# Continuous Evaluation of Monochromatic and Bichromatic Reverse Nearest Neighbors*

James M. Kang[1], Mohamed F. Mokbel[1], Shashi Shekhar[1], Tian Xia[2], Donghui Zhang[2]

[1]Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN

[2]College of Computer and Information Science, Northeastern University, Boston, MA

## Abstract

*This paper presents a novel algorithm for Incremental and General Evaluation of continuous Reverse Nearest neighbor queries (IGERN, for short). The IGERN algorithm is general as it is applicable for both the monochromatic and bichromatic reverse nearest neighbor queries. The incremental aspect of IGERN is achieved through determining only a small set of objects to be monitored. While previous algorithms for monochromatic queries rely mainly on monitoring six pie regions, IGERN takes a radical approach by monitoring only a single region around the query object. The IGERN algorithm clearly outperforms the state-of-the-art algorithms in monochromatic queries. In addition, the IGERN algorithm presents the first attempt for continuous evaluation of bichromatic reverse nearest neighbor queries. The computational complexity of IGERN is presented in comparison to the state-of-the-art algorithms in the monochromatic case and to the use of Voronoi diagrams for the bichromatic case. In addition, the correctness of IGERN in both the monochromatic and bichromatic cases are proved. Extensive experimental analysis shows that IGERN is efficient, is scalable, and outperforms previous techniques for continuous reverse nearest neighbor queries.*

## 1 Introduction

A recent trend of sensor networks and location-based environments has been on the rise in the past decade. These systems have been assimilating into real world applications such as the enhanced 911 services, army strategic planning, retail services, and mixed-reality games. The continuous movement of data objects within these applications calls for new query processing techniques that scale up with the high rates of location updates. While there have been numerous work for addressing continuous range queries (e.g., see [5, 8, 13, 15, 19]) and nearest neighbor queries (e.g., see [9, 17, 23, 27, 31]), there is still a lack of research in addressing continuous reverse nearest neighbor queries.

There are two cases of Reverse Nearest Neighbor queries (RNN), namely, *monochromatic* RNN and *bichromatic* RNN [12]. In the *monochromatic* RNN, all moving data and query objects are of the same type. Thus, a data object $o$ is considered a reverse nearest neighbor to a query object $q$ if there does not exist another data object $o'$ where $dist(o, o') < dist(o, q)$. Applications of the continuous *monochromatic* RNN include mixed reality games (e.g., Botfighters) where the objective is to shoot only those players that are nearest to you. Thus, each player should monitor her own reverse nearest neighbors to avoid their shootings. In the *bichromatic* RNN, there are two distinct object types $A$ and $B$. Thus, a data object of type $B$, $o_B$, is considered a reverse nearest neighbor to a query object of type $A$, $q_A$, if there does not exist another object of type $A$, $o'_A$, where $dist(o_B, o'_A) < dist(o_B, q_A)$. Applications of the continuous *bichromatic* RNN include army strategic planning where a medical unit $A$ in the battlefield is always in search for wounded soliders of type $B$ in which $A$ is their nearest medical unit. RNN queries are also crucial in data mining applications where the RNNs of a query point $q$ are those objects on which $q$ has significant influence [12, 18].

Most of the previous work on reverse nearest neighbor queries focus on snapshot queries in static environments, i.e., the continuous movement of both the query and data objects are not taken into account (e.g., see [1, 3, 12, 21, 22, 24, 29, 30]). Up to the authors' knowledge, the CRNN algorithm [26] is the only attempt to evaluate *continuous* RNN queries. However, CRNN is applicable only to *monochromatic* queries where there is no direct extension for the case of *bichromatic* RNN. The main idea of CRNN is to divide the whole spatial space into six pie regions where each region is monitored independently for potential reverse nearest neighbors. Such idea has been widely employed for most of the snapshot RNN algorithms (e.g., see [21]) as it is based on the theoretical foundation that there can be up to six answers for any *monochromatic* RNN query [21].

In this paper, we present a novel algorithm for Incremental and General Evaluation of continuous Reverse Near-

*James M. Kang is supported by NSF IGERT, Shashi Shekhar is partially supported by NSF SEI Award 431141, and Donghui Zhang is partially supported by NSF CAREER Award IIS-0347600.

1-4244-0803-2/07/$20.00 ©2007 IEEE.

est neighbor queries (IGERN, for short). IGERN is a unified framework that is applicable to both the *monochromatic* and *bichromatic* RNN queries. The IGERN algorithm goes beyond the idea of six pies in evaluating *monochromatic* RNN queries. Furthermore, the IGERN algorithm provides the first attempt for continuous evaluation of *bichromatic* RNN queries. The main idea of both the *monochromatic* and *bichromatic* IGERN algorithms is to initially identify a single region $r$ around the query object and a set of objects $S$ such that *only* $r$ and $S$ need to be monitored to trigger subsequent changes of the answer. The incremental aspect of IGERN comes from the fact that each execution instance of IGERN updates the shape of $r$ and the objects in $S$. Then, subsequent executions of IGERN will need to monitor only $r$ and $S$ rather than monitoring the whole space. With its incremental nature, IGERN is a scalable algorithm that scales up for large numbers of moving objects and queries in highly dynamic environments. In general, the contributions of this paper can be summarized as follows:

1. We propose the IGERN algorithm for *monochromatic* continuous RNN queries that goes beyond the traditional method of dividing the space into six pie regions.
2. We propose the first *bichromatic* continuous RNN algorithm within the IGERN framework.
3. We prove the correctness of IGERN by proving its: (a) accuracy, i.e., a returned result by IGERN is an exact RNN, and (b) completeness, i.e., IGERN returns all possible RNNs.
4. We give analytical and experimental evidence that IGERN outperforms previous approaches for both *monochromatic* and *bichromatic* RNNs.

The rest of this paper is organized as follows. Section 2 highlights the related work. The *monochromatic* and *bichromatic* algorithms of IGERN are described in Sections 3 and 4, respectively. The correctness proof of IGERN is given in Section 5. Section 6 gives an analytical analysis of IGERN. Experimental evidence that the IGERN algorithm outperforms previous algorithms is given in Section 7. Finally, Section 8 concludes the paper.

## 2 Related Work

There is a recent interest in developing new continuous query processors to cope with the recent advances in dynamic location-aware environments [10, 14]. As a result, new algorithms have been developed for various types of continuous location-based queries, e.g., continuous range queries [5, 8, 13, 15, 19], continuous nearest neighbor queries [9, 17, 23, 27, 31], and continuous aggregates [7, 11]. Although reverse nearest neighbor queries are of the same importance as these query types, there is not much work in developing efficient algorithms for continuous reverse nearest neighbor queries.

Various algorithms are proposed for snapshot reverse nearest neighbors (RNNs) in different environments, e.g., euclidian space [12, 21, 22, 28], metric space [1, 24], high-dimensional space [20], ad-hoc space [29], and large graphs [30]. In this paper, we mainly focus on the euclidian space in which it is proved that there are at most six reverse nearest neighbors for the *monochromatic* case [21]. By utilizing this property, a filter-refine approach has been introduced by dividing the spatial space into six pie regions. Then, six nearest neighbor objects (one object in each pie) are used as filters to limit the search space. A completely different approach, denoted as TPL [22], relies mainly on recursively filtering the data by finding perpendicular bisectors between the query point and its nearest object.

Up to the authors' knowledge, there is only one algorithm, termed CRNN [26], for continuous evaluation of reverse nearest neighbor queries. CRNN extends the idea of dividing the space into six pies, originally developed for snapshot queries [21], to dynamic environments. As a result, CRNN monitors each pie region along with six moving objects at every time interval. However, CRNN has two main disadvantages: (1) CRNN is limited to only *monochromatic* RNN queries and (2) CRNN always assumes a constant worst-case scenario at every time interval where it is assumed that there are always six RNNs. Such drawback comes from the fact that CRNN ignores the relation between the neighboring pies.

Our proposed algorithm, IGERN, avoids the drawbacks of CRNN by being applicable to both the *monochromatic* and *bichromatic* RNNs. In addition, IGERN adapts itself based on the current data to monitor only one closed region and less than six objects as opposed to constantly monitoring six regions and six objects in CRNN.

## 3 Continuous Evaluation of Monochromatic Reverse Nearest Neighbors

This section presents the IGERN algorithm for *monochromatic* reverse nearest neighbor queries. The IGERN algorithm maintains a grid data structure $G$ of $N \times N$ equal size cells. Each cell $c \in G$ keeps track of the set of objects that lie within the cell boundary. In general, the IGERN algorithm has two main steps, namely, the *initial* and *incremental* steps. The *initial* step is executed only once at the query issuing time $T_0$ while the *incremental* step is triggered every $T$ time units throughout the life time of the continuous query. The main idea is that the *initial* step reports the first query answer along with a bounded region and a set of objects to be monitored within the *incremental* step. Then, the *incremental* step continuously updates the query answer while changing the monitored region and the monitored set of objects. The *initial* and *incremental* steps are described in Sections 3.1 and 3.2, respectively, while Section 3.3 gives a general discussion about IGERN.

**Algorithm 1** Pseudo code for the Mono Initial Step

```
1: Function INITMONOIGERN(Query q)
2: RNNcand ← ∅, Mark all grid cells G as alive
   {Phase I: Bounded Region}
3: while (o_j ← the nearest object to q in the alive cells) ≠ NULL do
4:     RNNcand ← RNNcand ∪ o_j, b_j ← The ⊥ bisector of q and o_j
5:     Mark grid cells from b_j to the furthest boundaries from q as dead
6: end while
   {Phase II: Verification}
7: RNN ← RNNcand - {Objects that do not have q as the nearest object}
8: return RNN, RNNcand
```

**Algorithm 2** Pseudo code for Mono Incremental Step

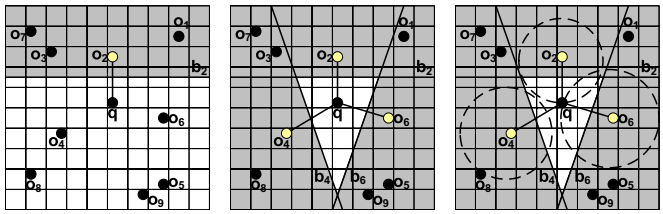
```
1: Function INCRMONOIGERN(Query q, set RNNcand)
2: if q or any objects in RNNcand have moved then
3:     Redraw the bisectors between q and all objects in RNNcand
4:     Only the cells between q and the bisectors are marked as alive.
5: end if
6: if there is any object o within the alive cells then
7:     Tighten the region as in Phase I of Algorithm 1 (Lines 3 to 6)
8:     For any two objects o_i, o_j ∈ RNNcand, remove object o_i from
       RNNcand only if dist(o_i, o_j) < dist(o_i, q)
9: end if
10: RNN ← RNNcand - {Objects that do not have q as nearest object}
11: return RNN, RNNcand
```

### 3.1 Step 1: Getting the Initial Answer

The *initial* step of IGERN has three main objectives: (1) Obtaining a bounded region $r$ around the query object $q$ which will be monitored in the *incremental* step, (2) Identifying a set of objects *RNNcand* that need to be monitored in the *incremental* step, and (3) Identifying the set of initial reverse nearest neighbor objects (*RNN* $\subseteq$ *RNNcand*) to $q$. Algorithm 1 gives the pseudo-code of the IGERN *initial* step. The input is the query object $q$ while the output consists of the two sets *RNN* and *RNNcand*. Initially, *RNNcand* is empty while all grid cells in the grid data structure $G$ are set as *alive*, i.e., every cell has the potential of containing reverse nearest neighbors of $q$ (Line 2 of Algorithm 1). Then, the *initial step* has the following two main phases:

**Phase I: Bounded Region.** This phase is concerned with the first two objectives of the *initial* step. The *bounded region* phase starts by finding the object $o_j$ as the nearest object to the query $q$ within all *alive* cells (Line 3 of Algorithm 1). Then, object $o_j$ is considered as a candidate to be a reverse nearest neighbor, i.e., $o_j$ is added to *RNNcand*. A bisector $b_j$ between $o_j$ and $q$ indicates that all objects between $b_j$ and the furthest space boundaries from $q$ would be closer to $o_j$ than $q$. Thus, all the grid cells between $b_j$ and these boundaries are marked as *dead*, i.e., there cannot be any reverse nearest neighbor to $q$ within these cells (Lines 4-5 of Algorithm 1). This phase continues to run until there are no objects within the *alive* cells. Figure 1 gives an example of the *initial* step with nine objects $o_1$ to $o_9$ and a query object $q$. $o_2$ is the nearest object to $q$ while $b_2$ is its corresponding bisector (Figure 1a). Thus, all cells above $b_2$ are shaded, i.e., marked as *dead*. Such process continues as $o_6$ followed by $o_4$ are identified as the nearest objects to $q$ within the *alive* cells and the bisectors $b_6$ and $b_4$ are drawn (Figure 1b). As there are no more objects within the *alive* cells, the candidate set becomes *RNNcand* $= \{o_2, o_4, o_6\}$.

**Phase II: Verification.** In this phase, we go through all objects in *RNNcand* and only those objects that $q$ is their nearest are considered as RNNs (Line 7 in Algorithm 1). In Figure 1c, the dotted circles indicate the nearest neighbor test for each object in *RNNcand*. Thus, *RNN* $= \{o_2, o_6\}$ where $o_2$ and $o_6$ are the reverse nearest neighbors to $q$.



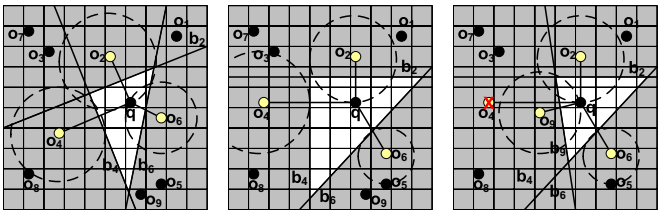

(a) First NN (b) Phase I (c) Phase II

*Figure 1:* Example of the *monochromatic initial* step

### 3.2 Step 2: Incremental Maintenance

The *incremental* step of IGERN is repetitively executed every $T$ time units. Algorithm 2 gives the pseudo-code of the *incremental* step. The input to this algorithm is the query object $q$ and the set *RNNcand* that came from either the *initial* step at time $T_0$ or a previous execution of the *incremental* step. Upon its execution, the *incremental* step checks for three different scenarios: (1) The query object $q$ moves to a new location (Line 2 in Algorithm 2), (2) At least one of the objects in *RNNcand* moves to a new location (Line 2 in Algorithm 2), and (3) A new object moves into the *alive* cells (Line 6 in Algorithm 2). If none of these scenarios took place, then the *incremental* step will only *verify* the current query answer in a similar way to the *verification* phase in the *initial* step (Line 10 in Algorithm 2) while the *RNNcand* set will not be changed. However, if any of these three scenarios took place, then the IGERN *incremental* step needs to perform more computations in order to efficiently maintain the query answer.

The *incremental* step starts by checking the first two events, i.e., if either the query object $q$ or any of the objects in the candidate list have moved (Line 2 in Algorithm 2). If this is the case, then new bisectors will be drawn from $q$ to the objects in *RNNcand*. Then, only the cells between $q$ and its bisectors are considered *alive* while all other grid cells are considered as *dead*. Figures 2a and 2b give the cases when only the query $q$ moves and all objects in *RNNcand* move, respectively. In both cases, the bisectors from $q$ to



(a) $q$ moves (b) $o_i$ move (c) new object

*Figure 2:* An example of the *monochromatic Incremental* step

the objects in *RNNcand* $o_2$, $o_4$, and $o_6$ are redrawn and the set of *alive* cells are adjusted.

Then, the *incremental* step checks the third scenario, i.e., a new object is found in an *alive* cell. This condition also captures the new *alive* cells that result from redrawing the bisectors upon the movement of any of the monitored objects. If there are no objects in the *alive* cells, then only the verification step is needed (Line 10 in Algorithm 2). In our example, $RNN = \{o_2, o_6\}$ in Figures 2a while $RNN = \{o_2\}$ in Figures 2b. However, if there is one or more objects in the *alive* cells, we will tighten the bounded region by finding the nearest objects within the *alive* cells and drawing the corresponding bisectors (Line 7 in Algorithm 2). Then, the list *RNNcand* will be cleaned by removing any object that could not be a reverse nearest neighbor (Line 8 in Algorithm 2). Finally, the answer is verified (Line 10 in Algorithm 2). Figure 2c depicts the case when $o_9$ moves inside an *alive* cell. The object $o_9$ is added to *RNNcand* while object $o_4$ is removed. Also, the bisector $b_9$ is drawn and the *shaded* cells are adjusted. Finally, $RNN = \{o_2, o_9\}$.

### 3.3 Discussion

Notice the difference between the IGERN *incremental* step and the state-of-the-art RNN algorithm, CRNN [26], for *monochromatic* continuous reverse nearest neighbors. CRNN always monitors six different bounded regions, six candidate objects, and the query object while IGERN monitors only one single bounded region, always less than six candidate objects (experimental study shows an average of 3.3 monitored objects), and the query object. Furthermore, the size of the monitored region in IGERN is always less than those of CRNN as it is more likely to get open-ended regions in CRNN than IGERN. Thus, IGERN will always monitors an area that is about one sixth of the area monitored by CRNN. Also, IGERN always monitors about half of the objects monitored by CRNN. More details about the difference between the two algorithms will be discussed analytically in Section 6 and experimentally in Section 7. The *initial* step in IGERN is similar to the static approach TPL [22] with the difference that we embed new functionalities to produce a set of objects that will be monitored later in the *incremental* step.

## 4 Continuous Evaluation of Bichromatic Reverse Nearest Neighbors

Unlike the *monochromatic* case where all objects are of the same type, in *bichromatic* reverse nearest neighbors, we distinguish between two types of objects $A$ and $B$. For a query object of type $A$, the objective is to find data objects of type $B$ in which the query point is their nearest $A$ object. While the number of reverse nearest neighbors in the *monochromatic* reverse nearest neighbor case is limited to only six, in the *bichromaatic* case, there is no limit on the number of reverse nearest neighbors. Instead, it could be the case that for a query object $A$, all data objects of type $B$ are considered as its reverse nearest neighbors.

With these fundamental differences between the *monochromatic* and *bichromatic* reverse nearest neighbors, the IGERN algorithm still keeps the same flavor and the same framework to handle both the *monochromatic* and the *bichromatic* reverse nearest neighbor queries. This section presents the IGERN algorithm for *bichromatic* reverse nearest neighbor queries, which is basically an adaptation of the *monochromatic* IGERN algorithm to work for the *bichromatic* case. Thus, as in the *monochromatic* case, a grid data structure $G$ is maintained where each cell $c \in G$ keeps track of the moving objects within its boundaries. Also, similar to the *monochromatic* case, the *bichromatic* IGERN has two main steps, the *initial* step that is executed only once to report the first query answer and the *incremental* step that is triggered every $T$ time units to continuously maintain the query answer.

Throughout this section, Figure 3 gives a running example of 13 moving objects of two different types: six circle objects of type $A$, $o_{A1}$ to $o_{A6}$, six square objects of type $B$, $o_{B1}$ to $o_{B6}$, and a query object of type $A$, $q_A$. The objective is to find objects of type $B$ in which the query point $q_A$ of type $A$ is their nearest $A$ object. As a notation, objects $o_A$ and $o_B$ are of types $A$ and $B$, respectively, while sets $S_A$ and $S_B$ contain only data objects of types $A$ and $B$, respectively. The *initial* and *incremental* steps are described in Sections 4.1 and 4.2, respectively, while Section 4.3 gives a general discussion about the *bichromatic* IGERN algorithm.

### 4.1 Step 1: Getting the Initial Answer

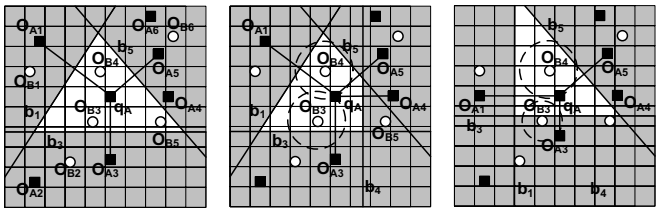
For a query object of type $A$ ($q_A$), the objectives of the *initial* step in the *bichromatic* IGERN algorithm are: (1) Obtaining a bounded region $r$ around $q_A$ to be monitored in the *incremental* step, (2) Identifying a set objects of type $A$ ($NN_A$) that need to be monitored later as their movement may trigger a change of answer, and (3) Identifying the set of initial reverse nearest neighbors of type $B$ ($RNN_B$) to $q_A$. Algorithm 3 gives the pseudo-code for the *initial* step. The input is the query object $q_A$ while the output is the two sets $RNN_B$ and $NN_A$. Initially, the set $NN_A$ is empty while all grid cells in $G$ are set to *alive*, i.e., all cells

**Algorithm 3** Pseudo code for the Bi Initial Step

```
1: Function INITBIIGERN(Query q_A)
2: NN_A ← ∅, Mark all grid cells G as alive
   {Phase I: Bounded Region}
3: while (o_A ← the nearest A object to q_A in alive cells) ≠ NULL do
4:     NN_A ← NN_A ∪ o_A, b ← The bisector of q_A and o_A
5:     Mark grid cells from b to the furthest boundaries from q_A as dead
6: end while
   {Phase II: Verification}
7: RNN_B ← ∅
8: for each object o_B ∈ the alive cells do
9:     o_A ← is the nearest object of type A to o_B
10:    if o_A = q_A then
11:        RNN_B ← RNN_B ∪ o_B
12:    else
13:        NN_A ← NN_A ∪ o_A, b ← The bisector of q_A and o_A
14:        Mark grid cells from b to the furthest boundaries of q_A as dead
15:        For any two objects o_A, o_A' ∈ NN_A, remove object o_A from
           NN_A only if dist(o_A, o_A') < dist(o_A, q_A)
16:    end if
17: end for
18: return RNN_B, NN_A
```

**Algorithm 4** Pseudo code for Bi Incremental Step

```
1: Function INCRBIIGERN(Query q_A, set NN_A)
2: if q_A or any objects ∈ NN_A have moved then
3:     Redraw the bisectors between q_A and all objects in NN_A
4:     Only the cells between q_A and the bisectors are marked as alive
5: end if
6: if there is any object o_B within the alive cells then
7:     Tighten the region as in Phase I of Algorithm 3 (Lines 3 to 6)
8:     For any two objects o_A, o_A' ∈ NN_A, remove object o_A from NN_A
       only if dist(o_A, o_A') < dist(o_A, q_A)
9: end if
10: Verify the answer as in Phase II of Algorithm 3 (Lines 7 to 17)
11: return RNN_B, NN_A
```



(a) Phase I (b) Phase II (c) Incremental

*Figure 3:* An example of the *bichromatic* IGERN.

have the potential of containing a reverse nearest neighbor of $q_A$. Then, the *initial* step has the following two phases:

**Phase I: Bounded Region.** This phase starts by finding object $o_A$ that is nearest to $q_A$ in the *alive* cells (Line 3 in Algorithm 3). Then, object $o_A$ is added to the list of $A$ objects ($NN_A$) that will be monitored later in the *incremental* step. Similar to the *monochromatic* case, the bisector $b$ between $q_A$ and $o_A$ is drawn while all the grid cells between $b$ and the space boundaries that are furthest from $q_A$ are marked as *dead* (Lines 4-5 in Algorithm 3). Such process continues until there are no more objects of type $A$ in any of the *alive* cells. In Figure 3a, the nearest neighbor search in the *alive* cells results in finding $o_{A5}$, $o_{A3}$, and $o_{A1}$, respectively, and the corresponding bisectors $b_5$, $b_3$, and $b_1$ are drawn until the *alive* cells (non-shaded cells) do not contain any square $A$ objects. Thus, $NN_A = \{o_{A1}, o_{A3}, o_{A5}\}$.

**Phase II: Verification.** This phase aims to find the current reverse nearest neighbors, tighten the monitored bounded region, and modify the list of objects that need to be monitored in the *incremental* step. The main idea is to go through every single object $o_B$ within the *alive* cells and check for its nearest $A$ object, $o_A$ (Line 9 in Algorithm 3). If it ends up that $o_A$ is the query object $q_A$, then $o_B$ is considered as a reverse nearest neighbor to $q_A$, thus added to the set $RNN_B$ (Line 11 in Algorithm 3). However, if $o_A$ is not $q_A$, then $o_A$ is considered as one of the objects to be monitored, a bisector is drawn between $q_A$ and $o_A$, the corresponding grid cells are marked as *dead*, and the set $NN_A$ is cleaned to make sure that it contains the minimal required objects that need to be monitored (Lines 13-15 in Algorithm 3). Figure 3b depicts such scenario where there are three $B$ objects in the *alive* cells, $o_{B3}$, $o_{B4}$, and $o_{B5}$. Upon testing for their nearest $A$ objects (the dotted circles in Figure 3b), it turns out that only $o_{B3}$, and $o_{B4}$ are reverse nearest neighbors while $o_{B5}$ has $o_{A4}$ as its nearest $A$ object. Thus, a bisector $b_4$ is drawn between $q_A$ and $o_{A4}$ and the corresponding cells are shaded. Object $o_{A4}$ is then removed from the monitored nearest neighbors $NN_A$ because it is closer to $o_{A5}$ than $q_A$. As a result, $NN_A = \{o_{A1}, o_{A3}, o_{A5}\}$ while $RNN_B = \{o_{B3}, o_{B4}\}$.

### 4.2 Step 2: Incremental Maintenance

The *incremental* step of IGERN is repetitively executed every $T$ time units. Algorithm 4 gives the pseudo-code of the *incremental* step where the input is the query object $q_A$ and the set of monitored objects $NN_A$ that came from either the initial step at time $T_0$ or a previous execution of the *incremental* step. The output is the current reverse nearest neighbors $RNN_B$ and a modified set of objects $NN_A$ to be monitored in the next execution instance of the *incremental* step. Similar to the *monochromatic* case of IGERN, the *incremental* step of the *bichromatic* case checks for three different scenarios: (1) The query object $q_A$ moves to a new location (Line 2 in Algorithm 4), (2) At least one of the objects in $NN_A$ moves to a new location (Line 2 in Algorithm 4), and (3) A new object of type $A$ moves into the *alive* cells (Line 6 in Algorithm 4). If none of these scenarios took place, then the *incremental* step will only *verify* the current sets ($RNN_B$ and $NN_A$) in a similar way to the *verification* phase in the *initial* step (Line 10 in Algorithm 4).

However, if it is the case that either the query object $q_A$ or one of the objects in $NN_A$ have moved, then new bisec-

tors will be drawn from $q_A$ to the objects in $NN_A$. Also, only the cells between $q_A$ and the new bisectors are considered *alive* (Lines 3-4 in Algorithm 4). Then, the *incremental* step checks if a new object of type $A$ is found in an *alive* cell. Such a check also accommodates the new bounded region formed by the movement of any of the monitored objects. If there are no objects in any of the *alive* cells, then only the verification step is needed (Line 10 in Algorithm 4). However, if there is one or more objects of type $A$ in the *alive* cells, we will tighten the *alive* cells in a similar way to the first phase in the *initial* step (Line 7 in Algorithm 4). Then, the list $NN_A$ is cleaned by removing any object that is not participating in drawing the bisectors (Line 8 in Algorithm 4). Finally, the sets $NN_A$ and $RNN_B$ are verified (Line 10 in Algorithm 4).

Figure 3c depicts the case where two of the monitored objects have moved, namely, $o_{A1}$ and $o_{A3}$. Thus, new bisectors will be drawn. Then, it turns out that object $o_{B3}$ has $o_{A3}$ as its nearest $A$ object. Thus, $o_{B3}$ is not a reverse nearest neighbor to $q_A$ any more. The returned answer from the *incremental* step will be $NN_A = \{o_{A1}, o_{A3}, o_{A5}\}$ while $RNN_B = \{o_{B4}\}$.

### 4.3 Discussion

IGERN is the first algorithm that addresses the continuous *bichromatic* reverse nearest neighbor queries. One of the main attractive features of the *bichromatic* IGERN is that it is based on the *monochromatic* case, i.e., IGERN provides a unified framework for continuous evaluation of both *monochromatic* and *bichromatic* reverse nearest neighbor queries. In this sense, IGERN is more attractive to industry and system-oriented research prototypes (e.g., [6, 16, 25]) as one framework would be enough for different cases. This is in contrast to previous approaches for reverse nearest neighbor queries that can solve only the *monochromatic* case without any direct extension to the *bichromatic* case. For example, *monochromatic* RNN algorithms that rely on the fact that there could be only six reverse nearest neighbors (e.g., [21, 26]) cannot be extended to the *bichromatic* case where the number of reverse nearest neighbors could be much greater than six. The *initial* step of the *bichromatic* IGERN is similar to getting only the Voronoi cell around the query object [2], however, we embed several functionalities inside the algorithm for finding the Voronoi cell in order to exactly maintain a minimal set of objects and a bounded region that will be monitored in later executions of the *incremental* step.

## 5 Proof of Correctness

In this section, we present the proof of correctness for both the *monochromatic* and *bichromatic* IGERN algorithms by proving that: (1) IGERN is accurate, i.e., an object $p$ returned by IGERN is an exact RNN, and (2) IGERN is complete, i.e., IGERN returns all possible RNNs.

### 5.1 Monochromatic: Accurate and Complete

**Theorem 1** *For any query $q_T$, executed at time $T$, an object $o$ returned by the monochromatic IGERN algorithm is an exact reverse nearest neighbor to $q$.*

**Proof:** Assume that for the query $q_T$, IGERN returns an object $o$ which is not a reverse nearest neighbor to $q_T$. Then, there must be another object $o'$ where $dist(o, o') < dist(o, q)$. However, both the *initial* and *incremental* steps of IGERN are concluded by a verification phase which guarantees that $q$ is the nearest object to any returned object $o$ (Line 7 in Algorithm 1 and Line 10 in Algorithm 2). Thus, object $o'$ cannot exist either in the *initial* or the *incremental* step. Thus, $q$ is the nearest object to $o$, and hence, the object $o$ returned by IGERN is an exact reverse nearest neighbor to $q$. □

**Theorem 2** *For any query $q_T$, executed at time $T$, the monochromatic IGERN algorithm returns ALL reverse nearest neighbors to $q_T$.*

**Proof:** Assume that for the query $q_T$, IGERN did not return an object $o$ that is a reverse nearest neighbor to $q_T$, i.e., $q_T$ is the nearest object to $o$. Then, there are exactly two cases:

**Case 1:** $o$ is in an *alive* cell. Phase I in the *initial step* of IGERN continues to iterate till there are no objects located in the *alive* cells (Lines 2-6 in Algorithm 1). Similarly, the *incremental* step makes sure that there are no objects in the *alive* cells (Line 7 in Algorithm 2). Thus, object $o$ cannot be located in any of the *alive* cells.

**Case 2:** $o$ is in a *dead* cell. In the *initial* step, each bisector $b_j$ between $o_j$ and $q$ divides the space into *dead* and *alive* cells such that any object $o$ (other than $o_j$) in the dead cells has $dist(o, o_j) < dist(o, q)$. Thus, $q$ cannot be nearest neighbor to $o$. If $o_j$ is an RNN to $q$, $o_j$ will be returned in the *verification* phase. Thus, object $o$ cannot be in a *dead* cell. Similar argument holds for the *incremental* step.

From Cases 1 and 2, object $o$ cannot exist. Thus, IGERN produces all reverse nearest neighbors to $q$. □

### 5.2 Bichromatic: Accurate and Complete

**Theorem 3** *For any query $q_{AT}$, of type $A$, executed at time $T$, an object $o_B$ returned by the bichromatic IGERN algorithm is an exact reverse nearest neighbor of type $B$ to $q_A$.*

**Proof:** Assume that for the query $q_{AT}$, the *bichromatic* IGERN returns an object $o_B$ which is not a reverse nearest neighbor to $q_A$. Then, there must be another object $o'_A$ where $dist(o_B, o'_A) < dist(o_B, q_A)$. However, both the *initial* and *incremental* steps of the *bichromatic* IGERN concludes by a verification step which verifies that every returned object $o_B$ has to have $q_A$ as its nearest $A$ object. Thus, object $o'_A$ cannot exist, and hence, the object $o_B$ returned by the *bichromatic* IGERN is an exact reverse nearest neighbor to $q_A$. □

**Theorem 4** *For any query $q_{AT}$, of type $A$, executed at time $T$, the bichromatic IGERN algorithm returns ALL reverse nearest neighbors to $q_{AT}$.*

**Proof:** Assume that for the query $q_{AT}$, the *bichromatic* IGERN did not return an object $o_B$ that is a reverse nearest neighbor to $q_A$, i.e., $q_A$ is the nearest $A$ object to $o_B$. Similar to the proof of Theorem 3, object $o_B$ is either in an *alive* or a *dead* cell in which both cases cannot take place in Algorithms 3 and 4. Thus, we conclude that object $o_B$ cannot exist and the *bichromatic* IGERN produces all the reverse nearest neighbors to $q_A$. □

## 6 Analytical Comparison of RNN Algorithms

This section presents the cost model for three *monochromatic* and two *bichromatic* algorithms. Namely for the *monochromatic* case, we present the cost model for IGERN, CRNN [26], and repetitive evaluation of the static TPL algorithm [22]. For the *bichromatic* case, we present the cost model for IGERN and the repetitive computations of the static creation of Voronoi cells [2]. Finally, we compare the cost model of IGERN with its counterparts for both the *monochromatic* and *bichromatic* cases.

**Monochromatic IGERN cost.** Let $mI$ be the cost function for the *monochromatic* IGERN algorithm. Then, for a query $q$ that is executed for $T$ time units, $mI(q) = mI_{init}(q_0)+\Sigma_{t=1}^{T} mI_{incr}(q_t)$ where $mI_{init}$ and $mI_{incr}$ are the cost of the *initial* and *incremental* steps, respectively, while $q_t$ is the execution of the query $q$ at time $t$. Thus, $mI(q) = r_0(NN_c(q_0) + NN(q_0)) + \Sigma_{t=1}^{T}(NN_b(q_t) + r_t NN(q_t))$ where $r_t$ is the number of objects that are candidates to be RNNs in time step $t$, ($r_t \leq 6$). $NN_c(q_t)$, $NN(q_t)$, and $NN_b(q_t)$ represent the cost for the *constrained*, *unconstrained*, and *bounded* nearest neighbor algorithms to $q_t$, respectively. The *constrained* NN is done only within the remaining *alive* cells (Line 3 in Algorithm 1) while the *unconstrained* NN is performed in the whole space (Line 7 in Algorithm 1) and the *bounded* NN is performed only within a bounded region of the space (Line 7 in Algorithm 2).

**CRNN cost.** Let $C$ be the cost function for CRNN. Then, $C(q) = 6(NN_c(q_0)+NN(q_0))+ \Sigma_{t=1}^{T} 6(NN_b(q_t)+ NN(q_t))$ where $NN_c(q_i)$, $NN(q_i)$, and $NN_b(q_i)$ are similar to those of the IGERN algorithm. Notice that CRNN always monitors six regions and six RNN candidates regardless of the data distribution. In addition, the bounded NN search $NN_b$ is consistently repeated six times.

**TPL cost.** Let $L$ be the cost function for a repetitive evaluation of the static TPL approach. Then, $L(q) = \Sigma_{t=0}^{T} r_t(NN_c(q_t) + NN(q_t))$. As there is no incremental evaluation in TPL, all execution instances perform a constrained nearest neighbor search followed by an unconstrained one for verification.

**IGERN vs. CRNN.** Based on our cost model, the cost ratio between the *monochromatic* IGERN and CRNN is:
$\frac{mI(q)}{C(q)} = \frac{r_0(NN_c(q_0)+NN(q_0))+ \Sigma_{t=1}^{T}(NN_b(q_t)+r_t NN(q_t))}{6(NN_c(q_0)+NN(q_0))+ \Sigma_{t=1}^{T} 6(NN_b(q_t)+NN(q_t))}$
Thus, for any single time instance $T$, the ratio is $\frac{r_0}{6}$ if $T = 0$ and $\frac{NN_b(q_t)+r_t NN(q_t)}{6NN_b(q_t)+6NN(q_t)}$ for $T > 0$. Notice that for each time instance $T > 0$, the bounded nearest neighbor search is done only once in IGERN as opposed to six times in CRNN. Also, the unconstrained nearest neighbor search is performed only $r_t$ times in IGERN rather than exactly six times in CRNN. Since $r_t \leq 6$, IGERN always achieves better performance than CRNN.

**IGERN vs. TPL.** The cost ratio between the *monochromatic* IGERN and the reevaluation of the static TPL is:
$\frac{mI(q)}{L(q)} = \frac{r_0(NN_c(q_0)+NN(q_0))+ \Sigma_{t=1}^{T}(NN_b(q_t)+r_t NN(q_t))}{\Sigma_{t=0}^{T} r_t(NN_c(q_t)+NN(q_t))}$
Thus, for any single time instance $T$, the ratio is one if $T = 0$ while the ratio is $\frac{NN_b(q_t)+r_t NN(q_t)}{r_t NN_c(q_t)+r_t NN(q_t)}$ for $T > 0$. Notice that the bounded nearest neighbor search in IGERN is much less expensive than the constrained one in TPL as the bounded case searches only within a small bounded region. In addition, the bounded search in IGERN is done only once while the constrained search in TPL is done $r_t$ times at each time instance. Thus, IGERN always achieves better performance than the repetitive evaluation of TPL.

**Bichromatic IGERN cost.** Let *bI* be the cost function for the *bichromatic* IGERN algorithm. Then, for a query of type $A$ ($q_A$) that is executed for $T$ time units, $bI(q_A) = bI_{init}(q_{A0}) + \Sigma_{t=1}^{T} bI_{incr}(q_{A_t})$ where $bI_{init}$ and $bI_{incr}$ are the cost of the *initial* and *incremental* steps, respectively, while $q_{A_t}$ is the execution of the query $q_A$ at time $t$. Thus, $bI(q_A) = a_0 NN_c(q_{A0}) + b_0 NN(q_{A0}) + \Sigma_{t=1}^{T}(NN_b(q_{A_t}) + b_t NN(q_{A_t}))$ where $a_t$ and $b_t$ are the number of $A$ objects that need to be monitored and the number of $B$ objects in the monitored bounded region, at time step $t$, respectively. $NN_c(q_t)$, $NN_b(q_t)$, and $NN(q_t)$ are similar to those of the *monochromatic* IGERN algorithm. Notice that the constrained NN is done only once in the *initial* step while the bounded NN is done only once at each execution of the *incremental* step.

**Voronoi cost.** Let $V$ be the cost function for the continuous re-creation of a Voronoi cell around $q_A$. Then, $V(q_A) = \Sigma_{t=0}^{T} a_t NN_c(q_t) + b_t NN(q_t)$. The main idea is that creating a new Voronoi cell is repeated at each time step $t$.

**IGERN vs. Voronoi cell.** The cost ratio between the *bichromatic* IGERN algorithm and repetitive creation of Voronoi cells is:
$\frac{bI(q)}{V(q)} = \frac{a_0 NN_c(q_0)+b_0 NN(q_0)+ \Sigma_{t=1}^{T}(NN_b(q_t)+b_t NN(q_t))}{\Sigma_{t=0}^{T}(a_t NN_c(q_t)+b_t NN(q_t))}$
Thus for any single time instance $T$, the ratio is one if $T = 0$ and $\frac{(NN_b(q_t)+b_t NN(q_t))}{(a_t NN_c(q_t)+b_t NN(q_t))}$ for $T > 0$. Notice that the bounded search in IGERN is much cheaper than the constrained search in the Voronoi cell. Furthermore, the bounded search in IGERN is done only one time at each

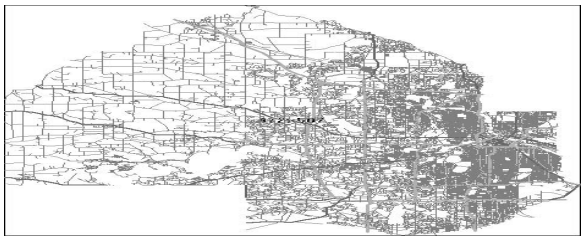
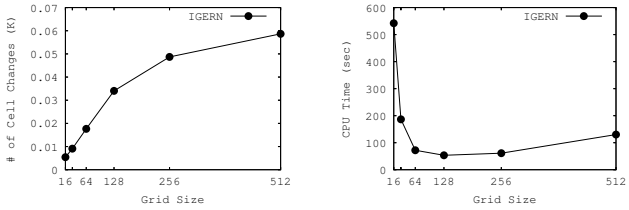
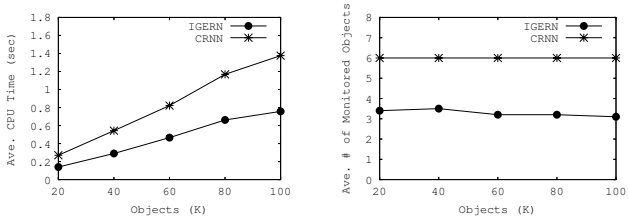


*Figure 4:* Map of Hennepin County, MN, USA.




(a) Number of Cell Changes (b) CPU Time

*Figure 5:* Grid Size Costs




(a) Processing Time (b) Monitored Objects

*Figure 6:* Monochromatic Costs

time instance while the constrained search is done $a_t$ times in the Voronoi cell construction. Thus, the *bichromatic* IGERN achieves much better performance than repetitive construction of Voronoi cells.

## 7 Experimental Results

In this section, we experimentally evaluate the performance of IGERN for both the *monochromatic* (Section 7.2) and *bichromatic* (Section 7.3) reverse nearest neighbors. In the *monochromatic* case, we compare IGERN against the state-of-the-art algorithm for continuous *monochromatic* reverse nearest neighbor queries CRNN [26]. However, for the *bichromatic* case, IGERN is the first algorithm for continuous *bichromatic* reverse nearest neighbor queries. Thus, IGERN is compared against a static approach of repetitive computation of Voronoi cells [2]. To ensure consistency and fairness among different approaches, we use the algorithm in [31] as the underlying nearest neighbor search for all approaches of reverse nearest neighbor queries.

We use the *Network-Based Generator of Moving Objects* [4] to generate a set of moving objects and moving queries. The input to this generator is the road map of Hennepin County in Minneapolis (Figure 4). The output of the generator is a set of moving objects that move on the road network of the given city. Unless mentioned otherwise, we generate 100K moving objects and a duration of 100 time units. All experiments were performed on an Intel Pentium IV CPU 2.0GHz with 512MB RAM.

### 7.1 Grid Size

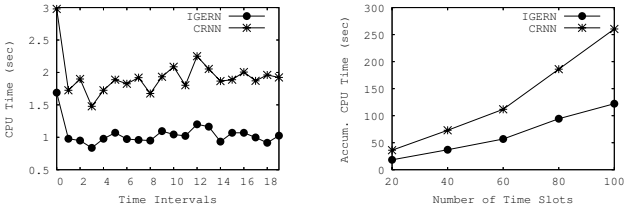
Figure 5 gives the effect of increasing the grid size from 16×16 to 512×512. Although this experiment is performed for the *monochromatic* IGERN, similar performance came out from the *bichromatic* case. Figure 5a gives the number of cell changes for all objects as the grid size increases. The number of cell changes is an indicator of the overhead needed to maintain the grid structure. As more grid cells are maintained, more updates need to take place. On the other side, Figure 5b gives the CPU time when executing the IGERN algorithm for different grid sizes. For small grid sizes, IGERN encounters higher costs as each grid cell contains large number of objects, hence, the underlying nearest neighbor search would search within a lot of unnecessary objects. The performance becomes better when the grid size increases to 64-128 as the nearest neighbor search is performed within limited number of objects. However, more increase in the grid size badly affect IGERN. This is mainly because the cost of data updates for objects that change their cells take place as has been depicted in Figure 5a. As a compromise, in the rest of our experiments, we use a grid structure of size 64.

### 7.2 Monochromatic RNNs

This section compares the *monochromatic* IGERN algorithm to CRNN in terms of scalability and stability.

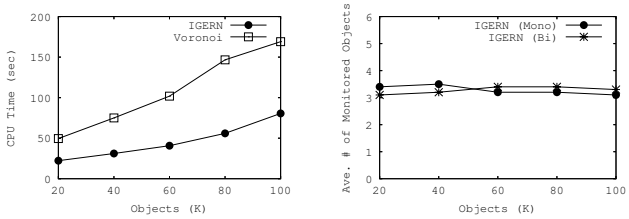
#### 7.2.1 Scalability

Figure 6a gives the effect of increasing the number of moving objects from 20K to 100K on both IGERN and CRNN. The average CPU time is taken over 100 time units. The IGERN consistently performs better than CRNN. As the number of objects increases, IGERN takes advantage of the fact that it monitors only one region and less number of objects. On the other side, CRNN consistently monitors six regions and six moving objects. Furthermore, the single area monitored by IGERN is bounded while some of the areas of CRNN may be open-ended based on the data distribution. As a result, the average CPU cost of IGERN is much less than that of CRNN. Figure 6b gives the average number of monitored objects in both IGERN and CRNN for




(a) Time Intervals (b) Time Slots

*Figure 7:* Monochromatic Time Slot Cost




(a) Processing Time (b) Monitored Objects

*Figure 8:* Bichromatic Costs

different data sizes. While CRNN exactly monitors six objects, IGERN, on average, monitors about 3.3 regions. This experiments supports the argument of Figure 6a in which IGERN has much less CPU time than CRNN. In general, the experiments in this section supports the analytical comparison of IGERN and CRNN as depicted in Section 6.

### 7.2.2 Stability

Figure 7a gives the execution of both IGERN and CRNN for the first 20 time units. As both algorithms are designed for *continuous* evaluation, the first time evaluation at time $T = 0$, is more expensive than later time instances that only maintains the current answer. However, IGERN consistently gives better performance than CRNN. The stable performance of IGERN for all time units $T > 0$ indicates that the performance of the *incremental* step does not deteriorate over time. Also, we got similar performance when running the same experiment for longer time periods.

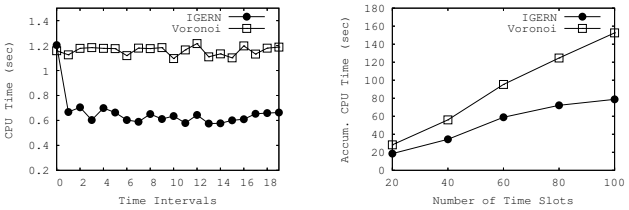
Figure 7b gives the total accumulated time spent on one execution of the *initial* step followed by a sequence of executions of the *incremental* step for both IGERN and CRNN for up to 100 time units. It is clear that when the query runs for longer time periods, the amount of saving we achieve with IGERN becomes much greater. Thus, IGERN consistently gives a stable increase of performance over CRNN.

## 7.3 Bichromatic

This section compares the *bichromatic* IGERN algorithm to a repetitive computation of static Voronoi cells in terms of scalability and stability.

### 7.3.1 Scalability

Figure 8a gives the effect of increasing the number of moving objects from 20K to 100K on both IGERN and the creation of a Voronoi cell. The IGERN algorithm consistently gives better performance than Voronoi as IGERN only needs to maintain the answer. The increase in CPU time of IGERN with the increase of the number of objects is much less than that of Voronoi. Thus, IGERN can scale well to a large number of moving objects. Basically, IGERN




(a) Time Intervals (b) Time Slots

*Figure 9:* Bichromatic Time Slot Cost

takes advantage of the fact that it monitors only one region and few number of objects rather than reconstructing the answer at each time instance. This experiment supports the analytical comparison of IGERN and Voronoi as depicted in Section 6.

Figure 8b gives the number of monitored objects over the course of execution for both the *monochromatic* and *bichromatic* reverse nearest neighbor queries. Basically, the conclusion from this figure is that IGERN almost has a similar performance for both *monochromatic* and *bichromatic* cases. Thus, even the case of *bichromatic* reverse nearest neighbors is more complicated, however, IGERN can still achieve similar performance for both cases.

### 7.3.2 Stability

Figure 9a gives the execution of both IGERN and repetitive Voronoi for the first 20 time units. Only at the first time instance ($T = 0$), Voronoi gives better performance. This is basically, because IGERN use a modified version of Voronoi cells to be able to reduce the amount of work that will be needed later in the *incremental* step. Thus, for all time instances $T > 0$, IGERN consistently gives much higher performance than Voronoi. Also, the performance of IGERN is stable over time as we got similar performance when running the same experiment for longer time periods. Figure 9b gives the total accumulated time spent on one execution of the initial step followed by a sequence of exe-

cutions of the *incremental* step for IGERN with the repetitive construction of Voronoi cells. It is clear that as time continues, IGERN saves a great amount of CPU time as its incremental approach avoids the intensive computations of Voronoi cells.

## 8 Conclusions

In this paper, we have presented a novel algorithm for Incremental and General Evaluation of continuous Reverse Nearest neighbor queries (IGERN, for short). IGERN is *general* as it is a unified framework for both *monochromatic* and *bichromatic* reverse nearest neighbors. In addition, IGERN is *incremental* as it limits the attention to only a single bounded region and a few set of moving objects rather than focusing on the whole space. IGERN clearly enhances over the state-of-the-art algorithms in continuous *monchromatic* reverse neighbor queries. Furthermore, IGERN is the first algorithm that deals with continuous *bichromatic* reverse nearest neighbors. The correctness of both the *monochromatic* and *bichromatic* IGERN is proved by showing that IGERN is accurate, i.e., any object returned by IGERN is a reverse nearest neighbor, and is complete, i.e., IGERN returns all reverse nearest neighbors. Analytical comparison of IGERN with previous approaches for reverse nearest neighbors is provided. Experimental evidence that supports the analytical comparison is given where IGERN outperforms previous approaches in both the *monochromatic* and *bichromatic* reverse nearest neighbors.

## References


[1] E. Achtert, C. Bahm, P. Krager, P. Kunath, A. Pryakhin, and M. Renz. Efficient Reverse k-Nearest Neighbor Search in Arbitrary Metric Spaces. In *SIGMOD*, 2006.

[2] F. Aurenhammer. Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[3] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis. Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects. In *IDEAS*, 2002.

[4] T. Brinkhoff. A Framework for Generating Network-Based Moving Objects. *GeoInformatica*, 6(2):153–180, 2002.

[5] B. Gedik and L. Liu. MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System. In *EDBT*, 2004.

[6] R. H. Güting, V. T. de Almeida, D. Ansorge, T. Behr, Z. Ding, T. Hose, F. Hoffmann, M. Spiekermann, and U. Telle. SECONDO: An Extensible DBMS Platform for Research Prototyping and Teaching. In *ICDE*, 2005.

[7] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras. On-Line Discovery of Dense Areas in Spatio-temporal Databases. In *SSTD*, 2003.

[8] H. Hu, J. Xu, and D. L. Lee. A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects. In *SIGMOD*, 2005.

[9] G. S. Iwerks, H. Samet, and K. Smith. Continuous K-Nearest Neighbor Queries for Continuously Moving Points with Updates. In *VLDB*, 2003.

[10] C. S. Jensen. *Location-Based Services*, chapter Database Aspects of Location-Based Services, pages 115–148. Morgan Kaufmann, 2004.

[11] C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang. Effective Density Queries on ContinuouslyMoving Objects. In *ICDE*, 2006.

[12] F. Korn and S. Muthukrishnan. Influence Sets Based on Reverse Nearest Neighbor Queries. In *SIGMOD*, 2000.

[13] I. Lazaridis, K. Porkaew, and S. Mehrotra. Dynamic Queries over Mobile Objects. In *EDBT*, 2002.

[14] M. F. Mokbel and W. G. Aref. PLACE: A Scalable Location-aware Database Server for Spatio-temporal Data Streams. *IEEE Data Engineering Bulletin*, 28(3):3–10, 2005.

[15] M. F. Mokbel, X. Xiong, and W. G. Aref. SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. In *SIGMOD*, 2004.

[16] M. F. Mokbel, X. Xiong, W. G. Aref, S. Hambrusch, S. Prabhakar, and M. Hammad. PLACE: A Query Processor for Handling Real-time Spatio-temporal Data Streams (Demo). In *VLDB*, 2004.

[17] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou. Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring. In *SIGMOD*, 2005.

[18] A. Nanopoulos, Y. Theodoridis, and Y. Manolopoulos. C2P: Clustering based on Closest Pairs. In *VLDB*, 2001.

[19] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch. Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. *IEEE Transactions on Computers*, 51(10):1124–1140, 2002.

[20] A. Singh, H. Ferhatosmanoglu, and A. S. Tosun. High Dimensional Reverse Nearest Neighbor Queries. In *CIKM*, 2003.

[21] I. Stanoi, D. Agrawal, and A. ElAbbadi. Reverse Nearest Neighbor Queries for Dynamic Databases. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2000.

[22] Y. Tao, D. Papadias, and X. Lian. Reverse kNN Search in Arbitrary Dimensionality. In *VLDB*, 2004.

[23] Y. Tao, D. Papadias, and Q. Shen. Continuous Nearest Neighbor Search. In *VLDB*, 2002.

[24] Y. Tao, M. L. Yiu, and N. Mamoulis. Reverse Nearest Neighbor Search in Metric Spaces. *TKDE*, 18(8), 2006.

[25] O. Wolfson, A. P. Sistla, B. Xu, J. Zhou, and S. Chamberlain. DOMINO: Databases fOr MovINg Objects tracking (Demo). In *SIGMOD*, 1999.

[26] T. Xia and D. Zhang. Continuous Reverse Nearest Neighbor Monitoring. In *ICDE*, 2006.

[27] X. Xiong, M. F. Mokbel, and W. G. Aref. SEA-CNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. In *ICDE*, 2005.

[28] C. Yang and K.-I. Lin. An Index Structure for Efficient Reverse Nearest Neighbor Queries. In *ICDE*, 2001.

[29] M. L. Yiu and N. Mamoulis. Reverse Nearest Neighbors Search in Ad-hoc Subspaces. In *ICDE*, 2006.

[30] M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao. Reverse Nearest Neighbors in Large Graphs. *TKDE*, 18(4):540–553, 2006.

[31] X. Yu, K. Q. Pu, and N. Koudas. Monitoring K-Nearest Neighbor Queries Over Moving Objects. In *ICDE*, 2005.