This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Incremental and General Evaluation of Reverse Nearest Neighbors

James M. Kang, *Student Member, IEEE,* Mohamad F. Mokbel, *Member, IEEE,*
Shashi Shekhar, *Fellow, IEEE,* Tian Xia, Donghui Zhang

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 in that it is applicable for both continuous monochromatic and bichromatic reverse nearest neighbor queries. This problem is faced in a number of applications such as enhanced 911 services and in army strategic planning. A main challenge in these problems is to maintain the most up to date query answers as the dataset frequently changes over time. Previous algorithms for monochromatic continuous reverse nearest neighbor queries rely mainly on monitoring at the worst case of six pie regions, whereas 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. We also propose a new optimization for the monochromatic IGERN to reduce the number of nearest neighbor searches. Furthermore, a filter and refine approach for IGERN (FR-IGERN) is proposed for the continuous evaluation of bichromatic reverse nearest neighbor queries which is an optimized version of our previous approach. The computational complexity of IGERN and FR-IGERN is presented in comparison to the state-of-the-art algorithms in the monochromatic and bichromatic cases. In addition, the correctness of IGERN and FR-IGERN in both the monochromatic and bichromatic cases respectively are proved. Extensive experimental analysis using synthetic and real datasets shows that IGERN and FR-IGERN is efficient, is scalable, and outperforms previous techniques for continuous reverse nearest neighbor queries.

Index Terms

Continuous Queries, Query Optimization, and Reverse Nearest Neighbor

I. Introduction

THE past decade has seen the assimilation of sensor networks and location-based systems in real world applications such as enhanced 911 services, army strategic planning, retail services, and mixed-reality games. The continuous[1] movement of data objects within these applications calls for new query processing techniques that scale up with the high rates of location updates. While numerous works have addressed continuous range queries (e.g., see [1], [2], [4], [6], [7]) and continuous nearest neighbor queries (e.g., see [3], [5], [8], [9], [11]), there is still a lack of research in addressing the continuous reverse nearest neighbor queries.

There are two types continuous evaluation of Reverse Nearest Neighbor (RNN) queries, namely, *monochromatic* RNN and *bichromatic* RNN [12]. In the *monochromatic* RNN, all moving data and query objects are of the same type and their locations are reported at every time interval t. 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 $dist(o, o') < dist(o, q)$ and the answer is updated at every t. 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 his own reverse nearest neighbors to avoid being shot. In the *bichromatic* RNN, all moving objects and queries are one of two distinct types, A and B, and their locations are reported at every t. 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 $dist(o_B, o'_A) < dist(o_B, q_A)$ and the answer is updated at every t. Applications

J.M. Kang, M.F. Mokbel, and S. Shekhar is with the Department of Computer Science and Engineering, University of Minnesota-Twin Cities, 200 Union Street SE, Minneapolis, MN 55455. E-mail: {jkang,mokbel,shekhar}@cs.umn.edu.

T. Xia and D. Zhang is with the College of Computer and Information Science, Northeastern University, 360 Huntington Ave., Boston, MA 02115. E-mail: {tianxia,donghui}@ccs.neu.edu.

[1]The term 'continuous' may be interpreted either in the calculus sense or to mean an approximation. In this paper, we use 'continuous' in the latter sense where the approximation is based on a sampling rate of some middleware system that captures the data. Many related studies use 'continuous' in this sense of approximation (e.g., [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11]).

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:59 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

of the continuous *bichromatic* RNN include army strategic planning where a medical unit A in the battlefield is always in search of wounded soldiers of type B and A is their nearest medical unit. The continuous evaluation of RNN queries is also crucial in data mining applications where the RNNs of a query point q are those objects on which q has significant influence [12], [13].

Most of the previous work on reverse nearest neighbor queries focuses on snapshot queries in static environments, i.e., the continuous movement of both the query and data objects is not taken into account (e.g., see [14], [15], [12], [16], [17], [18], [19], [20]). To our knowledge, the CRNN algorithm [10] is the only attempt to evaluate *continuous* RNN queries. However, CRNN is applicable only to *monochromatic* queries, and 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, each of which is monitored independently for potential reverse nearest neighbors. This idea has been widely employed for most of the snapshot RNN algorithms (e.g., see [16]) as it is based on the theoretical foundation that there can be up to six answers for any *monochromatic* RNN query [16].

In this paper, we present a novel algorithm for Incremental and General Evaluation of continuous Reverse Nearest neighbor queries (IGERN, for short) and a filter and refine approach for IGERN (FR-IGERN) for both *monochromatic* and *bichromatic* RNN queries respectively. The IGERN algorithm goes beyond the idea of six pies in evaluating *monochromatic* RNN queries. Furthermore, the FR-IGERN algorithm provides an improved continuous evaluation of *bichromatic* RNN queries using filter and refine concepts that outperforms our previous method [21]. The main idea of both the IGERN and FR-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 filtering in FR-IGERN determines objects that are guaranteed to be the reverse nearest neighbors once the region r is found and the remaining objects are refined for any additional answers. The incremental aspect of both approaches comes from the fact that each execution instance of IGERN and FR-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, our contributions can be summarized as follows:

1) We present an IGERN algorithm for *monochromatic* continuous RNN queries that goes beyond the traditional method of dividing the space into six pie regions.
2) We propose an optimization to the *monochromatic* IGERN technique that reduces the number of nearest neighbor searches to find the reverse nearest neigbors.
3) We propose a Filter and Refine approach for IGERN (FR-IGERN) for the continuous evaluation of *bichromatic* RNNs that performs more efficiently than our previous method [21].
4) We prove the correctness of IGERN and FR-IGERN by proving their: (a) accuracy, i.e., a returned result by IGERN is an exact RNN, and (b) completeness, i.e., IGERN returns all possible RNNs.
5) We give analytical evidence that IGERN and FR-IGERN outperform previous approaches for both *monochromatic* and *bichromatic* RNNs.
6) We present experimental evaluation of both IGERN and FR-IGERN using both synthetic and real datasets.

The rest of this paper is organized as follows. Section II highlights the related work. The *monochromatic* IGERN and *bichromatic* FR-IGERN algorithms are described in Sections III and IV, respectively. We present the proof of correctness for IGERN in Section V. Section VI gives an analytical analysis of IGERN. Experimental evidence that the IGERN algorithm outperforms previous algorithms is given in Section VII. Finally, Section VIII concludes the paper.

II. Related Work

There is a recent interest in developing new continuous query processors to cope with the recent advances in dynamic location-aware environments [22], [23]. As a result, new algorithms have been developed for various types of continuous location-based queries, e.g., continuous range queries [1], [2], [4], [6], [7], continuous nearest neighbor queries [3], [5], [8], [9], [11], and continuous aggregates [24], [25]. Although reverse nearest neighbor queries are of the same importance as these query types, little work has been done to develop efficient algorithms for continuous reverse nearest neighbor queries.

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Various algorithms have been proposed for snapshot reverse nearest neighbors (RNNs) in different environments, e.g., in euclidian space [12], [16], [17], [26], metric space [14], [18], high-dimensional space [27], ad-hoc space [19], and large graphs [20]. In this paper, we mainly focus on euclidian space in which it is proved that there are at most six reverse nearest neighbors for the *monochromatic* case [16]. Utilizing this property, an 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 [17], relies mainly on recursively filtering the data by finding perpendicular bisectors between the query point and its nearest object. TPL introduces a concept for the perpendicular bisector between an object o and a query q where no objects in the half-plane containing o can be closer to q than o. TPL uses an R-tree and then identifies minimum bounding rectangles (MBR) that can be pruned based on the perpendicular bisectors around the query object. However, TPL only handles snapshot queries and becomes significantly expensive for continuous problems. In this paper, we compare our proposed approaches against TPL analytically to show this difference. Briefly, we use a grid index for TPL instead of an R-tree to account for continuously moving objects. As in the R-tree for TPL, where portions of an MBR are trimmed based on the perpendicular bisector, the grid cells where the bisectors intersect and bordering the query object may be half "dead" (i.e., objects in the half-plane not containing the query object) and half "alive" (i.e., objects in the half-plane containing the query object). Further details of this grid cell structure will be presented in Section III.

A recent technique for finding *monochromatic* reverse nearest neighbors for moving objects [28] is similar to our problem except that the velocity of each object is given as part of the input and each object is assumed to move on a plane which can then be indexed using a TPR-tree. However, in our proposed methods, we do not assume a specific velocity and objects can move in any direction which is not constrained to a single direction.

To our knowledge, there is only one algorithm that does not assume objects move on a single plane and a velocity is not given, termed CRNN [10], for continuous evaluation of reverse nearest neighbor queries. CRNN extends the idea of dividing the space into six pies, originally developed for snapshot queries [16], 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 *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. These drawbacks arise from the fact that CRNN ignores the relationship 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 CRNN, which monitors six regions and six objects.

III. 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 at each time unit for all time intervals T 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 III-A and III-B, respectively, while Section III-D gives a general discussion of IGERN.

A. 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 and the Grid index G 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:

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Algorithm 1 Pseudo code for the Mono Initial Step

```
1: Function initMonoIGERN(Query $q$, Grid index $G$)
2: RNNcand $\leftarrow \emptyset$, Mark all grid cells $\in G$ as alive
   {Phase I: Bounded Region}
3: while ($o_j \leftarrow$ the nearest object to $q$ in the alive cells) $\neq$ NULL do
4:     RNNcand $\leftarrow$ RNNcand $\cup o_j$, $b_j \leftarrow$ The $\perp$ bisector of $q$ and $o_j$
5:     Mark grid cells in the half-plane bounded by $b_j$ and containing $o_j$ as dead
6: end while
   {Phase II: Verification}
7: RNN $\leftarrow$ 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 $RNNcand$, Grid Index G)
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 $o_i, o_j \in$ RNNcand, remove object $o_i$ from RNNcand only if $dist(o_i, o_j) < dist(o_i, q)$
9: end if
10: RNN $\leftarrow$ RNNcand - {Objects that do not have $q$ as nearest object} // Verification Step
11: return RNN, RNNcand
```

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 in the half-plane bounded by b_j and containing o_j 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. Cells having a perpendicular bisector intersecting them and a border around the query object are still considered "alive" while only objects in the half-plane containing the query object are considered to be "alive". 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*. This 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). Since there are no more objects within the *alive* cells, the candidate set becomes *RNNcand* $= \{o_2, o_4, o_6\}$.

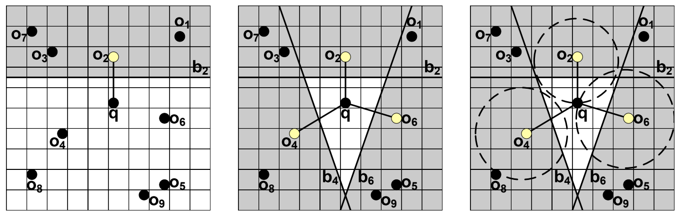
Phase II: Verification. In this phase, we go through all objects in *RNNcand* and only those objects that q is their nearest are considered as RNNs (Line 7 in Algorithm 1). In Figure 1c, the dotted circles indicate the nearest neighbor search for each object in *RNNcand*. Thus, *RNN* $= \{o_2, o_6\}$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

B. Step 2: Incremental Maintenance

The *incremental* step of IGERN is repetitively executed at each time unite for all time intervals T. Algorithm 2 gives the pseudo-code of the *incremental* step. The input to this algorithm is the query object q, the set *RNNcand* that came from either the *initial* step at time T_0 or a previous execution of the *incremental* step, and the Grid index G. 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.

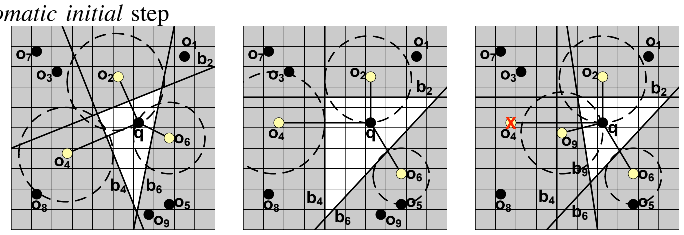
Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.



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

Fig. 1. Example of the *monochromatic initial* step



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

Fig. 2. An example of the *monochromatic Incremental* step

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 *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 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 Figure 2a while $RNN = \{o_2\}$ in Figure 2b. However, if there are 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, $RNN = \{o_2, o_9\}$.

C. Design Decisions

A significant computational cost within the monochromatic IGERN method occurs within the verification step in the incremental method (Line 10 of Algorithm 2). After each time interval, each of the RNN answers found in the previous time interval must be verified by performing a NN search. To reduce the number of NN searches at every time interval, we propose an optimization to Algorithm 2 to monitor the region around each RNN answer with a radius of the distance to the query object. This optimization only applies to the scenario when the RNN answer r and the query object q do not move to a new location at the next immediate time interval. In all other scenarios (i.e., objects r and/or q moves to a new location), the original verification phase (Line 10 of Algorithm 2) will be performed. If both r and q do not move to a new location and an object o enters the monitoring region of r, then r is no longer a RNN of q and no NN search needs to be performed. Otherwise, if an object does not move into the monitoring region of r, then r remains to be a RNN for q and no NN search needs to be performed to verify r. Unlike traditional continuous NN approaches where the monitoring region shrinks when an object enters the monitoring region, this optimization removes the monitoring region when r is no longer a RNN. Only a monitoring region is kept for RNN answers for q. Experimental evaluation shows that when objects do not move as often at every time interval, a significant savings may occur for the proposed optimization (Section VII-C.1 and Section VII-D.1).

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

D. Discussion

To summarize the difference between the IGERN *incremental* step and the state-of-the-art RNN algorithm, CRNN [10], 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 that of CRNN as CRNN is more likely to have open-ended regions than IGERN. Thus, IGERN 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 VI and experimentally in Section VII. The *initial* step in IGERN is similar to the static approach TPL [17] with the difference that we embed new functionalities to produce a set of objects that will be monitored later in the *incremental* step.

IV. A Filter and Refine Approach to the 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 2D space, 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 *monochromatic* and *bichromatic* reverse nearest neighbors, the IGERN algorithm still keeps the same flavor and the same framework to handle both *monochromatic* and the *bichromatic* reverse nearest neighbor queries. This section presents the FR-IGERN algorithm using a filter and refine-like approach for *bichromatic* reverse nearest neighbors; it is an improved optimization of our previous work [21] and an adaptation of the *monochromatic* IGERN algorithm. In general, there are two main phases in our previous bichromatic approach [21]: the first phase creates the bounded region similar to the monochromatic case and is followed by the second phase, which determines the reverse nearest neigbors within the bounded region. This second phase is the major bottleneck of computation time due to the excessive amount of nearest neighbor calculations. The proposed filter and refine approach reduces the computation time in the second phase by the insight that objects having a distance to the query object in the bounded region that are less than the distance to each of the borders of the bounded region do not have to perform a nearest neighbor search. These objects are already guaranteed to be the reverse nearest neighbors of the query object and are filtered, where all the other objects in the bounded region are refined to determine the remaining reverse nearest neigbors.

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, which is executed only once to report the first query answer and the *incremental* step, which is triggered at each time unit for all time intervals T to continuously maintain the query answer.

Throughout this section, Figure 3 gives a running example of 16 moving objects of two different types: six square objects of type A, o_{A1} to o_{A6}; nine circle objects of type B, o_{B1} to o_{B9}; 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 IV-A and IV-B, respectively, while Section IV-C gives a general discussion of the *bichromatic* FR-IGERN algorithm.

A. Step 1: Getting the Initial Answer

For a query object of type A (q_A), the objectives of the *initial* step in the *bichromatic* FR-IGERN algorithm are: (1) Obtaining a bounded region r around q_A to be monitored in the *incremental* step, (2) Identifying a set of 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 and the Grid index G while the output is the two

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Algorithm 3 Pseudo code for FR-IGERN Initial Step

```
 1: Function INITFR-IGERN(Query $q_A$, Grid index G)
 2: $NN_A \leftarrow \emptyset$, Mark all grid cells $\in G$ as alive
    Phase I: Bounded Region
 3: while ($o_A \leftarrow$ the nearest $A$ object to $q_A$ in alive cells) $\neq$ NULL do
 4:     $NN_A \leftarrow NN_A \cup o_A$, $b \leftarrow$ The bisector of $q_A$ and $o_A$
 5:     Mark grid cells in the half-plane bounded by $b_j$ and containing $o_j$ as dead
 6: end while
    Phase II: Filter Step
 7: $RNN_B \leftarrow \emptyset$
 8: $REFINE_B \leftarrow \emptyset$
 9: for each object $o_B \in$ the $alive$ cells do
10:     if min($\perp$ dist($o_B, b \in NN_A$)) $\geq$ dist($o_B, q_A$) then
11:         $RNN_B \leftarrow RNN_B \cup o_B$
12:     else
13:         $REFINE_B \leftarrow REFINE_B \cup o_B$
14:     end if
15: end for
    Phase III: Refine Step
16: for each object $o_B \in REFINE_B$ do
17:     $o_A \leftarrow$ is the nearest object of type $A$ to $o_B$
18:     if $o_A = q_A$ then
19:         $RNN_B \leftarrow RNN_B \cup o_B$
20:     else
21:         $NN_A \leftarrow NN_A \cup o_A$, $b \leftarrow$ The bisector of $q_A$ and $o_A$
22:         Mark grid cells from $b$ to the furthest boundaries of $q_A$ as dead
23:         For any two objects $o_A, o_{A'} \in NN_A$, remove object $o_A$ from $NN_A$ only if $dist(o_A, o_{A'}) < dist(o_A, q_A)$
24:     end if
25: end for
26: return $RNN_B$, $NN_A$
```

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 have the potential of containing a reverse nearest neighbor of q_A. Then, the *initial* step has the following three 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 in the half-plane bounded by b_j and containing o_j are marked as *dead* (Lines 4-5 in Algorithm 3). This process continues until there are no more objects of type A in any of the *alive* cells. Grid cells having a perpendicular bisector intersecting them and a border around the query object are still considered "alive" but only objects in the half-plane containing the query object is considered to be "alive". 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: Filter. This phase aims to filter objects that are guaranteed to be the reverse nearest neighbors for the query object. The main idea is to filter an object o_B such that the minimum perpendicular distance from o_B and each of the bisectors b found in the previous phase is greater or equal to the distance from o_B and the query object (Lines 9-10 of Algorithm 3). The filtered objects are then guaranteed to be a reverse nearest neighbor of the query object (Line 11 of Algorithm 3). Objects in the *alive* cells that cannot be filtered will need further refinement in Phase 3 (Line 13 of Algorithm 3). Figure 3b depicts the filtering concept as a region around the query object where any object in this region is guaranteed to be the reverse nearest neighbor of the query object. In this example, the greyed objects o_{B4}, o_{B5}, and o_{B6} are filtered (i.e., no further calculations are needed on these objects) and added to the answer set, RNN_B.

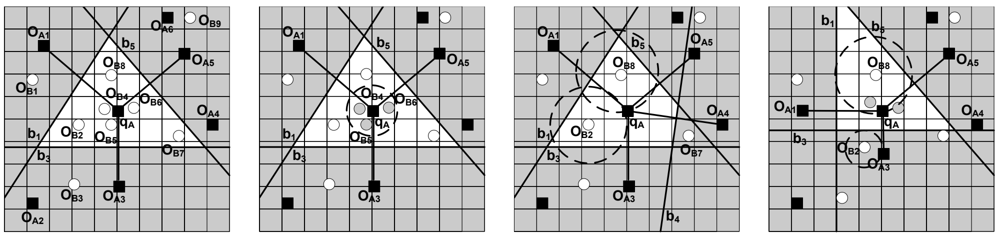
Phase III: Refine. This phase aims to refine any remaining objects within the bounded region to find additional reverse nearest neighbors, tighten the monitored bounded region, and modify the list of objects that need to be monitored in the *incremental* step. The main idea is to go through every object o_B that needs further refinement for its nearest A object, o_A (Lines 16-17 in Algorithm 3). If it ends up that the nearest o_A is the query object q_A, then o_B is considered as a reverse nearest neighbor to q_A, and thus added to the set *RNN*$_B$ (Line 19 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

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Algorithm 4 Pseudo code for FR-IGERN Incremental Step

```
1: Function INCRFR-IGERN(Query $q_A$, set $NN_A$, Grid index $G$)
2: if $q_A$ or any objects $\in NN_A$ 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 $o_A$ 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 $o_A, o_{A'} \in NN_A$, 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 and III of Algorithm 3 (Lines 7 to 25)
11: return $RNN_B$, $NN_A$
```



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

Fig. 3. An example of the *bichromatic* FR-IGERN.

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 21-23 in Algorithm 3). Figure 3c depicts such a scenario where there are three B objects in the *alive* cells that need further refinement, o_{B2}, o_{B7}, and o_{B8}. Upon testing for their nearest A objects (the dotted circles in Figure 3c), it turns out that only o_{B2}, and o_{B8} are reverse nearest neighbors while o_{B7} 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 $RNN_B = \{o_{B2}, o_{B8}\}$ is added to the answer set.

B. 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, 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, and the Grid index G. 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 *filter* and *refine* phases 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 has moved, then new bisectors 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 are 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).

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Figure 3d 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_{B2} has o_{A3} as its nearest A object. Thus, o_{B2} is no longer a reverse nearest neighbor to q_A. After the filter (grey objects) and refine steps, the returned answer from the *incremental* step will be $NN_A = \{o_{A1}, o_{A3}, o_{A5}\}$ while $RNN_B = \{o_{B4}, o_{B6}, o_{B8}\}$.

C. Discussion

FR-IGERN is an optimization method that addresses continuous *bichromatic* reverse nearest neighbor queries. One of the main attractive features of the FR-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., [29], [30], [31]) 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., [16], [10]) cannot be extended to the *bichromatic* case where the number of reverse nearest neighbors could be much greater than six. The *initial* step of FR-IGERN is similar to getting only the Voronoi cell around the query object [32]; 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. One of the main insights of the FR-IGERN method is the fact that objects that are near to the query objects do not have to perform a nearest neighbor search since no other objects can be its reverse nearest neighbor and can be filtered. Thus, the main distinction between the bichromatic FR-IGERN and the original bichromatic IGERN [21] is that the number of nearest neighbor searches may be reduced due to the filter and refine steps.

V. 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.

A. 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 until 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 a 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. A 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. ■

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

B. Bichromatic: Accurate and Complete

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

Proof: Assume that for the query q_{A_T}, the *bichromatic* FR-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* FR-IGERN conclude with the filter and refine steps which verify each returned o_B object as the reverse nearest neighbor. If the object o_B is found within the filter phase, then no other o_A object can exist because the distance from o_B to the query q_A will always be less than or equal to the distance between q_A and its borders. Otherwise, o_B is found in the refine phase that verified that every object o_B has to have q_A as the nearest A object. Thus, object o'_A cannot exist, and hence, the object o_B returned by the *bichromatic* FR-IGERN is an exact reverse nearest neighbor to q_A.

The order of the monitored objects found does not matter when finding the correct bounded region around the query object. This is ensured in Phase III of the refine step where objects are are not ensured to be RNNs, a NN search is performed and if the query object is not the NN of an object within the alive cells, a new bisector is created. Thus, the actual Voronoi cell can be created using this technique. ∎

Theorem 4: For any query q_{A_T}, of type A, executed at time T, the bichromatic FR-IGERN algorithm returns ALL reverse nearest neighbors to q_{A_T}.

Proof: Assume that for the query q_{A_T}, the *bichromatic* FR-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 proof for Theorem 3, object o_B is either in an *alive* or a *dead* cell in which both cases cannot take place in the filter and refine phases of Algorithms 3 and 4. Thus, object o_B cannot exist and the *bichromatic* FR-IGERN produces all the reverse nearest neighbors to q_A. ∎

VI. 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 [10], and repetitive evaluation of the static TPL algorithm [17]. For the *bichromatic* case, we present the cost model for IGERN [21], the refined Filter and Refine FR-IGERN approach, and the repetitive computations of the static creation of Voronoi cells [32]. Finally, we compare the cost model of IGERN with its counterparts for both the *monochromatic* and *bichromatic* cases.

For each cost model, the total time T is used to determine the entire costs of the methods for the complete duration. In general, the sampling rate of a 'middleware' system may change due to the speed of the object. Since our methods sit on top of this architecture, the total number of samples is accounted for within our cost models. The details of this middleware architecture are beyond the scope of this paper.

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 upon a grid index G, $mI(q,G) = mI_{init}(q_0, G) + \Sigma_{t=1}^{T} mI_{incr}(q_t, G)$ 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, $mI(q,G) = r_0(NN_c(q_0,G) + NN(q_0,G)) + \Sigma_{t=1}^{T}(NN_b(q_t,G) + r_t NN(q_t,G))$, where r_t is the number of objects that are candidates to be RNNs in time step t, ($r_t \leq 6$). $NN_c(q_t,G)$, $NN(q_t,G)$, and $NN_b(q_t,G)$ represent the cost for the *constrained*, *unconstrained*, and *bounded* nearest neighbor algorithms to q_t, respectively. The *constrained* NN search is done only within the remaining *alive* cells (Line 3 in Algorithm 1) while the *unconstrained* NN search is performed in the whole space (Line 7 in Algorithm 1) and the *bounded* NN search 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, $C(q,G) = 6(NN_c(q_0,G)+NN(q_0,G))+\Sigma_{t=1}^{T} 6(NN_b(q_t,G)+NN(q_t,G))$, where $NN_c(q_i,G)$, $NN(q_i,G)$, 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, $L(q,G) = \Sigma_{t=0}^{T} r_t(NN_c(q_t,G) + NN(q_t,G))$. 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:
$$\frac{mI(q,G)}{C(q,G)} = \frac{r_0(NN_c(q_0,G)+NN(q_0,G))+ \Sigma_{t=1}^{T}(NN_b(q_t,G)+r_t NN(q_t,G))}{6(NN_c(q_0,G)+NN(q_0,G))+ \Sigma_{t=1}^{T} 6(NN_b(q_t,G)+NN(q_t,G))}$$

Lemma 1: If $r_t < 6$, $mI(q,G) < C(q,G)$

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Proof: For any single time instance T, the ratio is $\frac{r_0}{6}$ if $T = 0$ and $\frac{NN_b(q_t,G)+r_tNN(q_t,G)}{6NN_b(q_t,G)+6NN(q_t,G)}$ 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. When $r_t < 6$, IGERN always achieves better performance than CRNN. ■

Lemma 2: As the number of objects increases, $mI(q, G) < C(q, G)$.

Proof: The performance of a single execution of a bounded, constrained, and unconstrained nearest neighbor search requires the lookup of objects in a single grid cell. As the number of objects increases, more lookups in the grid cell are required. For $T > 0$, CRNN requires six bounded and six unconstrained bounded nearest neighbor searches whereas IGERN needs a single bounded and r_t unconstrained searches. Thus the gap in the costs between IGERN and CRNN will increase as more objects are placed in the grid index. ■

Lemma 3: As the number of time intervals T increases, $mI(q, G) < C(q, G)$.

Proof: In the incremental step for IGERN and CRNN, the worst case of $r_t = 6$, the performance in IGERN will be a little less than half of the cost in CRNN. Thus, as T is increased, the savings in IGERN will significantly improve. Further savings may occur at a degree of r_t when the monitored number of objects is less than six. ■

Lemma 4: As the spatial area of the dataset decreases, $mI(q, G) < C(q, G)$.

Proof: As the spatial area of the dataset decreases, so does the number of grid cells that needs to be analyzed, thereby reducing the peformance cost of both algorithms. However, based on the cost ratio between IGERN and CRNN, IGERN will still show better performance than its counterpart. ■

IGERN vs. TPL. The cost ratio between the *monochromatic* IGERN and the reevaluation of the static TPL is: $\frac{mI(q,G)}{L(q,G)} = \frac{r_0(NN_c(q_0,G)+NN(q_0,G))+\ \Sigma_{t=1}^{T}(NN_b(q_t,G)+r_tNN(q_t,G))}{\Sigma_{t=0}^{T}r_t(NN_c(q_t,G)+NN(q_t,G))}$

Lemma 5: As the number of objects increases, $mI(q, G) < L(q, G)$.

Proof: As in Lemma 2, the cost of a nearest neighbor search will increase as more objects occur in the grid index G. Based on our cost ratio between IGERN and TPL, TPL will always perform more nearest neighbor searches than IGERN for all possible values of r_t. Also, TPL will perform r_t constrained nearest neighbor searches whereas IGERN will perform a single bounded nearest neighbor search. Thus, IGERN will improve in performance over TPL as more objects occur in the grid index. ■

Lemma 6: As the number of time intervals T increases, $mI(q, G) < L(q, G)$.

Proof: In Lemma 6, for any single time instance T, the ratio is one if $T = 0$ while the ratio is $\frac{NN_b(q_t,G)+r_tNN(q_t,G)}{r_tNN_c(q_t,G)+r_tNN(q_t,G)}$ 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. ■

Lemma 7: As the spatial area of the dataset decreases, $mI(q, G) < L(q, G)$.

Proof: As in Lemma 4, the effect of the size of the dataset is the same as in Lemma 7. ■

Bichromatic IGERN cost. Let *bI* be the cost function for the *bichromatic* IGERN algorithm. Then, for a query of type A (q_A, G) that is executed for T time units upon a grid index G, $bI(q_A, G) = bI_{init}(q_A, G_0) + \Sigma_{t=1}^{T} bI_{incr}(q_A, G_t)$ where bI_{init} and bI_{incr} are the cost of the *initial* and *incremental* steps, respectively, while q_{At} is the execution of the query q_A at time t. Thus, $bI(q_A, G) = a_0NN_c(q_A, G_0)+b_0NN(q_A, G_0)+\ \Sigma_{t=1}^{T}(NN_b(q_A, G_t)+ b_tNN(q_A, G_t))$ 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. $NN_c(q_t, G)$, $NN_b(q_t, G)$, and $NN(q_t, G)$ are similar to those of the *monochromatic* IGERN algorithm. Notice that the constrained NN search is done only once in the *initial* step while the bounded NN search is done only once at each execution of the *incremental* step.

Bichromatic FR-IGERN cost. Let *bFRI* be the cost function for the *bichromatic* FR-IGERN algorithm. Then, for a query of type A (q_A, G) that is executed for T time units upon a grid index G, $bFRI(q_A, G) = bI_{init}(q_A, G_0) + \Sigma_{t=1}^{T} bI_{incr}(q_A, G_t)$, where bI_{init} and bI_{incr} are the cost of the *initial* and *incremental* steps, respectively, while q_{At} is the execution of the query q_A at time t. Thus, $bFRI(q_A, G) = a_0NN_c(q_A, G_0)+(b_0-c_0)NN(q_A, G_0)+\ \Sigma_{t=1}^{T}(NN_b(q_A, G_t)+ (b_t - c_t)NN(q_A, G_t))$ where a_t, b_t, and c_t are the number of A objects that need to be monitored, the number of B objects in the monitored bounded region, and the number of C objects found within the Filter step, at time step t, respectively. The cost of finding the objects in the filter step is not included since the cost is minimal for a simple distance calcuation rather than a NN search. $NN_c(q_t, G)$, $NN_b(q_t, G)$, and $NN(q_t, G)$ 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.

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

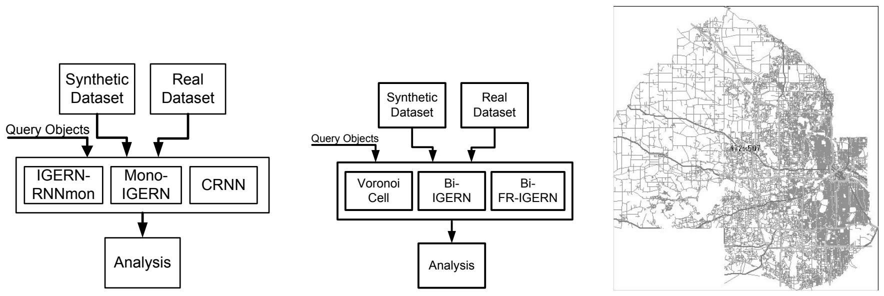
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Voronoi cost. Let V be the cost function for the continuous re-creation of a Voronoi cell around q_A. Then, $V(q_A, G) = \Sigma_{t=0}^{T} a_t NN_c(q_t, G) + b_t NN(q_t, G)$. 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: $\frac{bI(q,G)}{V(q,G)} = \frac{a_0 NN_c(q_0,G)+b_0 NN(q_0,G)+\ \Sigma_{t=1}^{T}(NN_b(q_t,G)+b_t NN(q_t,G))}{\Sigma_{t=0}^{T}(a_t NN_c(q_t,G)+b_t NN(q_t,G))}$

Lemma 8: As the number of objects increases, $bI(q, G) < V(q, G)$.

Proof: As in Lemmas 2 and 5, the nearest neighbor cost will be higher in Voronoi cell due to the repititive executions of the constrained nearest neighbor search (Lemma 8). Thus, when the number of objects increases, the cost in the Voronoi cell approach will be much higher than in IGERN. ∎



(a) Monochromatic (b) Bichromatic (c) Map of Hennepin County, MN, USA.

Fig. 4. Experimental Setup

Lemma 9: As the number of time intervals T increases, $bI(q, G) < V(q, G)$.

Proof: For any single time instance T, the ratio is one if $T = 0$ and $\frac{(NN_b(q_t,G)+b_t NN(q_t,G))}{(a_t NN_c(q_t,G)+b_t NN(q_t,G))}$ for $T > 0$. Notice that the bounded search in IGERN is cheaper than the constrained search in the Voronoi cell. Furthermore, the bounded search in IGERN is done only one time at each time instance while the constrained search is done a_t times in the Voronoi cell construction. Thus, the *bichromatic* IGERN achieves better performance than repetitive construction of Voronoi cells. ∎

Lemma 10: As the spatial area of the dataset decreases, $bI(q, G) < V(q, G)$

Proof: The effect of the size of the dataset is the same as in Lemma 10. ∎

IGERN vs. FR-IGERN. The cost ratio between the *bichromatic* IGERN algorithm and the *bichromatic* FR-IGERN algorithm is: $\frac{bI(q,G)}{bFRI(q,G)} = \frac{a_0 NN_c(q_0,G)+b_0 NN(q_0,G)+\Sigma_{t=1}^{T}(NN_b(q_t,G)+b_t NN(q_t,G))}{a_0 NN_c(q_0,G)+(b_0-c_0)NN(q_0,G)+\Sigma_{t=1}^{T}(NN_b(q_t,G)+(b_t-c_t)NN(q_t,G))}$

Lemma 11: As the number of objects increases, $bFRI(q, G) < bI(q, G)$ when one or more objects are filtered.

Proof: As more objects are found in the filter step of FR-IGERN, fewer nearest neighbor searches are required to verify the remaining reverse nearest neighbors. Thus, as the number of objects increases, more objects may be filtered, reducing the execution time over IGERN. ∎

Lemma 12: As the number of time intervals increases, $bFRI(q, G) < bI(q, G)$ when one or more objects are filtered.

Proof: As objects are filtered within FR-IGERN, the number of nearest neigbor searches as the time intervals increase will be significantly lower than in IGERN, where all objects within the bounded region must be verified. ∎

Lemma 13: As the spatial area of the dataset decreases, $bFRI(q, G) < bI(q, G)$ when one or more objects are filtered.

Proof: The effect of the size of the dataset is the same as in Lemma 10. ∎

VII. Experimental Results

In this section, we experimentally evaluate the performance of IGERN for both *monochromatic* (Section VII-B.1) and *bichromatic* (Section VII-B.2) reverse nearest neighbors. In the *monochromatic* case, we compare our proposed optimization for IGERN (denoted as IGERN-RNNmon) against IGERN and the state-of-the-art algorithm for continuous *monochromatic* reverse nearest neighbor queries CRNN [10]. In the *bichromatic* case, our proposed FR-IGERN is compared with our previous approach [21] and a static approach of repetitive computation of Voronoi

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

cells [32]. To ensure consistency and fairness among different approaches, we use the algorithm in [11] as the underlying nearest neighbor search for all approaches of reverse nearest neighbor queries (See Section VII-A.4 for details).

1) Setup:

A. Experimental Design

Figure 4 gives the experimental setup to evaluate our approaches. The input to both the Monochromatic and Bichromatic IGERN is a synthetic and a real dataset which are described in Sections VII-A.1 and VII-A.2 respectively. We experimentally compare our proposed monochromatic optimization (denoted as IGERN-RNNmon) against continuous query processing methods IGERN and CRNN. The bichromatic approach is compared to our previous approach and the repititive creation of a Voronoi cell. Analysis is performed on the synthetic dataset by observing the effect on the number of objects and the number of time intervals. Real dataset analysis is performed to measure the practicality of each of the methods. All experiments were performed on an Intel Pentium IV CPU 2.0GHz with 512MB RAM.

1) Synthetic Dataset Generator: We use the *Network-Based Generator of Moving Objects* [33] 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 4c). The output of the generator is a set of moving objects that move on the road network of the given city.

In our synthetic experiments, we used two different sets of experiments: (1) Number of Objects and (2)Number of Time intervals. The first synthetic dataset was generated using the generator based on the following settings: (1) Maximum time of 100, (2) Maximum number of objects as 100,000, (3) a report probability of 1000 (i.e., objects are reported at every timestamp), (4) the number of objects generated at the beginning of 100,000, (5) the number of moving objects generated at every time stamp of 1000 to account for objects being deleted, and (6) 1 class for monochromatic and 2 classes for bichromatic (each class had approximately the same number of objects). The external objects were ignored. For experiments requiring less than 100,000 objects or less than 100 time intervals, subsets of this generated dataset were used. The second synthetic dataset was based on the number of time intervals. Since only 30 objects were used and a maximum of 30,000 time intervals, 30 objects were taken from the prior dataset. The time intervals were then interpolated based on a a constant rate to create 30,000 time intervals. Due to this interpolation, objects may not move as often as the original dataset. Again, subsets of the second dataset were taken for experiments with less than 30,000 time intervals.

2) Real Dataset: The real dataset used for our experiments is a courier dataset from a delivery company called *eCourier* [34]. This company is located in the United Kingdom and delivers packages throughout London. The public dataset consists of a set of GPS tracks of delivery vehicles where each vehicle reports its location approximately every ten seconds. We used the following five attributes from the courier dataset: (1) a timestamp containing the day and time of the reported location, (2) a unique courier identification number, (3) the type of delivery vehicle (e.g., van, motorbike, pushbike, etc.), (4) longtitude, and (5) latitude.

For our experiments, approximately 24 hours worth of data was extracted from *eCourier*, more specifically on May 12, 2007 (we assumed a high volume of delivery activity on this day because it was the day before Mother's Day). This dataset consisted of approximately 30 unique delivery vehicles having 30,000 time intervals. This dataset had two distinct object types: Van and Motorbike. For the monochromatic experiments, all vehicles were considered of the same type, whereas the bichromatic dataset considered both distinct object types. Figure 5 gives an example of the spatial distribution taken at 6K time intervals of the two object types: (1) Van (shown in red), and (2) Motorbike (shown in blue). Figure 5f gives an example of the spatio-temporal distribution of a single object taken at 15 minute samples. The object starts at approximately 7:15 AM on May 12, 2007 at the farthest east point near *Newham Way* and ends at approximately 5:50 PM on the same day near the original starting location at the intersection *A102*. This object remains at the ending location for the remainder of the extracted dataset. Unfortunately, this dataset does not have any information on why the object stops moving for a time period and then begins again, and we assume the vehicle has finished its deliveries and is waiting for another package to deliver.

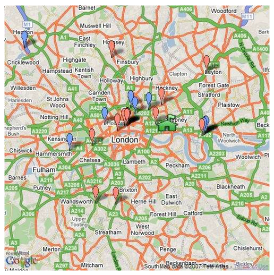
The raw dataset contained two main issues that needed to be addressed before any experiments could be executed. First, the dataset contained several pieces of missing information. Each object is reported at every ten seconds, but not all objects are started at the same time. Thus, there may be some time intervals that may not report all

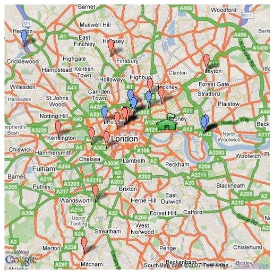
Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

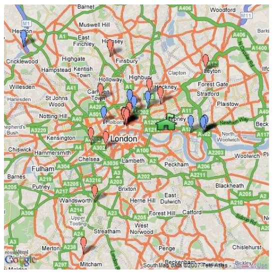
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

(a) $T = 6K$

(b) $T = 12K$


(c) $T = 18K$


(d) $T = 24K$


(e) $T = 30K$

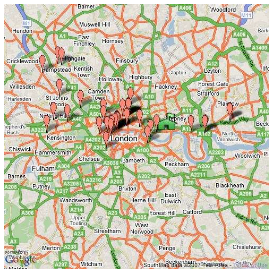
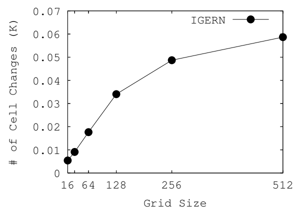
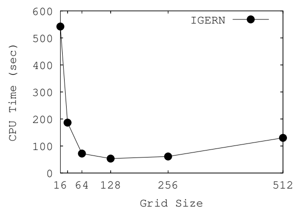

(f) Distribution of a single object

Fig. 5. Spatial Distribution of the Real Dataset (Best Viewed in Color) [Source: Generated using Google Maps]



(a) Number of Cell Changes



(b) CPU Time

Fig. 6. Grid Size Costs

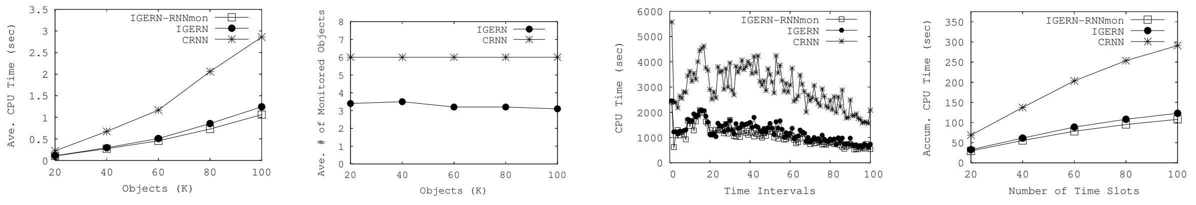
objects. This issue was addressed by taking the latest occurrence of the object if it was not reported at a certain time interval. Other methods such as interpolation using the speed of the object or pruning the dataset may be used for future work. Second, the dataset contained several duplicate occurrences of an object at the same time interval. These duplicate objects contained different locations but reported at the same time interval. To address this issue, only a single occurrence of the object was taken.

3) Grid Data Structure: To account for the continuously moving objects in our proposed approaches, we use a grid data structure to index the locations of each object. The grid data structure G of $N \times N$ equal size cells. Each cell $c \in G$ keeps track of the set of cells that lies within the cell boundary and whether the cell is considered "alive" or "dead". For cases where the bisector intersects with the cell and borders the query object, the cell itself is considered "alive" but only objects in the half-plane containing the query object are considered "alive".

Figure 6 gives the effect of increasing the grid size from 16×16 to 512×512 based on the synthetic experiments for the number of objects. Although this experiment was performed for the *monochromatic* IGERN, similar performance was found for the *bichromatic* case. We did not perform a similar study in the second set of experiments based on the number of time intervals and the real dataset because it contained only 30 objects. Figure 6a 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. By contrast, Figure 6b gives the CPU time which includes the update costs (i.e., objects moving to a new cell) and query costs when executing the IGERN algorithm for different grid sizes. For small grid sizes, IGERN encounters higher costs as each grid cell contains large numbers of objects, hence, the underlying nearest neighbor search would search within a lot of unnecessary objects. The performance improves when the grid size increases to 64-128 as the nearest neighbor search is performed within limited numbers of objects. However, greater increase

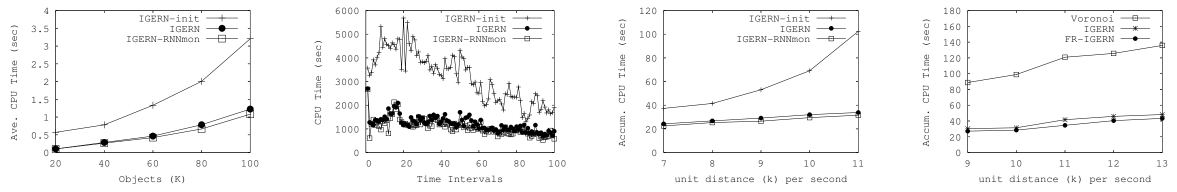
Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.



(a) Processing Time (b) Monitored Objects (c) Time Intervals (d) Time Slots

Fig. 7. Monochromatic Costs



(a) Processing Time (b) Time Intervals (c) Monochromatic: Speed (d) Bichromatic: Speed

Fig. 8. Monochromatic Self Evaluation and Effect on the Speed of the Objects

in the grid size badly affects IGERN due to the cost of data updates for objects that change their cells as shown in Figure 6a. As a compromise, in the rest of our experiments, we used a grid structure of size 64.

4) Nearest Neighbor Algorithms: In all of the reverse nearest neighbor methods presented in the experimental section, our underlying nearest neighbor search is [11] (denoted as YPK). These authors proposed a k-nearest neighbor approach for moving objects and all of our methods use this simplest form of a 1-nearest neighbor. Using a grid data structure to track the moving objects, YPK expands a set of two rectangles around the query object to find the first nearest neighbor. The first or inner rectangle is centered around the query object and the second or outer rectangle surrounds the approximated circle around the query object using the first rectangle. All objects within the grid cells of the outer rectangle are examined to determine the nearest object to the query. If there are no objects found, this process continues and the rectangles are expanded until the 1-NN is found.

B. Synthetic Dataset Experiments Based on Number of Objects

1) Monochromatic RNNs: This section analyzes the effect on the number of objects by comparing the *monochromatic* IGERN-RNNmon to IGERN and CRNN.

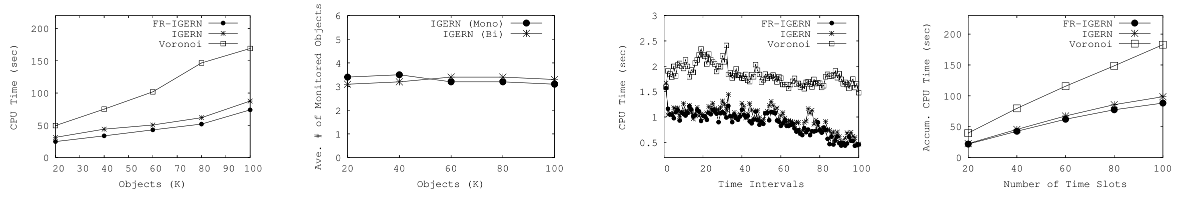
Figure 7a gives the effect of increasing the number of moving objects from 20K to 100K on IGERN-RNNmon, IGERN, and CRNN. The average CPU time is taken over 100 time units. The IGERN consistently performs better than CRNN. As the number of objects increases, IGERN takes advantage of the fact that it monitors only one region and a smaller number of objects. By contrast, 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. This experiment validates Lemma 2 and Lemma 5 based on the effect on the number of objects. IGERN-RNNmon performs less than IGERN by reducing the number of NN searches in the verification phase. Only a slight decrease occurs due to only a few number of objects not moving at every time interval.

Figure 7b gives the average number of monitored objects in both IGERN and CRNN for different data sizes. While CRNN monitors exactly six objects, IGERN, on average, monitors about 3.3 objects. These results are in line with the results of Figure 7a, which shows that IGERN has much less CPU time than CRNN. This experiment validates Lemma 1 by showing that the CPU costs are lower when the monitored number of objects is less than in CRNN. Since IGERN and IGERN-RNNmon are essentially the same except for the number of NN searches in the verification phase for a specific scenario (i.e., RNN answer and query object do not move to a new location), both have the same number of monitored objects.

Figure 7c gives the execution time of IGERN-RNNmon, IGERN, and CRNN for the first 100 time units of 100K moving objects. As all three algorithms are designed for *continuous* evaluation, the first time evaluation at time $T = 0$ is more expensive than later time instances that only maintain only the current answer. CRNN

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.



(a) Processing Time (b) Monitored Objects (c) Time Intervals (d) Time Slots

Fig. 9. Bichromatic Costs

is more expensive than IGERN because of the creation of six regions as opposed to one in IGERN. IGERN is slightly more expensive than IGERN-RNNmon due to several more NN searches in the verification phase. The stable performance of IGERN-RNNmon for all time units $T > 0$ indicates that the performance of the *incremental* step does not deteriorate over time. This experiment validates Lemma 2 and Lemma 5 for 100K moving objects throughout 100 time intervals. The decline in CPU costs of all methods is due to an artifact in the synthetic generator discussed in Section VII-A.1.

Figure 7d gives the total accumulated time spent on one execution of the *initial* step followed by a sequence of executions of the *incremental* step for IGERN-RNNmon, 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 and validates both Lemmas 2 and 5.

Figure 8a gives the monochromatic self evaluation of the average time for the repititive executions of the initial step of IGERN and the other two proposed approaches IGERN and IGERN-RNNmon on 20K to 100K moving objects. It is clear that the repititive executions of the initial step of IGERN performs more poorly than the proposed incremental approaches because the bounded region needs to be recomputed at every time interval. Figure 8b gives the self evaluation of the execution time at each time interval for the first 100 time units of 100K moving objects. At $T = 0$, both continuous approaches is more expensive than the later time instances. Whereas, the static approach of the initial step of IGERN is much more expensive at all time intervals due to its repititive computations of the bounded region.

Figure 8c gives the monochormatic self evaluation on the effect of the speed of the objects. The speed of the objects is based on the average unit distance traveled per second for 20 time intervals. Figure 8c gives the accumulated execution time based on the speed of the object from 7K to 11k unit distances per second. In general, the speed of the object may affect the computation time of the proposed continuous approaches, IGERN and IGERN-RNNmon, due to the size of the monitoring region increasing. As expected, the static approach of the initial step of IGERN is much more expensive than the continuous approaches due to its recomputation of the bounded region. Likewise, the computation costs of both continuous approaches also increase as the speed of the objects increase.

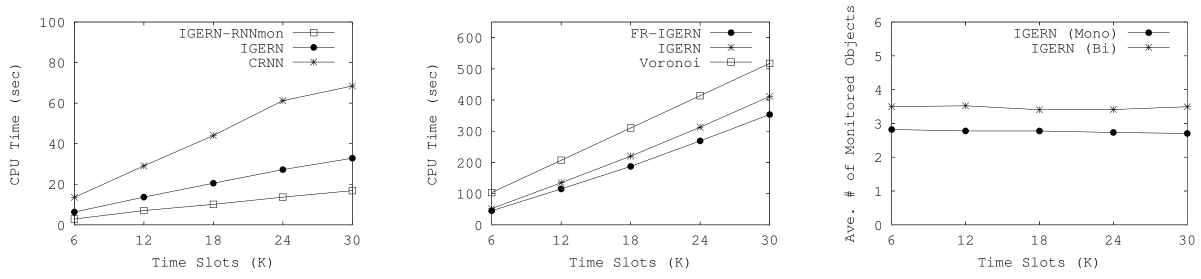
2) Bichromatic: This section analyzes the effect on the number of objects by comparing the *bichromatic* FR-IGERN algorithm to our previous *bichromatic* IGERN algorithm and the repetitive computation of static Voronoi cells.

Figure 9a gives the effect of increasing the number of moving objects from 20K to 100K on FR-IGERN, IGERN and the creation of a Voronoi cell where each object type has 10K to 50K objects. The FR-IGERN algorithm consistently gives better performance than Voronoi as FR-IGERN only needs to maintain the answer. The increase in CPU time of FR-IGERN with the increase of the number of objects is much less than that of Voronoi. Basically, FR-IGERN takes advantage of the fact that it monitors only one region and a small number of objects rather than reconstructing the answer at each time instance. FR-IGERN also performs better than our previous IGERN method due to the filtered objects in FR-IGERN, which reduces the number of nearest neigbor searches compared to IGERN. Thus, FR-IGERN can scale well to a large number of moving objects in comparison to previous algorithms and validates Lemma 8 and 11.

Figure 9b gives the number of monitored objects over the course of execution for both the *monochromatic* and *bichromatic* approaches. IGERN performs very similarly for both cases and even though the case of *bichromatic* reverse nearest neighbors is more complicated, IGERN can still achieve similar performance for both cases. Also,

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.



(a) Monochromatic Processing Time (b) Bichromatic Processing Time (c) Monitored Objects

Fig. 10. Synthetic Data: Number of Time Intervals

the number of monitored objects will be the same in both *bichromatic* IGERN and FR-IGERN because the main difference occurs within the verification and not in the bounded region phase.

Figure 9c gives the execution of FR-IGERN, our previous method IGERN, and repetitive Voronoi cell creation for the first 100 time units. Only at the first time instance ($T = 0$) does Voronoi perform better. This is basically because both FR-IGERN and 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 $T > 0$, both FR-IGERN and IGERN consistently give much higher performance than Voronoi. Also, FR-IGERN gives slightly better performance than IGERN due to the reduction in nearest neighbor searches. The reduction in computation time over the time intervals for all three approaches is due to an artifact in the synthetic data generator discussed in Section VII-A.1.

Figure 9d gives the total accumulated time spent on one execution of the initial step followed by a sequence of executions of the *incremental* step for FR-IGERN, IGERN, and the repetitive construction of Voronoi cells. It is clear that as time continues, both FR-IGERN and IGERN save a great amount of CPU time as their incremental approach avoids the intensive computations of Voronoi cells. FR-IGERN also shows better performance over IGERN due to fewer nearest neighbor searches. Both experiments in Figures 9c and 9d validates Lemma 8 and 11.

Figure 8d gives the bichromatic accumulative execution time when the average speed of the objects increases from 9K to 13K unit distance per second for 100K moving objects on the first 20 time intervals. In general, as the speed of the objects increases, the monitored region may also increase. Thus, as the speed of the objects increases, the computation costs also increases for all bichromatic approaches.

C. Synthetic Dataset Experiments Based on Number of Time Intervals

1) Monochromatic RNNs: This section analyzes the effect on the number of time intervals by comparing the *monochromatic* IGERN-RNNmon algorithm to IGERN and CRNN.

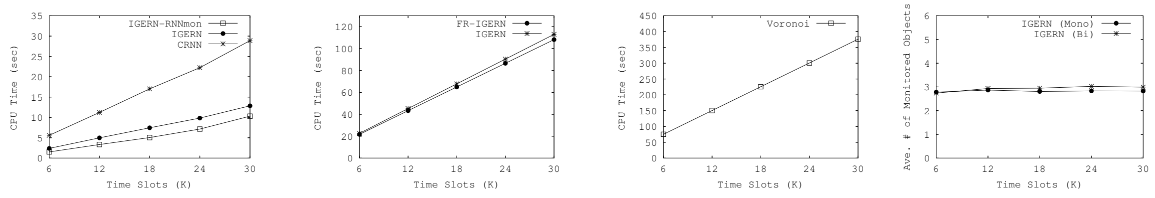
Figure 10a gives the total accumulated CPU time of one initial and a sequence of incremental steps for IGERN-RNNmon, IGERN and CRNN. The CPU costs of each time slot are not illustrated because the incremental costs for both continuous methods having a small number of objects (30) in the dataset may not be significant; thus, only the total accumulated time is shown. IGERN performs better than CRNN due to its monitoring of a single region versus six regions in CRNN. This experiment validates Lemma 3 and Lemma 6 by showing that IGERN scales better than CRNN as the number of time intervals increase. Due to the fact that objects may not move as often, IGERN-RNNmon performs significantly better than IGERN by reducing the number of NN searches needed in the verification phase.

Figure 10c gives the average number of objects monitored by the *monochromatic* IGERN and *bichromatic* FR-IGERN approach. For 30K time intervals and 30 moving objects, the average number of monitored objects is less than 3 objects. This experiment and the CPU costs in Figure 10a validates the Lemma 1 that IGERN will perform better than CRNN when the number of monitored objects is less than six. Also, there may be less than three monitored objects when the number of objects in the dataset is low or due to the edge effect in the spatial region. Since IGERN-RNNmon is essentially the same as IGERN in terms of how the monitored objects are found, both will have the same number of monitored objects.

2) Bichromatic RNNs: This section analyzes the effect on the number of time intervals by comparing the *bichromatic* IGERN algorithm to the repetitive creation of Voronoi Cells.

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.



(a) Monochromatic (b) Bichromatic: FR-IGERN vs. IGERN (c) Bichromatic: Voronoi (d) Monitored Objects

Fig. 11. Number of Time Intervals

Figure 10b gives the total accumulated CPU time of one initial and a sequence of incremental steps for FR-IGERN, IGERN, and the repetition of Voronoi cells. The individual time slot costs are not illustrated because the CPU costs to process a small set of objects (30) for both the continuous and static methods may not be significant and thus only the total accumulated time is illustrated. The costs for repetitively creating Voronoi cells are significantly greater because of the re-creation of the bounded region at every time slot. By contrast, both FR-IGERN and IGERN maintain a single monitored region which reduces the costs. FR-IGERN shows slightly better performance due to the reduction in the number of nearest neighbor search executions than in IGERN and validates Lemma 9 and 12.

Figure 10c illustrates the behavior of both the *monochromatic* and *bichromatic* methods in terms of the number of monitored objects. The *bichromatic* method has an average of 3.3 monitored objects, whereas the *monochromatic* method has 2.9 objects. As in the *monochromatic* approach, it is quite possible to have less than 3 monitored objects due to the size of the dataset or to the edge effect on the given spatial region. Both FR-IGERN and IGERN show the same behavior in terms of monitored objects since the main distinction is in the verification phase.

D. Real Dataset Experiments

1) Monochromatic RNNs: This section compares the *monochromatic* IGERN-RNNmon algorithm to IGERN and CRNN using the real dataset from *eCourier* [34].

Figure 11a gives the total accumulated CPU time of one initial and a sequence of incremental steps for IGERN-RNNmon, IGERN, and CRNN. As in the overlapping synthetic experiment in Figure 10a, IGERN performs significantly better than CRNN due to its single monitored region. The difference in CPU costs in Figure 10a and in Figure 11a is because of the given spatial region of each dataset. The spatial region in the synthetic experiments is approximately three times larger than the region in the real dataset. The purpose of this comparison is to show that having a smaller spatial region and with resulting fewer grid cells leads to faster computations for all methods. Thus, this experiment validates Lemma 4 and Lemma 7 by showing that IGERN lower costs for smaller spatial regions while still performing better than CRNN. Also, IGERN-RNNmon shows signicant reduction in computational time over IGERN by reducing the number of NN searches needed in the verification phase.

Figure 11d gives the average number of monitored objects (≈ 2.9) over 30K time intervals and 30 objects. The behavior of *monochromatic* IGERN is very similar to the overlapping synthetic experiment in Figure 10c. Thus, this experiment shows the practicality of the *monochromatic* approach using real datasets. Since IGERN and IGERN-RNNmon is essentially the same in the way the monitored objects are found, both will have the same number of monitored objects.

2) Bichromatic RNNs: This section compares the *bichromatic* IGERN algorithm to the repetitive creation of Voronoi cells using the real dataset from *eCourier* [34].

Figure 11b and 11c gives the total accumulated CPU time of one initial and a sequence of incremental steps for FR-IGERN, IGERN, and the repetition of Voronoi cells. As in the overlapping synthetic experiment in Figure 10a, both FR-IGERN and IGERN perform significantly better than Voronoi cell due to their incremental nature and the monitoring of a single region. The difference in CPU costs in Figure 10b and in Figure 11b is due to the given spatial region of each dataset. The spatial region in the synthetic experiments is approximately three times larger than the region in the real dataset. The purpose of this comparison is to show that having a smaller spatial region and resulting fewer grid cells results in faster computations for all methods. Thus, this experiment validates Lemma 10 and 13 by showing IGERN has lower costs for smaller spatial regions while performing better than

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Voronoi cell. Also, FR-IGERN shows only a slight improvement over IGERN because only a small number of objects were filtered in this dataset.

Figure 11d gives the average number of objects for both *monochromatic* and *bichromatic* IGERN approaches. Both methods show similar behavior having three monitored objects. Also, the *bichromatic* approach shows similar behavior to that in the overlapping synthetic experiment in Figure 10c. This shows the practicality of the *bichromatic* approach using real datasets.

VIII. 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* in that it provides 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 small set of moving objects rather than focusing on the whole space. IGERN clearly shows enhanced performance over the state-of-the-art algorithms in continuous *monchromatic* reverse neighbor queries. We also propose a new optimization for the monochromatic IGERN to reduce the number of nearest neighbor searches in the validation phase which shows significant improvement over the IGERN approach. Furthermore, our previous IGERN is the first algorithm that deals with continuous *bichromatic* reverse nearest neighbors. An improved bichromatic FR-IGERN approach is proposed here using filter and refine steps. The correctness of both the *monochromatic* and *bichromatic* FR-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 for both *monochromatic* and *bichromatic* reverse nearest neighbor queries.

IX. Acknowledgements

This work has been supported by NSF IGERT, NSF SEI Award 431141, and partially supported by NSF CAREER Award IIS-0347600. We are grateful to Kim Koffolt for improving the readability of this paper.

References

[1] B. Gedik and L. Liu, “MobiEyes: Distributed Processing of Continuously Moving Queries on Moving Objects in a Mobile System,” in *EDBT*, 2004.
[2] H. Hu, J. Xu, and D. L. Lee, “A Generic Framework for Monitoring Continuous Spatial Queries over Moving Objects,” in *SIGMOD*, 2005.
[3] G. S. Iwerks, H. Samet, and K. Smith, “Continuous K-Nearest Neighbor Queries for Continuously Moving Points with Updates,” in *VLDB*, 2003.
[4] I. Lazaridis, K. Porkaew, and S. Mehrotra, “Dynamic Queries over Mobile Objects,” in *EDBT*, 2002.
[5] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou, “Conceptual Partitioning: An Efficient Method for Continuous Nearest Neighbor Monitoring,” in *SIGMOD*, 2005.
[6] M. F. Mokbel, X. Xiong, and W. G. Aref, “SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases,” in *SIGMOD*, 2004.
[7] 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*, vol. 51, no. 10, pp. 1124–1140, 2002.
[8] Y. Tao, D. Papadias, and Q. Shen, “Continuous Nearest Neighbor Search,” in *VLDB*, 2002.
[9] 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.
[10] T. Xia and D. Zhang, “Continuous Reverse Nearest Neighbor Monitoring,” in *ICDE*, 2006.
[11] X. Yu, K. Q. Pu, and N. Koudas, “Monitoring K-Nearest Neighbor Queries Over Moving Objects,” in *ICDE*, 2005.
[12] F. Korn and S. Muthukrishnan, “Influence Sets Based on Reverse Nearest Neighbor Queries,” in *SIGMOD*, 2000.
[13] A. Nanopoulos, Y. Theodoridis, and Y. Manolopoulos, “C2P: Clustering based on Closest Pairs,” in *VLDB*, 2001.
[14] 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.
[15] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis, “Nearest Neighbor and Reverse Nearest Neighbor Queries for Moving Objects,” in *IDEAS*, 2002.
[16] 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.
[17] Y. Tao, D. Papadias, and X. Lian, “Reverse kNN Search in Arbitrary Dimensionality,” in *VLDB*, 2004.
[18] Y. Tao, M. L. Yiu, and N. Mamoulis, “Reverse Nearest Neighbor Search in Metric Spaces,” *TKDE*, vol. 18, no. 8, 2006.

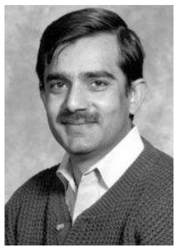
Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

[19] M. L. Yiu and N. Mamoulis, “Reverse Nearest Neighbors Search in Ad-hoc Subspaces,” in *ICDE*, 2006.
[20] M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao, “Reverse Nearest Neighbors in Large Graphs,” *TKDE*, vol. 18, no. 4, pp. 540–553, 2006.
[21] J. M. Kang, M. F. Mokbel, S. Shekhar, T. Xia, and D. Zhang, “Continuous Evaluation of Monochromatic and Bichromatic Reverse Nearest Neighbors,” in *ICDE*, 2007, pp. 806–815.
[22] C. S. Jensen, *Location-Based Services.* Morgan Kaufmann, 2004, ch. Database Aspects of Location-Based Services, pp. 115–148.
[23] M. F. Mokbel and W. G. Aref, “PLACE: A Scalable Location-aware Database Server for Spatio-temporal Data Streams,” *IEEE Data Engineering Bulletin*, vol. 28, no. 3, pp. 3–10, 2005.
[24] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras, “On-Line Discovery of Dense Areas in Spatio-temporal Databases,” in *SSTD*, 2003.
[25] C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang, “Effective Density Queries on ContinuouslyMoving Objects,” in *ICDE*, 2006.
[26] C. Yang and K.-I. Lin, “An Index Structure for Efficient Reverse Nearest Neighbor Queries,” in *ICDE*, 2001.
[27] A. Singh, H. Ferhatosmanoglu, and A. S. Tosun, “High Dimensional Reverse Nearest Neighbor Queries,” in *CIKM*, 2003.
[28] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis, “Nearest and reverse nearest neighbor queries for moving objects ,” *The VLDB Journal*, vol. 15, no. 3, pp. 229–249, 2006.
[29] 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.
[30] 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.
[31] O. Wolfson, A. P. Sistla, B. Xu, J. Zhou, and S. Chamberlain, “DOMINO: Databases fOr MovINg Objects tracking (Demo),” in *SIGMOD*, 1999.
[32] F. Aurenhammer, “Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure,” *ACM Computing Surveys*, vol. 23, no. 3, pp. 345–405, 1991.
[33] T. Brinkhoff, “A Framework for Generating Network-Based Moving Objects,” *GeoInformatica*, vol. 6, no. 2, pp. 153–180, 2002.
[34] “eCourier,” 2007. [Online]. Available: http://www.ecourier.co.uk

James M. Kang received his Bachelor’s degree at Purdue University, IN, USA in 2000, and his Master’s degree at the Rochester Institute of Technology, NY, USA in 2004, all in computer science. He worked as a system analyst at Eastman Kodak Company from 2000 to 2004. He is currently working towards the Ph.D. degree in Computer Science at the University of Minnesota, USA. His research interests include spatial and spatio-temporal data mining, spatial and spatio-temporal databases, and continuous query processing.

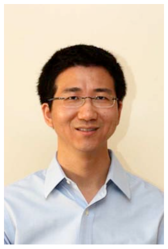
Mohamed F. Mokbel (Ph.D., Purdue University, 2005, MS, B.Sc., Alexandria University, 1999, 1996) is an assistant professor in the Department of Computer Science and Engineering, University of Minnesota. His main research interests focus on advancing the state of the art in the design and implementation of database engines to cope with the requirements of emerging applications (e.g., location-aware applications and sensor networks). Mohamed was the co-chair of the first and second workshops on privacy-aware location-based mobile services, PALMS, 2007 (Mannheim, Germany) and 2008 (Beijing, China). He is also the PC co-chair for the ACM SIGSPATIAL GIS 2008 and 2009 conferences. Mokbel has spent the summers of 2006 and 2008 as a visiting researcher at Hong Kong Polytechnic University and Microsoft Research, respectively. He is a member of ACM and IEEE.



Shashi Shekhar is a McKnight Distinguished University Professor at the University of Minnesota. For contributions to spatial databases, spatial data mining, and geographic information systems(GIS), he received IEEE-CS Technical Achievement Award and was elected an AAAS Fellow and an IEEE fellow. He co-authored a textbook on Spatial Databases, co-edited an Encyclopedia of GIS and served on the Mapping Science Committee of the National Research Council (National Academies). He is serving as a co-Editor-in-Chief of Geo-Informatica: An International Journal on Advances in Computer Sc. for GIS; and steering committee member of the ACM Intl. Conference on GIS.

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.



Tian Xia is a senior member of technical staff at Oracle Corporation. Before joining Oracle, he received the B.S. degree from University of Science and Technology of China in 2001, and the Ph.D. degree from Northeastern University, Boston in 2007, both in computer science. His research interests include spatial and spatio-temporal data management, database indexing and query optimization, and data privacy.

Donghui Zhang received his Ph.D. in 2002 from the University of California – Riverside. Since then, he has been working as an Assistant Professor in the College of Computer & Information Science, Northeastern University. Professor Zhang's primary research area is databases. In particular, indexing and query optimization in spatial, temporal, and spatiotemporal databases. Many real application data have spatial and/or temporal dimensions. For instance, the locations of apartment buildings, cars, mobile-phone users which may or may not change over time. The concern is how to index such objects and how to efficiently compute the result of interesting queries. Professor Zhang received the NSF CAREER Award: Fast Query Support for Emerging Spatial Database Applications. He has written five book chapters and published over thirty peer-refereed research papers. He has served on the panels of two NSF programs, on the Program Committees of various international conferences including VLDB'09, ICDE'07, VLDB'05, ICDE'04 and EDBT'04, and as referee for over 10 journals such as TODS and VLDBJ.

Authorized licensed use limited to: University of Minnesota. Downloaded on December 30, 2009 at 15:50 from IEEE Xplore. Restrictions apply.