point q.
- $dis_{net}(q, P_i)$: It involves two kinds of network distance calculation mechanisms: (a) Incremental Network Expansion (INE) [15], (b) Off-line Precalculation [18]. Within the first polygon NVP(P_q) including the query point, INE will be used to get the distances ($dis_{net}(q, b_x)$) from query point to the border points of NVP(P_q). After that, pre-calculated distances (e.g. border to generators and border to border) will be invoked to incorporate with $dis_{net}(q, b_x)$ to obtain the entire distance $dis_{net}(q, P_i)$, see Eq. (1).

$$dis_{net}(q, P_i) = dis_{net}(q, b_x) + dis_{net}(b_x, P_i) \tag{1}$$

- *Inside/outside*: A Boolean variable indicates the relative position of P_i within the given searching range. When P_i is added into Q_{pre}, the default value is true.
- Q_{pre}: A sorted queue storing a set of objects whose validity needs to be checked. Q_{pre} does not involve only all valid objects within the searching range, but also includes a few false hits just out of the searching range.

4.2. Terminologies

We also need to formally define a number of basic terms to be used in *VRS*, namely *Expected Searching Range*, *Current Searching Range* and *Range Borderline*.

Definition 2. Given a static query point q in the networks, the Expected Searching Range is a set of road segments, on which any point p to q is smaller than/equal to e.

$$p \in \textit{Expected Searching Range} \quad \Leftrightarrow \quad dis_{net}(p, q) \leqslant e$$

The thick lines in Fig. 3 represent an example of Expected Searching Range.

Definition 3. Range Borderline is the boundary of the expected searching range and it is represented as an irregular polygon by connecting the adjacent ends of expected searching range.

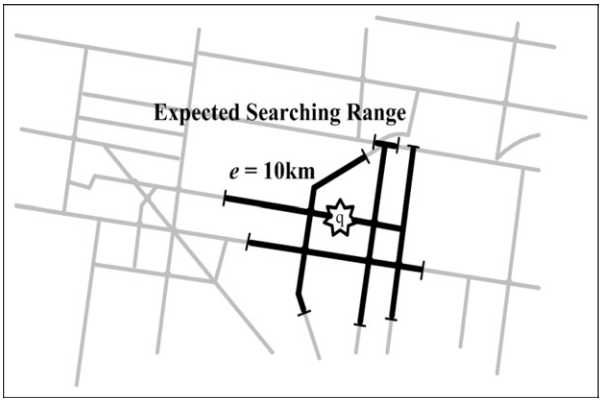


Fig. 3. Expected Searching Range.

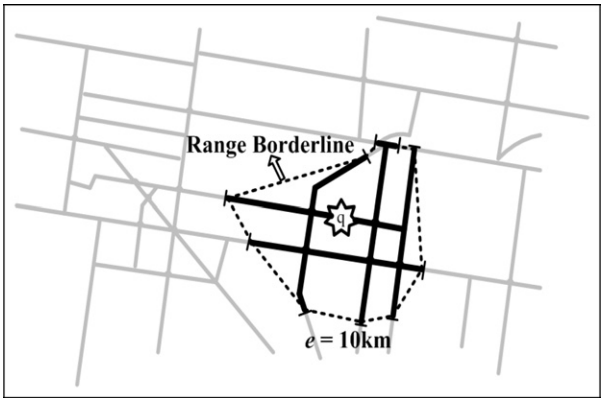


Fig. 4. Range Borderline.

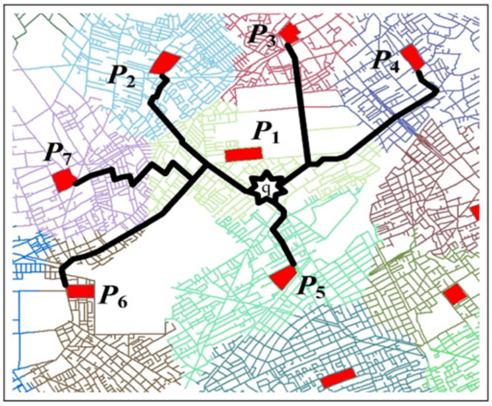


Fig. 5. Current Searching Range.

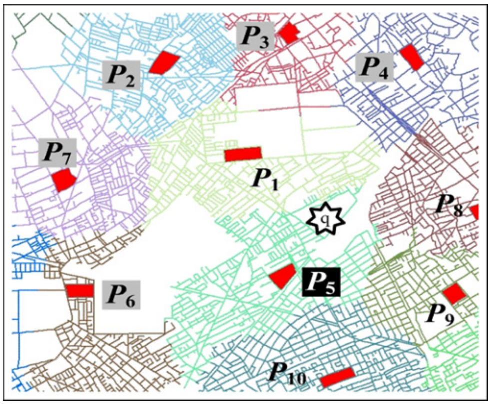


Fig. 6. Inside/outside NVP.

Range Borderline gives us a visualized boundary of the searching range, which provides a mappable way to observe the relative position of the objects of interest within the searching range. Fig. 4 shows a Range Borderline of the expected searching range shown in Fig. 3.

Definition 4. Current Searching Range is composed by a set of road segments ending at some generators of NVPs, whose network distances to q need to be compared with e. Current Searching Range is represented by the largest network distance from query point q to the furthest retrieved objects.

$$\mathit{Current\ Searching\ Range} = dis_{max} = MAX\big(dis_{net}(q, P_i)\big)$$

In Fig. 5, current searching range ends at P_2, P_3, P_4, P_5, P_6, P_7 and it is represented by $dis_{max} = MAX(dis_{net}(q, P_i)) = dis_{net}(q, P_6)$.

Definition 5. Inside/outside NVP (inNVP/outNVP) is a set of adjacent NVPs of current NVP(P_i), whose network distance from query point to its generator is smaller/larger than the distance from query point to P_i, denote as $\text{inNVP}_{P_i}()/\text{outNVP}_{P_i}()$. Its generator is called the inside/outside adjacent generator of P_i, denote as $\text{inAG}_{P_i}()/\text{outAG}_{P_i}()$.

$$\exists \text{NVP}(P_j) \in \text{inNVP}_{P_i} \quad \Rightarrow \quad dis_{net}(P_j, q) < dis_{net}(P_i, q)$$
$$\exists \text{NVP}(P_j) \in \text{outNVP}_{P_i} \quad \Rightarrow \quad dis_{net}(P_j, q) > dis_{net}(P_i, q)$$

The inside and outside is relevant to the relative position of the query point and the generator of NVP (see Fig. 6). For P_1, its adjacent NVPs are NVP(P_2), NVP(P_3), NVP(P_4), NVP(P_5), NVP(P_6), NVP(P_7) in which NVP(P_5) is P_1's inside NVP, while rest are its outside NVPs.

4.3. Main process

The main process of VRS can be summarized as performing network-segment expansion in the first polygon, and then bulk loading a set of NVPs recursively, until all objects of interest within the expected searching range are retrieved. During this procedure, there are three main steps: (i) *Locating the Query Point*, (ii) *Expanding Current Searching Range*, and (iii) *Locating Range Borderline*.

Step 1 *Locating Query Point*: invoke function contain(q) which can invoke a common spatial index structure (e.g. R-Tree) to find the NVP, called NVP(q), where the query point located, and add its generator P_q into Q_{pre}. In the process of locating the query point, no distance needs to be calculated to obtain the location of the query point in NVD. The efficiency of locating the query point will be manifested when the expected searching range is small (such as, expected searching range is inside of the NVP(q)), in this case, P_q is the only object within the searching range.

Step 2 *Current Searching Range Expansion*: use network Voronoi polygon expansion instead of the traditional network expansion used by other range search approaches. The network expansion method, e.g. INE, needs to go through every node (intersection, object) in the road network. In the worst environment, if the intersections of the networks are enormous, the efficiency of the network expansion will decrease dramatically. The NVP expansion bulk-loads a set of NVPs, with a consequence that it avoids going through the details in the polygons, until all objects within the range are retrieved. Meanwhile, all objects that need to be tested will be inserted into Q_{pre} and the range borderline will be located. The advantage of NVP expansion is that it is not restricted by the pattern of the networks. In each step, the expansion area is much larger than the network expansion methods.

We introduce the following properties to illustrate when the current searching range expansion needs to be implemented and to constrain the expansion area to only outNVPs of current retrieved NVPs.

Property 1. $Q_{pre} = (P_1, P_2, \ldots, P_n) \in \wp$, $i \in n$, $P_i \in \wp$, $\mathrm{outAG}_{P_i}() = (P_m, P_{m+1}, \ldots, P_{m'}) \in \wp$, $m' < 6n$, if $dis_{max} = dis_{net}(q, P_i)$. According to the definition of the outside adjacent generator, then $\forall P_j \in \mathrm{outAG}_{P_i}()$, $dis_{net}(q, P_j) > dis_{max}$.

Property 2. $Q_{pre} = (P_1, P_2, \ldots, P_n) \in \wp$, $i \in n$, $P_i \in \wp$, $dis_{max} = dis_{net}(q, P_i) > \text{Constant}$. According to the definition of the outside adjacent generator, then $P_j \in \mathrm{outAG}_{P_i}() \Rightarrow dis_{net}(q, P_j) > \text{Constant}$.

The initial current searching range is the first polygon determined in the locating query point step, whereby $dis_{max} = dis_{net}(q, P_q)$. According to Property 1, the outside adjacent generators of P_q must be further to q. If P_q is already outside of range $dis_{max} > e$, follow Property 2: $dis_{max} > e \Rightarrow dis_{net}(q, P_j) > e$, then there is no need to do the expansion. Otherwise, all outside adjacent generators of P_q will be inserted into Q_{pre} to expand the current searching range. After that, the third step will be called to verify the objects in Q_{pre} and check whether to keep expanding or to carry out Gradual Shrinking (refer to Step 3). Fig. 7 illustrates the procedure of current searching range expansion.

Referring to Fig. 7, the current search range can be slightly larger than the expected range, which causes Q_{pre} to produce some false hits that need to be filtered. For the final searching range which overflows only slightly, the number of the false hits can be controlled to a low-level. Compared with the efficiency of the expansion, the negative effects of this overflow and retrieval of false hits can be disregarded.

Example 1. Fig. 7 illustrates an example of current searching range expansion.

- The initial current searching range is NVP(q). $Q_{pre} = \{P_1\}$.
- The first expansion is represented by 1. $Q_{pre} = \{P_1, P_2, P_3, P_4, P_5, P_7, P_8\}$.
- The second expansion is represented by 2. $Q_{pre} = \{P_1, P_2, P_3, P_4, P_5, P_7, P_8, P_6, P_9, P_{10}, P_{11}, P_{12}, P_{13}, P_{14}, P_{15}\}$.

Step 3 *Validating Objects* and *Gradual Shrinking*: Essentially, Validating Objects is the distance comparison between range e and dis_{max} in Q_{pre}. It is normally iterative with current searching range expansion or Gradual Shrinking. If $e \leqslant dis_{max}$, the current searching range overflows the expected searching range. According to Property 2, we do not need to do an expansion at this moment. Although dis_{max} overflows the searching range, the algorithm could not be terminated according to Property 4, which indicates there could be the outside adjacent generators of other objects in Q_{pre} within the searching range. Then the current searching range needs to be shrunk naturally by discarding the current dis_{max} and go to next one in Q_{pre}. On the other hand, if $e > dis_{max}$, this means that the current searching range is inside the expected searching range. According to Property 3, the rest of the candidates in Q_{pre} do not need to be tested, then the current searching range expansion will be invoked again. When the algorithm is terminated, Q_{pre} must have the following structure:

$$Q_{pre}\big(\ldots, P_x(\ldots, False), P_y(\ldots, True), \ldots\big)$$

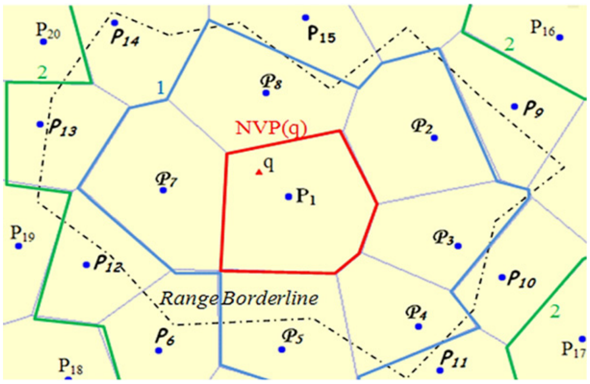


Fig. 7. Current Searching Range Expansion.

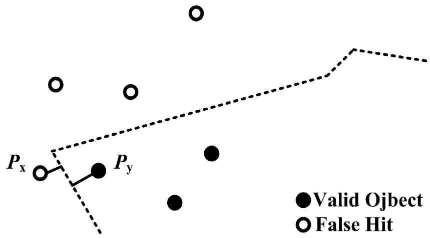


Fig. 8. Result of Q_{pre}.

Property 3. $Q_{pre} = (P_1, P_2, \ldots, P_n) \in \wp$, $i \in n$, $P_i \in \wp$, if $dis_{max} = dis_{net}(q, P_i) <$ Constant, then $\forall P_j(True) \in \wp$, $j \in n$, $dis_{net}(q, P_j) \ll$ Constant.

Property 4. $Q_{pre} = (P_1, P_2, \ldots, P_n) \in \wp$, $i, j \in n$, $P_i \in \wp$, then $dis_{max} = dis_{net}(q, P_i) > dis_{net}(q, P_j) \nRightarrow dis_{net}(q, P_{i'}) > dis_{net}(q, P_{j'})$, $P_{i'} \in \text{outAG}_{P_i}()$, $P_{j'} \in \text{outAG}_{P_j}()$.

The positions of P_x and P_y define the domain of the expected search range (refer to Fig. 8). Because Q_{pre} is an ordered queue, P_x is the nearest object out of the expected searching range to the query point, while P_y, is the furthest objects within the expected searching range.

4.4. VRS algorithm

The processing of VRS can be summarized as the extraction of candidate samples and the filtering of candidate samples. The former adds some sample objects of interest needed to be checked into Q_{pre}, while the latter finalizes the result precisely with a set of comparisons and filters the false hits. The pseudo of VRS algorithm is shown in Algorithm 1.

To clarify working principle of VRS, we explain the following example (refer to Fig. 9). The query point is symbolized as a triangle, and $\wp$ are a set of objects of interest which are expressed as dots in road networks. The range search query can be described as getting all objects of interest within 12 kms which is recognized as the expected searching range. Initially Q_{pre} is initialized without any candidate.

Example 2. Network range search using VRS:
Firstly, according to *Locating Query Point*, the contain(q) function is implemented to define the query position in NVD, then P_1 is retrieved into Q_{pre} with NULL value $dis_{net}(q, P_1)$. After that, INE is invoked to get this network distance, which is assigned to $dis_{max}()$.

$$Q_{pre} = \{P_1\}, \qquad dis_{max} = dis_{net}(P_1, q)$$

Secondly, *Validating Objects* and *Gradual Shrinking* suggests that if $e \leqslant dis_{max}()$, then the inside/outside property of P_1 will be set to false meaning that no object of interest is within the expected searching range, the algorithm of VRS will terminate. Otherwise, if $e > dis_{max}()$, then *Current Searching Range Expansion* will be invoked, which means the searching range needs to be expanded.

$$dis_{max} < e \quad \Rightarrow \quad \textit{Current Searching Range Expansion}$$

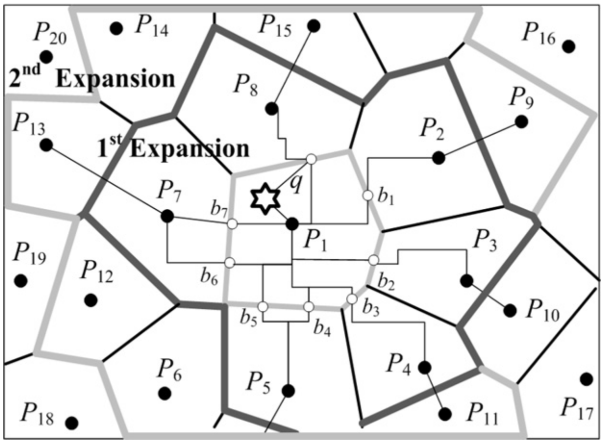


Fig. 9. An example of VRS.

Algorithm 1 VRS

Input: query point: q, searching range: e
Output: Q_{pre}

$P_i \leftarrow Contain_e(q)$ /*Locating the query point*/
$Q_{pre} \leftarrow Q_{pre} \cup P_q$ /*Add P_q into Q_{pre}*/
$dis_{max} \leftarrow dis_{net}(P_i, q)$ /*NVD(q) = NVD(P_i)*/
if $e < dis_{net}(q, P_i)$ **then**
  Set the property of current P_i to false
  Return $Q_{pre} \leftarrow \emptyset$
else
  repeat
    Add all the neighbors of object into Q_{pre} with true property
    if P already exists in Q_{pre} **then**
      Discard it
    end if
    Sort Q_{pre} in descending order according to $dis_{net}(q, P)$
    if Exist P_x whose property is false after P_y in Q_{pre} /*P_y is the new object inserted into Q_{pre}*/ **then**
      Set Inside/outside property of P_y to false
      $dis_{max} \leftarrow dis_{net}(P_j, q)$ /*P_j has the largest dis_{net} to q with true property*/
    end if
    while $e < dis_{max}$ **do**
      Set the property of current P to false
      Test next P in Q_{pre}
    end while
  until no P added to Q_{pre}
  Return Ps whose inside/outside property is true in Q_{pre}
end if

Thirdly, after implementing the *Current Searching Range Expansion*, the current searching range expanded from point NVP(P_1) to the neighbor NVPs of P_1, namely, P_2, P_3, P_4, P_5, P_7, P_8, which refers to 1st *Current Searching Range Expansion* in Fig. 9. These objects are inserted into Q_{pre} as well as the network distances to q that can be calculated by using $dis_{net}(q, P_1)$ and some pre-calculated distances. Then Q_{pre} is sorted in a descending order to filter some candidates further than the current object. Then the largest dis_{net} in Q_{pre} will be assigned to dis_{max}.

$$Q_{pre} = \big(P_4(\ldots, True), P_3, P_5, P_2, P_8, P_7, P_1,\big), \qquad dis_{max} = dis_{net}(P_4, q)$$

Fourthly, *Validating Objects* will be invoked again. For $dis_{max} = dis_{net}(q, P_4) < e$, then the current searching range should shrink a little. So the inside/outside property of P_4 is set to false and test the next P_i in Q_{pre} until $e \geqslant dis_{max}$, then we call *Current Searching Range Expansion*/*Gradually Shrinking* and *Validating Objects* iteratively until no objects of interest are added into Q_{pre}.

$$dis_{max} > e \quad \Rightarrow \quad \textit{Gradual Shrinking}$$

$$Q_{pre} = \big(P_4(\ldots, False), P_3(\ldots, True), P_5, P_2, P_8, P_7, P_1,\big), \qquad dis_{max} = dis_{net}(P_3, q)$$

Current Searching Range Expansion only adds the neighbors of objects whose inside/outside property is true into Q_{pre}, which narrows the searching range in the next step. To avoid adding the same object of interest into Q_{pre}, we inspect the inside/outside property and check whether the object already exists in the Q_{pre}. The object with false property will be skipped over and the duplicated objects will be discarded. The termination condition of VRS is that no object of interest is added to the Q_{pre} any more, then all objects of interest with true inside/outside property in Q_{pre} are returned to the user.

5. Voronoi Continuous Range (VCR)

Range query should not be confined to static range search, but also needs to be feasible for continuous range queries. Our previous work, *Continuous Range Search* (CRS) as the only existing algorithm cannot avoid employing network segment division used by most continuous *k* nearest neighbors query processing methods (e.g. IE, UBA and eDAR), which performance depends on the number of intersections in the networks. Since the actual pattern of the road networks is normally extremely complex (numerous intersections), road segmentation creates a significant problem in performance.

In this section, we propose a novel approach, *Voronoi Continuous Range* (VCR), a search method based on VRS, which does not do road segmentation.

5.1. Problem definition

Firstly, we need to define continuous range search query.

Predefined trajectory continuous range search query is defined as retrieving all objects of interest on any point of a given trajectory in the networks. It is similar to continuous *k*NN query. The predefined path is a necessary element of this kind of query. Our previous work illustrates how CRS can solve the traditional continuous range search query properly.

S–D continuous range search query is defined as retrieving all objects of interest on any point during the moving of the query point from the start point (S) to the destination (D) in the networks. In this case, the moving trajectory of the query point is not predefined, which requires the method to have a better flexibility in the road network environment.

VCR is designed for both of the aforementioned queries.

5.2. Operational principle of VCR

For Q_{pre} generated in VRS, this does not involve only the valid objects in the expected searching range, but also includes a few outside objects close to the searching range. The domain of the expected searching range can be located by P_x, P_y. P_x is the nearest object to the query point outside of the searching range, while P_y is the furthest one to the query point in the expected searching range.

Property 5. Given the Q_{pre} return by *VRS*, then P of $\min(|e - dis_{net}(P_x, q)|, |e - dis_{net}(P_y, q)|)$ is the priority whose inside/outside property can be changed, when the query point is moving.

In the continuous environment, when the query point is moving, it will cause a series of variations on the pattern of expected searching range in respect to the moving distance of the query point during the movement of the query point. Some objects could be moved out, others could be moved in. To decrease the frequency of updating, we need to know where the result can be changed and that is the position where the detection should be executed. According to Property 5, if the inside/outside property of P_x/P_y is unchanged, no object of interest can move in/out of the expected searching range, which provides us with a method to monitor the changes during the movement of the query point.

Definition 6. Detection point (dp) is a point used to predict the position where the range search result needs to be updated on the trajectory of the moving query in the networks. It can be expressed by a network distance to the current position of query point, $\min(|e - dis_{net}(P_x, q)|, |e - dis_{net}(P_y, q)|)$, P_x has minimum dis_{net} to q with false value, P_y has maximum dis_{net} to q with true value.

At every *dp*, VRS will be invoked to detect whether there is an object of interest that can be added/removed from Q_{pre}.

Because the velocity, the moving direction of the query point and the pattern of the road networks are ignored, the changes at the detection point cannot be guaranteed, and the number of *dp* will be larger than the split point used by other continuous approaches. But VCR focuses on the moving distance of the query point that can be calculated and monitored easily, meaning that the continuous query processing is put in a more simple environment by removing most of the influencing factors, which provides VCR with the capability to process both predefined trajectory and S–D continuous range search queries.

Definition 7. Critical point (cp) is a special object of interest, which satisfies $dis_{net}(q, P_x) = e$, $P_x \in \wp$.

Fig. 10 shows that P_1 and P_2 are critical points. If there is a critical point in the Q_{pre}, then $\min(|e - dis_{net}(P_x, q)|, |e - dis_{net}(P_y, q)|) = 0$. So the detection point cannot be found at this moment. To solve this problem, a new variable, *Precision Factor* (pf) needs to be defined.

Definition 8. Precision factor (pf) is a minimum metric unit, which is determined by the precision of the network distance stored in the database.

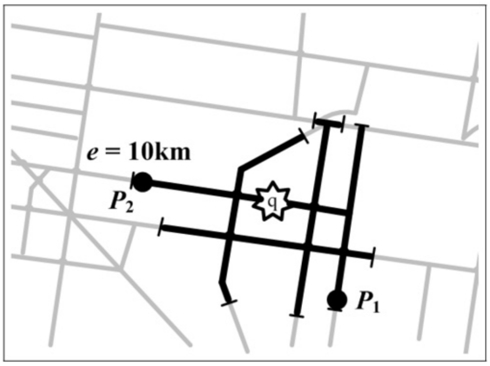


Fig. 10. Critical points.

For example, if the network distance stored in the database is 12.47 km, then $pf = 0.01$ km. If $\exists cp \in Q_{pre}$, then dp is no longer calculated by $\min(|e - dis_{net}(P_x, q)|, |e - dis_{net}(P_y, q)|)$. In this case, $dp = pf$, after updating Q_{pre} at the dp, cp will be cleared. To summarize, pf is used to detect the moving trend of a cp.

Using pf appears to be a time consuming method, but actually, when we examine the processing which involve, pf, we find its performance quite acceptable. Assuming $\min(|e - dis_{net}(P_x, q)|, |e - dis_{net}(P_y, q)|) = a$ and $\max(|e - dis_{net}(P_x, q)|, |e - dis_{net}(P_y, q)|) = A$. In the worst case, the times of calculation to find the next object is $n = 1 + \log_2 (A - a)/2pf$, and during this processing, the only data that can be slightly changed is the network distance and the inside/outside property of the critical point. In other words, there is no other object of interest moving in/out of the range, except this critical point. Moreover, the critical point is rare in real cases.

5.3. VCR algorithm

The pseudo code of VCR is shown below and the main step of our proposed Voronoi-based continuous range will be illustrated as follows (see Algorithm 2):

Algorithm 2 VCR

Input: start point: S, destination point: D, (pre-defined path: SD), searching range: e
Output: a set of Q_{pre}
VRS (S, e)
$Total_{dis} \leftarrow dis_{net}(S, D)$
repeat
  Find P_x and P_y from Q_{pre} /*P_x is the nearest object to q out e; P_y is the furthest object to q in e*/
  $dp \leftarrow \min(|e - dis_{net}(P_x, q)|, |e - dis_{net}(P_y, q)|)$
  if $dp \neq 0$ /*no object is a critical point*/ **then**
    VRS (dp, e)
    Update $Total_{dis} \leftarrow disnet(dp, D)$
  else
    $dp = dp + pf$
    Update the information of critical point at dp
    Update $Total_{dis} \leftarrow disnet(dp, D)$
  end if
until $Total_{dis} = 0$

Step 1 For a given start point S and end point D or a path SD, we define the value of $Total_{dis} = dis_{net}(S, D)$, which will be used to evaluate the position of query point to the destination.
Step 2 Find an object of interest, P_x, with the maximum network distance in the range, while finding one, P_y, with the minimum network distance out of the range from Q_{pre} will be used to calculate the detection point where the result of the range search can be changed. If there is no P within the range or outside the range, then the corresponding network distance is recorded as $dis_{net} = 0$.
Step 3 Find a detection point using $dp = \min(|e - dis_{net}(P_x, q)|, |e - dis_{net}(P_y, q)|)$.
Step 4 At the detection point, test any object of interest as a critical point to determine whether or not to employ pf for the next detection point. Update the corresponding information at the detection point.
Step 5 Repeat steps 3 and 4 until the query point reaches to the end point, $Total_{dis} = 0$.

Example 3. The initial search range is 12 km and $Total_{dis} = dis_{net}(S, D) = 18.8$, then at point S, the data in the Q_{pre} is shown in Table 2. Learning through observation and comparison, we get the first detection point $dp_1 = \min(|12 - 11.64|,$

Table 2
Q_{pre} at start point.

False objects in Q_{pre}		True objects in Q_{pre}	
P_x	Properties	P_y	Properties
P_{11}	(15.23, False)	$\boldsymbol{P_{12}}$	**(11.64, True)**
P_9	(14.54, False)	P_5	(9.63, True)
P_6	(14.20, False)	P_2	(8.37, True)
P_{14}	(13.83, False)	P_8	(8.19, True)
P_{13}	(13.53, False)	P_7	(7.85, True)
P_{15}	(13.20, False)	P_1	(5.06, True)
P_4	(12.94, False)		
$\boldsymbol{P_3}$	**(12.82, False)**		

Table 3
Performance evaluation parameters.

Parameter	Description
Comparison	The times of comparison between two variables in the algorithm
False hits	Number of invalid objects retrieved by the algorithm
Expansion area	The area has been expanded by the network expansion method
Segments	A subdivision of predefined path
sp	A split point in CRS, the result has to be changed at split point
dp	A detection point invoking VRS in VCR, the result can be changed at detection point

$|12 - 12.82|) = \min(0.36, 0.82) = 0.36$. It means that when the query point q moves 360 metres from the start point, we will do a range search using VRS to update the information of Q_{pre}.

At the detection point dp_1, the $dis_{net}()$ of P_{12} and $Total_{dis}$ will be updated, if P_{12} is a critical point, $disnet(P_{12}, q) = e = 12$ km and $Total_{dis} \neq 0$ (not arrive at destination point), then pf which equals 0.01 will intervene to test P_{12} will move further into the range or further out from the expected searching range in the next moment. So $dp_2 = pf = 0.01$, where the network distance and inside/outside property of P_{12} will be updated. On the other hand, if the $dis_{net}(P_{12}, q) \neq e$ at dp_1, then we will go through Q_{pre} to find the next detection point by estimating the new value of $\min(|e - dis_{net}(P_x, q)|, |e - dis_{net}(P_y, q)|)$.

If all information in Q_{pre} has been updated, we repeat the above procedure, until the query reaches the destination, $Total_{dis} = 0$ when the VCR will terminate.

Comparing to processing the pre-defined path which is just in a single NVP, processing the path through multiple NVPs will necessitate a massive updating. So if the pre-defined path is within a NVP, then the performance of VCR will be improved remarkably.

6. Performance evaluation

We carried out several experiments to evaluate the performance of VRS and VCR. The datasets provided by *Whereis*® (http://www.whereis.com.au), represent networks of thousands of links and nodes of the road system in Melbourne, Australia. We compare the performance of our proposed VRS with the existing work RER, RNE. We also compare our proposed VCR with the previous work CRS. We performed 20 sets of tests for each experiment to calculate the average for the tested data, and the range size was varied from 1 km to 100 km. All evaluated parameters are explained in Table 3.

6.1. Experimental results of VRS

The most important aspects that will affect the efficiency of VRS are Q_{pre} which reflects the utilization rate of the storage space. So we examine the percentage of false hits in Q_{pre} to evaluate the efficiency of VRS.

Fig. 11 shows the percentage of the false hits in both high-density and low-density environments. Intuitively, the percentage of false hits decreases dramatically when the searching range e increases. And the high-density objects will have a lower percentage than will the low-density objects. In conclusion, VRS is more suitable for large range searching of high-density objects.

In this section, we present a comparison of results for VRS and RER and RNE, on various different scenarios, including, changing the density of the object and altering the searching range to a different location and resizing it. These experiments are considered from three perspectives, namely, times of comparison of dis_{net} with e, the number of false hits, and the expansion area.

Figs. 12 and 13 show the comparison results in low and high density environments, respectively. It is obvious that the times of comparisons by VRS is far less than that of RER and RNE, because when RER/RNE process the network expansion segment by segment, it compares the value of $dis_{net}(n, q)$ (n is a node point which can be either an object of interest or

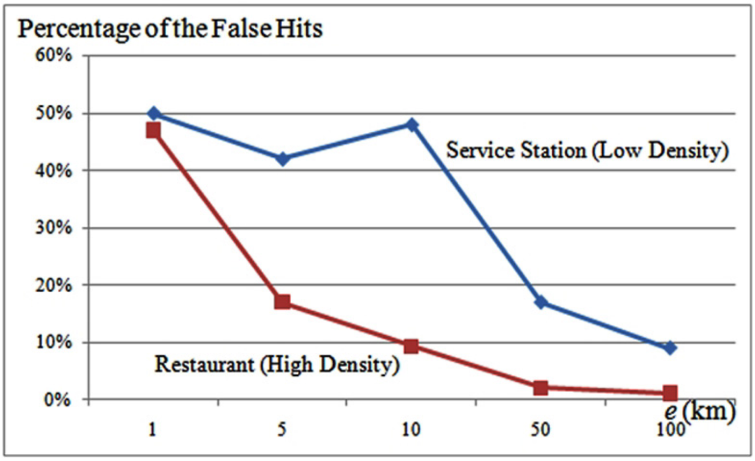
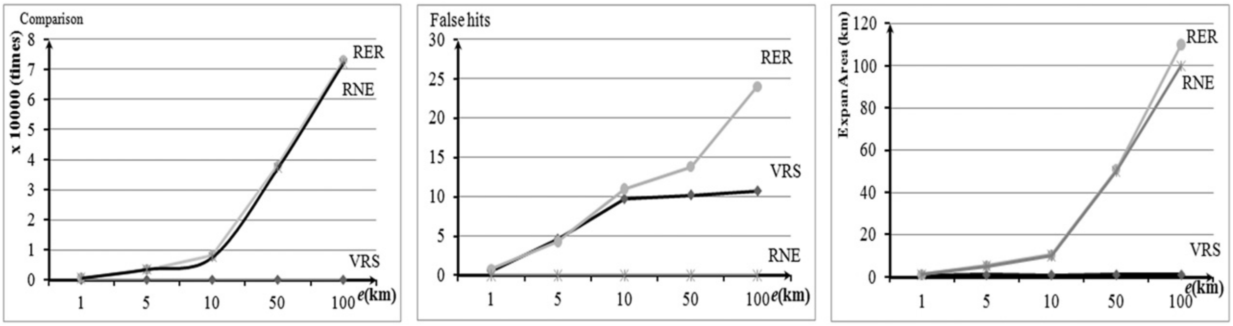
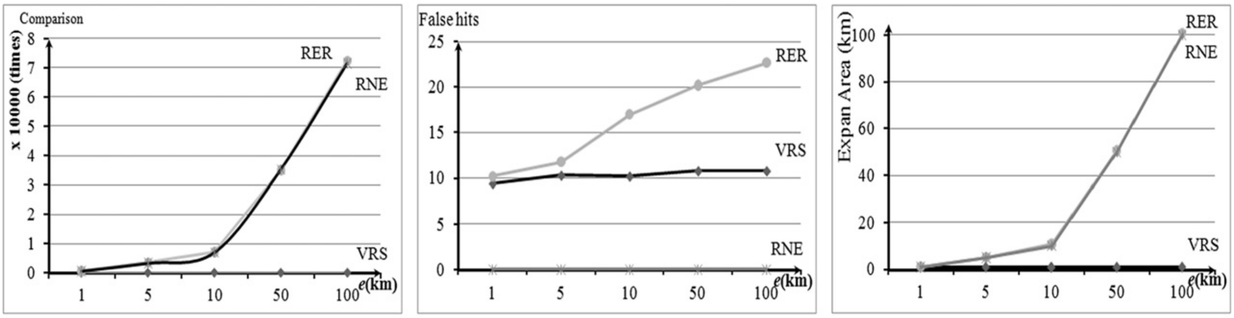


Fig. 11. Percentage of false hits in Q_{pre}.



(a) Times of Comparison (b) Number of False Hits (c) Expansion Area

Fig. 12. VRS vs. RER vs. RNE in low density environment (10 objects/10 sq.km).



(a) Times of Comparison (b) Number of False Hits (c) Expansion Area

Fig. 13. VRS vs. RER vs. RNE in high density environment (100 objects/10 sq.km).

an intersection node in networks) with e. So even if we ignore the Euclidean range search step in RER and R-tree indexing step in RNE, examining all segments in the Euclidean range is already a difficult task, especially, when e is very large. Moreover, in the Euclidean range search step of RER, the lengths of retrieved road segments are larger than e; that is why the expansion area in RER overflows e lightly. Since RNE applies the network expansion approach first, the expansion area will not exceed e. Both RER and RNE are not comparable with VRS, as the network expansion for VRS happens only within the first polygon to retrieve the distance from q to the borders, so no matter how large the range is, our times of comparison will remain at a lower level. The extra comparisons in VRS relates to the *locating range borderline* step. So when the range increases, the times of comparison will rise slightly.

We can also see that the difference in the number of false hits between VRS and RER is not much, although VRS is ahead slightly. Actually, even though the number of false hits in RER is not relevant to e, it is not self-dependent either.

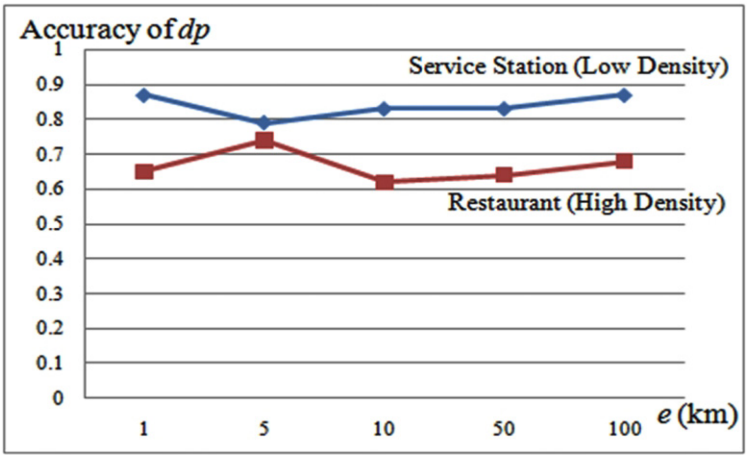


Fig. 14. Accuracy of *dp*.

Table 4
VCR vs. CRS.

VCR vs. CRS												
	Low Density environment 10 objects/10 sq.km											
	$e = 1$ km				$e = 10$ km				$e = 100$ km			
	VCR	CRS	VCR	CRS	VCR	CRS	VCR	CRS	VCR	CRS	VCR	CRS
Segments	1	532	1	4028	1	532	1	4028	1	532	1	4028
sp(CRS)	–	33	–	390	–	465	–	4379	–	4308	–	38745
dp(VCR)	67	–	382	–	826	–	3659	–	7948	–	78976	–
	High Density environment 100 objects/10 sq.km											
	$e = 1$ km				$e = 10$ km				$e = 100$ km			
	VCR	CRS	VCR	CRS	VCR	CRS	VCR	CRS	VCR	CRS	VCR	CRS
Segments	1	532	1	4028	1	532	1	4028	1	532	1	4028
sp(CRS)	–	325	–	3885	–	4530	–	448764	–	46971	–	400653
dp(VCR)	638	–	4007	–	8487	–	3779	–	78754	–	764399	–

The only influencing fact of the number of false hits is the degree of similarity of dis_{net} and dis_e. So, in good conditions, the number of false hits in RER can be very small, and vice versa. For the utilization of the sorting algorithm and inside/outside property, the number of false hits is dominated in a reasonable range and the external factors, have no effect; only the size of the Voronoi polygon which is determined by the density of the objects of interest.

As mentioned previously, RNE reduces the number of false hits, and is better than VRS, especially in a high density environment. But when we examine the algorithm of VRS carefully, we can find that false hits in VRS are quite different from those in RER. In fact, false hits in VRS are only for locating the range borderline and do not affect the performance of VRS at all. In summary, VRS performs very well in these three aspects, and in particular VRS outperforms RER and RNE.

6.2. Experimental results of VCR

The most frequent concept in VCR is *dp* that tries to detect the changes during the movement of the query point. But sometimes the changes do not happen at *dp*, so the accuracy of *dp* is an important parameter for estimating the performance of VCR.

Fig. 14 shows the rate of the accuracy of *dp* in both high-density and low-density environments. Intuitively, the rate of the accuracy of *dp* always remains at a good level when the searching range e increases. And the high-density objects will have a lower percentage than the low-density objects. So the density is the main factor that will affect the accuracy of *dp*.

Because of very limited existing works on continuous range using NVD, we compare only VCR and CRS, by offering the results of several experiments using different scenarios, including: changing the density of the object and altering the searching range to different location and resize it. These experiments focus on: number of comparison of dis_{net} with e, number of false hits and the proportion of the expansion area.

Table 4 shows the comparison results between VCR and CRS. It is obvious that VCR does not divide the path into segments, while CRS has to implement the path segmentation to avoid changes when moving on the segment. If there is an intersection in the middle of the segment, we cannot guarantee that the changes of dis_{net} are unidirectional. No segmentation gives VCR a better performance and applicability.

On the other hand, the number of *dp* in VCR is larger than the split node in CRS most of the time. Unlike the split node, detection points do not ensure that there will be a change in that point. If the accuracy of split node is 100%, then *dp* maintains the accuracy at a lower level. But the number of *dp* is very similar with the corresponding value of split node, for VCR does not focus on the details of the network.

According to the result in Table 4, VCR is not suitable for a wide range and high density environment and when the value of range *e* approximate to the path length, CRS is strongly competitive.

7. Conclusions and future work

After analyzing the existing range and continuous range search methods, and evaluating their disadvantages, we find that the main problem lies in the expansion technique. If we can shrink the expansion area, much computation time will be saved. In order to offer a better solution, we use a Network Voronoi Diagram as the basis for our methods, named *Voronoi-based Range Search* (VRS) and *Voronoi Continuous Range* (VCR). We utilize some new properties of NVD and develop three main steps for VRS, namely, *Current Searching Range Expansion*, *Query Position Definition* and *Locating Range Borderline*, to remove unnecessary expansion and decrease the number of false hits. Moreover, Q_{pre}, which stores the candidates, was also introduced as the basic data structure for VCR, which efficiently retrieves the objects of interest within the searching range for the moving user. Unlike other continuous algorithms, VCR focuses only on the moving distance of the query point instead of the details of networks. When the result of Q_{pre} is likely to change, we employ VRS at detection point to update the searching results. Our experiments show that both VRS and VCR outperform their competitors in most scenarios.

To sum up, VRS and VCR retrieve the expected results for network range search queries and provide better performances than do the existing approaches and it is also a solution for the new type of continuous range search. VRS stores a large set of pre-computed components in the database, which increases the storage space and indexing time. It is therefore imperative to address the storage requirements of these components [7]. We plan to apply an object-relational technique [16,17], and global indexing [24,21] for an efficient storage.

Although the performance of VRS and VCR outstrips their competitors, due to the use of the property of Voronoi diagram, performance can be expected to further increase by employing parallel techniques. We also plan to investigate the use of parallelism [27,22,23,25] to process spatial and mobile queries more efficiently. Performance of mobile queries is expected to be enhanced through data broadcast [30,29], instead of server processing [9]. Another aspect that may be incorporated into mobile continuous query processing is through mining mobile users' movements [5,6,26,1].

Acknowledgment

This research has been partially funded by the Australian Research Council (ARC) Discovery Project (Project No. DP0987687).

References

[1] O. Daly, D. Taniar, Exception rules mining based on negative association rules, in: Proceedings of the International Conference on Computational Science and Its Applications (ICCSA), part IV, in: Lecture Notes in Comput. Sci., vol. 3046, Springer, 2004, pp. 543–552.
[2] E.W. Dijkstra, A note on two problems in connection with graphs, Numer. Math. 1 (22) (1959) 269–271.
[3] P. Fülöp, K. Lendvai, T. Szálka, S. Imre, S. Szabó, Accurate mobility modeling and location prediction based on pattern analysis of handover series in mobile networks, Mobile Inf. Syst. 5 (3) (2009) 255–289.
[4] J. Goh, D. Taniar, Mobile data mining by location dependencies, in: Proceedings of the 5th International Conference on Intelligent Data Engineering and Automated Learning – IDEAL, in: Lecture Notes in Comput. Sci., vol. 3177, Springer, 2004, pp. 225–231.
[5] J. Goh, D. Taniar, Mining frequency pattern from mobile users, in: Proceedings of the 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES), part III, in: Lecture Notes in Comput. Sci., vol. 3215, Springer, 2004, pp. 795–801.
[6] J. Goh, D. Taniar, Mining parallel patterns from mobile users, Internat. J. Business Data Commun. Netw. 1 (1) (2005) 50–76.
[7] L. Gómez, B. Kuijpers, B. Moelans, A. Vaisman, A survey of spatio-temporal data warehousing, Internat. J. Data Warehousing Min. 5 (3) (2009) 28–55.
[8] A. Guttman, R-trees: A dynamic index structure for spatial searching, in: Proceedings of ACM SIGMOD, ACM Press, June 1984, pp. 47–57.
[9] J. Jayaputera, D. Taniar, Data retrieval for location-dependent queries in a multi-cell wireless environment, Mobile Inf. Syst. 1 (2) (2005) 91–108.
[10] M.R. Kolahdouzan, C. Shahabi, Voronoi-based *k* nearest neighbor search for spatial network databases, in: Proceedings of the 30th VLDB, Morgan Kaufmann Publishers Inc., Toronto, Canada, August 2004, pp. 840–851.
[11] M.R. Kolahdouzan, C. Shahabi, Alternative solutions for continuous *k* nearest neighbor queries in spatial network databases, GeoInformatica 9 (4) (2005) 321–341.
[12] D. Komaki, K. Ohnishi, Y. Arase, T. Hara, G. Hattori, S. Nishio, Design and implementation of a Click–Search interface for web browsing using cellular phones, Internat. J. Web Grid Serv. 5 (1) (2009) 66–84.
[13] Z. Mammeri, F. Morvan, A. Hameurlain, N. Marsit, Location-dependent query processing under soft real-time constraints, Mobile Inf. Syst. 5 (3) (2009) 205–232.
[14] A. Okabe, B. Boots, K. Sugihara, S.N. Chiu, Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, second edition, John Wiley and Sons Ltd., West Sussex, England, 2000.
[15] D. Papadias, J. Zhang, N. Mamoulis, Y. Tao, Query processing in spatial network databases, in: Proceedings of the 29th VLDB, Morgan Kaufmann Publishers Inc., Berlin, Germany, September 2003, pp. 802–813.
[16] J.W. Rahayu, E. Chang, T. Dillon, D. Taniar, A methodology for transforming inheritance relationships in an object-oriented conceptual model to relational tables, Inf. Software Tech. 42 (8) (2000) 571–592.
[17] J.W. Rahayu, E. Chang, T. Dillon, D. Taniar, Performance evaluation of the object-relational transformation methodology, Data Knowledge Engrg. 38 (3) (2001) 265–300.

[18] M. Safar, K nearest neighbor search in navigation systems, Mobile Inf. Syst. 1 (3) (2005) 207–224.
[19] M. Safar, D. Ebrahimi, eDAR algorithm for continuous KNN queries based on PINE, Internat. J. Inf. Tech. Web Engrg. 1 (4) (2006) 1–21.
[20] M. Safar, D. Ebrahimi, D. Taniar, Voronoi-based reverse nearest neighbor query processing on spatial networks, Multimedia Syst. 15 (5) (2009) 295–308.
[21] D. Taniar, J.W. Rahayu, A taxonomy of indexing schemes for parallel database systems, Distrib. Parallel Databases 12 (1) (2002) 73–106.
[22] D. Taniar, J.W. Rahayu, Parallel database sorting, Inform. Sci. 146 (1–4) (2002) 171–219.
[23] D. Taniar, J.W. Rahayu, Parallel sort-merge object-oriented collection join algorithms, Internat. J. Comput. Syst. Sci. Engrg. 17 (3) (2002) 145–158.
[24] D. Taniar, J.W. Rahayu, Global parallel index for multi-processors database systems, Inform. Sci. 165 (1–2) (2004) 103–127.
[25] D. Taniar, R. Tan, C. Leung, K. Liu, Performance analysis of "Groupby-After-Join" query processing in parallel database systems, Inform. Sci. 168 (1–4) (2004) 25–50.
[26] D. Taniar, J. Goh, On mining movement pattern from mobile users, Int. J. Distrib. Sensor Netw. 3 (1) (2007) 69–86.
[27] D. Taniar, C.H.C. Leung, J.W. Rahayu, S. Goel, High Performance Parallel Database Processing and Grid Databases, John Wiley and Sons Ltd., ISBN 978-0-470-10762-1, October 2008, 576 pp.
[28] Y. Tao, D. Papadias, Q. Shen, Continuous nearest neighbor search, in: Proceedings of the 28th VLDB, Morgan Kaufmann Publishers Inc., Hong Kong, China, August 2002, pp. 287–298.
[29] A. Waluyo, B. Srinivasan, D. Taniar, Optimal broadcast channel for data dissemination in mobile database environment, in: Proceedings of the 5th International Workshop on Advanced Parallel Programming Technologies, APPT, in: Lecture Notes in Comput. Sci., vol. 2834, Springer, 2003, pp. 655–664.
[30] A. Waluyo, B. Srinivasan, D. Taniar, A taxonomy of broadcast indexing schemes for multi channel data dissemination in mobile database, in: Proceedings of the 18th International Conference on Advanced Information Networking and Applications (AINA), vol. 1, IEEE Computer Society, 2004, pp. 213–218.
[31] A. Waluyo, B. Srinivasan, D. Taniar, Research in mobile database query optimization and processing, Mobile Inf. Syst. 1 (4) (2005) 225–252.
[32] K. Xuan, G. Zhao, D. Taniar, B. Srinivasan, Continuous range search query processing in mobile navigation, in: Proceedings of the 14th ICPADS 2008, IEEE, Melbourne, Victoria, Australia, December 2008, pp. 361–368.
[33] K. Xuan, G. Zhao, D. Taniar, B. Srinivasan, M. Safar, M.L. Gavrilova, Network Voronoi Diagram based range search, in: Proceedings of the IEEE 23rd International Conference on Advanced Information Networking and Applications (AINA), 2009, pp. 741–748.
[34] A. Yamazaki, A. Koyama, J. Arai, L. Barolli, Design and implementation of a ubiquitous health monitoring system, Internat. J. Web Grid Serv. 5 (4) (2009) 339–355.
[35] G. Zhao, K. Xuan, J.W. Rahayu, D. Taniar, M. Safar, M. Gavrilova, B. Srinivasan, Voronoi-based continuous k nearest neighbor search in mobile navigation, IEEE Trans. Ind. Electronics 56 (2009) (online since June 2009).
[36] G. Zhao, K. Xuan, D. Taniar, M. Safar, M. Gavrilova, B. Srinivasan, Multiple object types KNN search using network Voronoi diagram, in: Proceedings of the International Conference on Computational Science and Its Applications (ICCSA), part II, in: Lecture Notes in Comput. Sci., vol. 5593, Springer, 2009, pp. 819–834.