Discovery of Influence Sets in Frequently Updated Databases *

Ioana Stanoi
IBM T.J. Watson
Research Center
irs@us.ibm.com

Mirek Riedewald Divyakant Agrawal Amr El Abbadi
Department of Computer Science
University of California Santa Barbara
{mirek,agrawal,amr@cs.ucsb.edu}

Abstract

An increasing number of organizations are currently working on ways to express and provide location information to services and applications. A location aware system knows the position of each component, and it is able to track devices through changes due to movement. In this context, data management issues such as efficient storage and retrieval of data through frequent updates pose new challenges. While we believe that spatial queries in general are going to gain in importance due to the emerging type of applications, we are particularly interested in the discovery of influence regions and influence sets around a query point. An influence set is formed by all points that have q as their nearest neighbor, and are located within the boundaries of an influence region. In this paper we propose for the first time a technique that reduces such a query to the more familiar nearest neighbor and range queries. These queries not only perform well in a dynamic environment, but also allow for their domain to be specified on demand. Additionally, the method we propose is based on already existing indexing and retrieval framework, thus facilitating integration with more complex location queries.

1 Introduction

"Ubiquitous computing spreads intelligence and connectivity to more or less everything. [$\cdots$] You name it, and someone, sooner or later, will put a chip in it." [Tha00]. Mobile data management in a ubiquitous computing context, although a relatively new area, offers challenging research problems with great market potential. The new set of applications built on top of the data management layer for mobile devices, require the development of new computation models and algorithms. Due to the inherent properties of mobility, an increasing number of these applications are location, and in general, context aware. A location aware system knows the location of each component, and is able to use this information to enhance the operation of the system. By "enhancement" we mean either providing, improving, or redirecting services. While research has focused on developing architectures for location aware systems, less attention has been paid to the fundamental and challenging problems from the data management perspective. Complexity arises at all levels of data management; in capturing, representing and processing contextual data. The framework that we consider to be appropriate for the data management of mobile devices has several reoccurring features. First, information about servers (also referred to as *sites*) is relatively static, with a very low frequency of updates. This information can be indexed and queried in traditional ways. Second, customers or mobile devices (which we will call "points") change location frequently, and the amount of precomputation based on their coordinates should be minimal.

We expect that due to the emergence of location based applications, spatial queries such as range, nearest neighbor and reverse nearest neighbor will play a fundamental role in the processing and analysis of location data. In most cases, query processing methods need to be reformulated to take into consideration requirements of new applications. While a number of solutions have been proposed for nearest neighbor queries, reverse nearest neighbors have no existent solution adaptable to the requirements of location based services. This makes the exploration of possible new methods a priority in defining an appropriate framework for indexing and querying the location of mobile devices.

Intuitively, a reverse nearest neighbor query is significant because it retrieves the *influence set* around the query point q. An influence set is formed by all points that have q as their nearest neighbor (NN),

*This research was partially supported by the NSF under grant numbers EIA98-18320, IIS98-17432 and IIS99-70700.

Proceedings of the 27th VLDB Conference, Roma, Italy, 2001

and are located within the boundaries of an *influence region*. It is interesting to note that, although the classes of NN and RNN [KM00] queries are directly related, solutions from the first type of queries cannot be directly applied to the latter. The difference comes from the fact that if a data point p is an $RNN(q)$ (q is the nearest neighbor of p), it does not imply that p is the nearest neighbor $NN(q)$ of q. When formalized in [KM00], reverse nearest neighbor queries were described as being either mono or bi-chromatic. The number of colors is dependent on the number of distinct types of data points. In the mono-chromatic case, all the points in the database are considered of the same category. By contrast, bi-chromatic queries assume that the data points involved in the computation are of two types, grouped as *points* and *sites*. For a query *site* q, an $RNN(q)$ query retrieves all the points that are closer to q than to any other site. Although mono-chromatic reverse nearest neighbor queries have a variety of applications, in the context of location aware services naturally bi-chromatic RNN queries have a wider range of applications. In this setting, queries inherently have two types of data: the providers (*sites*) and the consumers of services (*points*).

This paper approaches a problem that previously has not been analyzed in the context of dynamic databases, although it applies to an emerging class of applications based on location aware services. We believe that this paper introduces several contributions in understanding the class of reverse nearest neighbor queries and preparing it for emerging mobile applications. Most importantly:

- *we show how the process of discovering influence sets can be reduced to nearest neighbor and range queries.* Thus an RNN query can be answered efficiently using standard techniques, without the need to pre-compute additional information or to develop new index structures. We therefore avoid the storage and update overhead due to precomputation, an aspect especially important in the case of dynamic databases such as those keeping track of mobile devices. Also, simplifying RNN queries to a sequence of more familiar types of spatial queries, allows for straight-forward integration with complex queries.

- *RNN query definition such as the partitioning of the data into service providers and consumers now can be defined on-demand.* It is an important aspect, especially in mobile applications where the features between sets of devices can be sometimes interchanged.

The organization of this paper is the following: We start by reviewing in Section 2 the previous work on RNN queries and discuss its shortcomings when placed in a mobile context. Next, in Section 4, we introduce our solution for answering reverse nearest neighbor queries. Finally, an experimental evaluation in Section 5 is presented in support of our proposed algorithm. We conclude with a discussion and directions for future work in Section 6.

2 Background and Related Work

Most applications based on tracking mobile devices need to support the storage and retrieval of location related information. For these applications, there is the need for new efficient algorithms that perform well under the load of frequently updated databases. We believe that existing solutions for answering spatial queries should be revisited from the perspective of the evolving applications. In this context, one of the classes of queries that can prove to be fundamental is that of discovery of influence regions and sets with reverse nearest neighbors [KM00]. Unlike the related nearest neighbor $NN(q)$ queries that return the data points closest to a given query point q, $RNN(q)$ outputs the data points that have q as their nearest neighbor. We chose the distance metric to be Euclidean, although the algorithms we propose are not restricted to this assumption. Formally, if we let the database DB contain points $p \in \{p_1, p_2, \cdots p_i \cdots\}$, and the distance between two points p_i and p_j is $d(p_i, p_j)$, then:

- *nearest neighbor query* $NN(q) =$
  $\{p_i \in DB | \forall p \in (DB - \{p_i\}), d(q, p_i) \leq d(q, p)\}$.
  p_i is an answer to query $NN(q)$ if it is not further (according to a given distance metric) from q than any other points in database DB.

- *mono-chromatic reverse nearest neighbor query* $RNN(q)_{mono} =$
  $\{p_i \in DB | \forall p \in (DB - \{p_i\}), d(q, p_i) \leq d(p_i, p)\}$
  p_i is returned for query $RNN(q)$ only if the distance from p_i to q is not greater than any distance from p_i to another data point in DB.

For the purpose of studying spatial queries on mobile databases, our underlying assumption is that there are two types of data, namely *sites* and *points*. Let database DB_{sites} store the location of the *sites*, while database DB_{points} indexes the location of all the points. Note that in the context of bi-chromatic data it is straight forward to implement nearest neighbor methods designed for the mono-chromatic case, while RNN queries are dependent on the distinction between data sets.

- *bi-chromatic reverse nearest neighbor query* $RNN(q)_{bi} =$
  $\{p_i \in DB_{points} | \forall s \in DB_{sites}, d(q, p_i) \leq d(p_i, s)\}$.
  Given a query site q, $RNN(q)$ retrieves all data points in DB_{points} that are closer to q than to any other site in DB_{sites}.

Note that, for the remainder of this paper, we will use the notation $RNN(q)$ to mean $RNN(q)_{bi}$ unless specified otherwise.

The only previous solution that pertains to bi-chromatic RNN queries was introduced by [KM00], together with the problem formulation. It is based on the fact that given the data sets DB_{sites} and DB_{points}, one can precompute nearest neighbor distances and store them in an $RNN-tree$. For each data point $p \in DB_{points}$, the $RNN-tree$ stores the description of the nearest neighbor circle by its radius $(p, NN(p))$, where $p \in DB_{points}$ and $NN(p) \in DB_{sites}$. For a query site $q \in DB_{sites}$, $RNN(q)$ retrieves points $p \in DB_{points}$ such that q is inside the nearest neighbor circle of p. Reverse nearest neighbor queries are then answered by point enclosure queries in the RNN-tree.

Our main interest lies in providing algorithms to help construct applications for mobile devices. For static databases, precomputing the nearest neighbor is an efficient method. However, when applied to the very dynamic databases found in mobile applications, this type of solution has two inherent constraints. First, the domain of RNN queries has to be pre-defined. Especially in the poly-chromatic case, we would like to allow queries to identify on demand the data sets they access. Second, in the case of dynamic databases, updates become fairly expensive. To support updates, [KM00] introduce a nearest neighbor tree $NN-tree$ in addition to the $RNN-tree$. The assumption is that the data management framework supports both point as well as spatial data. Inserting a point p' leads to the modification of the nearest neighbor circles for all points $p = RNN(p')$ as well as the computation and insertion of the new nearest neighbor circle $(p', NN(p'))$ in the $RNN-tree$. Both the insertion and the deletion of an element involve searches and modification in both indexing trees $NN-tree$ and $RNN-tree$.

The only other published solutions for the class of reverse nearest neighbor queries referred to mono-chromatic queries [SAA00]. Recall that in the mono-chromatic case, there is no distinction between data sets, as for example between *points* and *sites*. In this framework, the work of [SAA00], takes advantage of the geometrical properties of reverse nearest neighbors in the data space. The underlying observation is that, for a given dimensionality of the data space, there is an upper bound on the number of points to be returned by the $RNN(q)$ query. The algorithm in [SAA00] partitions the space around query point q, finds a nearest neighbor in each region and then verifies if the candidates are the correct reverse nearest neighbors. The advantage of this type of approach is that it relies on the existing indexing structure, a standard R-tree, and does not require any additional data structures specifically for RNN queries. Our goal is to provide the same benefits from using the existing database framework, but in the context of a bi-chromatic class of RNN queries. For our purpose, this method specifically proposed for mono-chromatic RNN queries, cannot accommodate the bi-chromatic case.

3 Preliminaries

Our goal is to add efficient RNN capability without imposing any update overhead to the databases. We present a first method that reduces the retrieval of influence sets to the more thoroughly studied and already optimized nearest neighbor and range queries. Since our method does not rely on data structures additional to the existing indexing and support for nearest neighbor and range queries, RNN queries do not introduce update overhead.

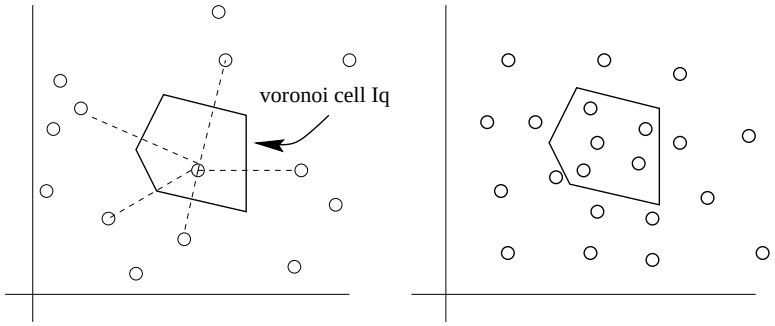
Given a set DB_{sites} of sites, a set DB_{points} of points, and a query site q, $RNN(q)$ finds all points that have q as their nearest neighbor site. In order to implement a fast method for answering bi-chromatic RNN queries with no pre-computation, we can divide the problem into two phases, and improve their computation/I/O cost accordingly. Note however that the distinction between these two phases is done mostly to clarify the requirements, and it will be modified slightly during the presentation of the algorithm development. For a given query site q, the first phase involves the processing of information on the location of sites from DB_{sites}, and the computation of the region of influence I_q around q. I_q is defined to have the property that any point within its boundaries is closer to q than to any other site s in DB_{sites}. Then any point p in DB_{points} that is constrained by region I_q (calculated based on the data in DB_{sites}) must be closer to q than to any other s in DB_{sites} (Figure 1(a)). The on-the-fly computation of I_q can be very expensive if all the distances between points and sites are compared; a practical solution should be able to identify the influence region based only a subset of the data.

The second phase retrieves points in the influence set, i.e., points in DB_{points} that are within the range of region I_q (Figure 1(b)). Since the latter is a straight forward range query, the difficulty is in defining the constraints of the influence region.

The definition of the influence region I_q is equivalent by construction to that of Voronoi diagrams [BKOS00]. The Voronoi Diagram of a collection of sites is the partition of space into cells, each of which consists of the points closer to one particular site than to any others.

Definition 1 *For any site $s \in DB_1$ and a given distance function $d()$ the Voronoi cell of order 1 of s is defined as $\{p \in DB_2 | \forall (s' \in DB_1 - \{s\} : d(p,s) \leq d(p,s')\}$.*

In the field of data management, the properties of Voronoi cells have recently received more attention.



(a) phase 1: compute influence region Iq in DBsites

(b) phase 2: retrieve points in DBpoints based on Iq

Figure 1: Answering RNN Queries in Two Phases

In [BKKS98] it was proposed to store approximations of Voronoi cells in an indexing Vornoi-tree in order to support fast nearest neighbor queries. The significance of a Voronoi cell in this case is that, if the query point q falls within a point p's Voronoi cell, then q is closer to p than to any other point and therefore p is a nearest neighbor. The key to the solution proposed by [BKKS98] is that it uses *approximations* to Voronoi cells, i.e., minimum bounding boxes as an upper bound to the Voronoi cells. A first argument against using this type of approach for RNN queries is that, to exploit its benefits, the query site q is constrained to already indexed points in DB_sites and in the Voronoi-tree. Second, while an approximation is sufficient to answer nearest neighbor queries, answering reverse nearest neighbor queries requires information about the exact constraints of the Voronoi cell. Unlike the nearest neighbor queries, comparison is then needed between distances from all points in DB_{points} inside the approximation with all the sites in DB_{sites}. Additionally, in cases such as location aware applications tracking mobile devices, updates are frequent. The overhead of maintaining the pre-computed Voronoi diagram in the form of a Voronoi-tree becomes more critical.

The discovery of influence sets requires a method to retrieve *all* information necessary in defining an influence region (or Voronoi cell), without pre-computation (to avoid high maintenance costs). More, the retrieval of this information should be efficient. We next give the details of such a method.

4 The *Approximate, Refine and Filter* Method

Recall that, for clarity in the presentation of the algorithm's development, we divided the method for answering poly-chromatic $RNN(q)$ queries into two phases:

1. First, given a set of sites in the database DB_{sites} and a query point q, we calculate the boundaries of the influence region I_q.
2. Following the computation of the influence region, we retrieve all the points in the database DB_{points} that lie within the range of I_q. These points form the influence set of q.

Since the second phase of the algorithm is simply a range query, we will concentrate on defining the boundaries of an influence region I_q. Computing the boundaries of influence region I_q by comparing the location of q with that of *all the sites* in DB_{sites} is impractical. The way to reduce this cost is to make use of approximations as a first step in retrieving the exact boundaries of region I_q, without accessing all the data in DB_{sites}. For a correct and efficient computation of the influence region, we further subdivide this phase into the following:

- Compute $Approx(I_q)$, a first approximation to I_q, based only on the location of a small subset of sites from DB_{sites}
- Refine the approximation by inferring boundaries on the location of all the sites needed to define the exact I_q. Based on this approximation to the influence region I_q, $Approx(I_q)$, we can define the constraints of a region $Refine(I_q)$ such that it includes all the sites in DB_{sites} needed to improve the first approximation to I_q.

Compute Approximation to I_q, $Approx(I_q)$
A first approximation $Approx(I_q)$ can generally be computed by considering only the location of query site q and a subset of the sites in DB_{sites}. We choose this subset to be the nearest neighbors in each quadrant formed by axes centered at q and parallel with the original axes. This selection of sites is semi-arbitrary. The sites in DB_{sites} found by performing simultaneous NN queries [BEKS00, BBK00] in all quadrants, together with the query site, are then used to compute $Approx(I_q)$. Since the nearest neighbors are closer to the site q than other sites, there is a considerable chance that they indeed restrict the exact influence region I_q.

Finding the boundaries of $Approx(I_q)$ is a necessary step, but not sufficient. If the second phase of the algorithm uses the approximation to the Voronoi cell rather than the exact constraints, then the RNN query returns a superset of the reverse nearest neighbors of q. Consequently, some of the data points in the answer to $RNN(q)$ may be closer to a site in DB_{sites} than to q. In this case ,without computing the exact boundaries of I_q, a range query based on $Approx(I_q)$ can return an incorrect answer.

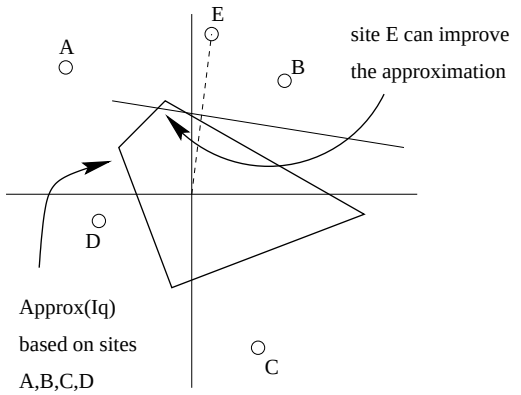


Figure 2: Point E Can Improve the First Approximation of the Voronoi Cell

Example: Consider the subset of the database DB_{sites}, with sites $\{A, B, C, D, E\}$ in Figure 2. If we divide the space around query site q into quadrants, the computation of the Voronoi cell based on a nearest neighbor from each quadrant ,($\{A, B, C, D\}$), defines a region $Approx(I_q)$ that includes and is larger than the exact influence region I_q. The boundaries of this approximation to the influence region are easily obtained by calculating the intersection of the bisectors of $|qA|$, $|qB|$, $|qC|$, $|qD|$. Since $Approx(I_q)$ is calculated using only a *subset* of the constraints that define the exact I_q, it spans over a *superset* of the points in DB_{points}. If in the second phase of the RNN algorithm we perform a range query with the boundaries of $Approx(I_q)$ on data in DB_{points}, the set of points returned is a superset of the correct answer. To filter out the points within the boundaries of $Approx(I_q)$ that are not in I_q, we need to compare these points with all the data points in DB_{sites}. Site E for example contributes to bounding the exact region I_q, and a point in $Approx(I_q)$ that is closest to E but included in $Approx(I_q)$ is an incorrect answer to $RNN(q)$.

Defining the constraints of $Approx(I_q)$ is only a first step in the computation of the exact influence region I_q.

Refine Approximation to Influence Region

Based on the coordinates of $Approx(I_q)$, it is possible to efficiently obtain the boundaries of a region $Refine(I_q)$ around the query site q, such that it contains all data points in DB_{sites} that further constrain the influence region I_q (Figure 2). A range query based on $Refine(I_q)$ and applied to data set DB_{sites}, thus enables the computation of the exact I_q. The goal is to compute I_q based only on a subset of points in DB_{sites}, which significantly improves the performance of retrieving reverse nearest neighbors.

Assume for now that the computation of a correct $Refine(I_q)$ is defined. Since it is a challenging problem in itself, we will present it in detail in a separate section. The retrieval of points in $Refine(I_q)$ facilitates the computation of the exact boundaries of I_q and a correct range query in DB_{points} over the set of reverse nearest neighbors (Figure 3,4).

Notation:

q = query site

I_q = influence region around q

$Approx(I_q)$ = a fist approximation to influence region I_q, based on nearest neighbors in DB_{sites}

$Refine(I_q)$ = region containing all points in DB_{sites} used to refine the approximation to I_q

Phase I:

1. find $Approx(I_q)$ based on a set of nearest neighbors;
2. using the coordinates of $Approx(I_q)$, find $Refine(I_q)$;
3. perform range query $Refine(I_q)$ in DB_{sites} and output the respective data points in S;
4. based on the location of the q and the sites in P, compute $I_q = exact$ Voronoi cell around the query site;

Figure 3: Calculate Voronoi Cell around q

Phase 2:

1. $answer$ = retrieve range query constrained by I_q, in DB_{points};

Figure 4: Retrieval of Influence Set

Although it answers correctly, the retrieval of points in the exact Voronoi cell of q may involve a complex range query. Once there is a method to calculate $Refine(I_q)$, the actual refinement of the influence region can be done either before or after the range query over DB_{points}. On one hand, the algorithm can compute the exact I_q, and then retrieve the exact influence set from DB_{points}. On the other hand, the refinement step can be delayed. If the influence region is based on the approximation to I_q, $Refine(I_q)$ is used as a filtering step to check the correctness of the influence set. Note that in this case, the filtering step requires comparison between distances from points from DB_{points} in the first approximation to the influence set and the sites contained within $Refine(I_q)$ in DB_{sites}. In the performance section we choose to evaluate the second

version of the algorithm, since it involves simple rectangular range queries (Figure 5, 6).

Phase I:
1. find $Approx(I_q)$;
2. calculate the boundaries of $Refine(I_q)$;
3. perform range query in DB_{sites} based on $Refine(I_q)$, and store the result in set S.

Figure 5: Calculate Boundaries of Influence Region

Phase 2:
1. retrieve range query based on the minimum bounding rectangle of $Approx(I_q)$ in DB_{points}, store in P;
2. $answer$ = filter out incorrect reverse nearest neighbors based on S and P;

Figure 6: Retrieval of Influence Set

In order to give a clear overview of the algorithms, we delayed the details of computing $Refine(I_q)$. Next, we present a method to define ($Refine(I_q)$, whose correct computation plays a crucial role in the solution we proposed.

4.1 Computing the boundaries of $Refine(I_q)$

Recall that, for a query site q, $Refine(I_q)$ is the region containing all sites in DB_{sites} that constrain the Voronoi cell I_q around q. The boundaries of $Refine(I_q)$ should be calculated efficiently, based on a subset of DB_{sites} rather than based on a comparison of all distances between q and sites in DB_{sites}. A site s constrains I_q iff computing I_q with and without s in the database leads to different results. Our goal is to define region $Refine(I_q)$, given only the vertices of $Approx(I_q)$ and query site q. In general, the vertices of $Approx(I_q)$ are computed as the intersection of the bisectors of segments $|q, NN_i|$ where NN_i is a quadrant nearest neighbor of q. To refine the first approximation to the exact influence region I_q, we want to find the region $Refine(I_q)$ that contains all the sites s such that the bisector of $|qs|$ intersects an edge of $Approx(I_q)$.

Consider a set of sites in database DB_{sites}, and a query site q (Figure 7). The approximation $Approx(I_q)$ of the Voronoi cell of q, polygon $[m, n, o, p]$, is obtained by retrieving nearest neighbor points $\{A, B, C, D\}$. Region $Refine(I_q)$ then includes all the points whose bisectors with q can intersect any of the lines $|m, n|, |n, o|, |o, p|, |p, m|$.

For notation purposes, we refer to the middle of the segment between q and a site s, $|q, s|$, as $mid|q, s|$. In general, the bisectors of a segment $|q, NN_i|$ (NN_i = quadrant nearest neighbor) through a vertex v of $Approxi(I_q)$ is by definition perpendicular to $|q, NN_i|$. Let the middle point of segment $|q, NN_i|$ be $mid|q, NN_i|$. Then segments $|q, mid|q, NN_i||$, $|mid|q, NN_i|, v|$ and $|q, v|$ form a right triangle circumscribed in a circle with diameter $|q, v|$. Let $MBR_{circles}$ be the minimum bounding rectangle containing all circles with diameters $|q, v|$ for all vertices v of $Approx(I_q)$. Going back to our example, $|q, c|$ is by definition perpendicular to $|c, o|$ and $|q, b|$ is perpendicular to $|b, o|$. They form two right triangles $[c, q, o]$ and $[b, q, o]$ that are circumscribed in a circle of diameter $|q, o|$. In Figure 8 we show all circumscribing circles corresponding to $|q, m|, |q, n|, |q, o|, |q, p|$ and their minimum bounding rectangle $MBR_{circles}$.

Lemma 1 *Let s be a site such that $mid|q, s|$ lies outside $MBR_{circles}$. Then s does not constrain the Voronoi cell I_q of q.*

Assume by contradiction that s does constrain I_q, i.e., the bisector of $|q, s|$ intersects the edges of $Approx(I_q)$. Then the bisector must also intersect one of the diameters $|q, m|, |q, n|, |q, o|, |q, p|$. Without loss of generality, assume that the intersected diameter be $|q, o|$, the point of intersection is o' and the circle with diameter $|q, o|$ is C. Then points $mid|q, s|, q$ and o' form a right triangle whose circumscribing circle C' has a diameter $|q, o'|$. Since o' is a point between q and o on $|q, o|$, then C' must be contained in C. Consequently, $mid|q, s|$ lies inside of C, which contradicts our assumption.

The above argument can be easily generalized for any $MBR_{circles}$ for a region $Approx(I_q)$. The proof of correctness is only a generalization of the proof above, taking in consideration all circumscribing circles of diameter $|q, v|$ (where v is a vertex of $Approx(I_q)$).

We showed that, for any site $s \in DB_1$ that contributes to the computation of the Voronoi cell around q, $mid|q, s|$ must be included in one of the circumscribing circles C_i. Let the coordinates of C_i's center be c_ix and c_iy. A minimum bounding rectangle $MBR_{circles}$ that contains all the circumscribing circles C_i with diameter $|q, v_i|$ and radius $r_i = d(q, v_i)/2$ has coordinates $low_x, high_x, low_y, high_y$, where

$low_x = min((c_1x - r_1), (c_2x - r_2), \cdots (c_ix - r_i), \cdots)$,
$high_x = max((c_1x + r_1), (c_2x + r_2), \cdots (c_ix + r_i), \cdots)$,
$low_y = min((c_1y - r_1), (c_2y - r_2), \cdots (c_iy - r_i), \cdots)$,
$high_y = max((c_1y + r_1), (c_2y + r_2), \cdots (c_iy + r_i), \cdots)$.

Recall that our goal was to define $Refine(I_q)$, the location of all sites s that can contribute to the Voronoi cell of q. If $mid|q, s|$ is within $MBR_{circles}$, site s can only be twice as far from q. Since we already defined $MBR_{circles}$, it is now straight forward to calculate the coordinates of $Refine(I_q)$: $2 \times low_x - q_x, 2 \times high_x - q_x, 2 \times low_y - q_y, 2 \times high_y - q_y$. That is, the distance from the vertices of $Refine(I_q)$ to q is double the distance between q and the edges of the minimum bounding rectangle that includes all the circumscribing circles C_i (Figure 8). In general, the steps to compute $Refine(I_q)$ and MBR_Vmax are shown in Figure 9. There are a few special cases, for a query site q that is

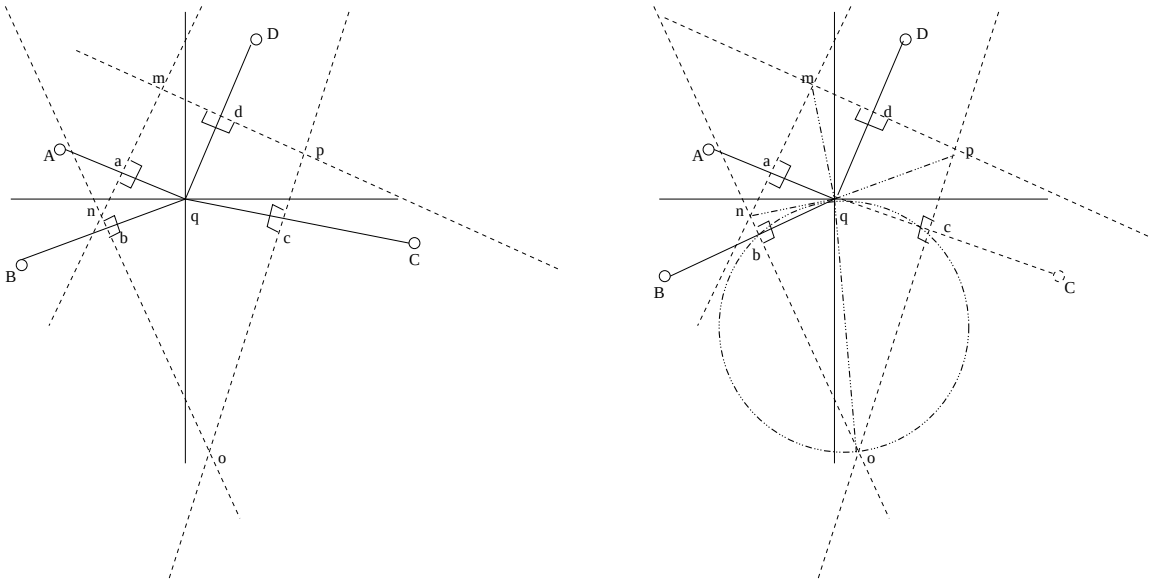


Figure 7: Example of Circumscribing Circle

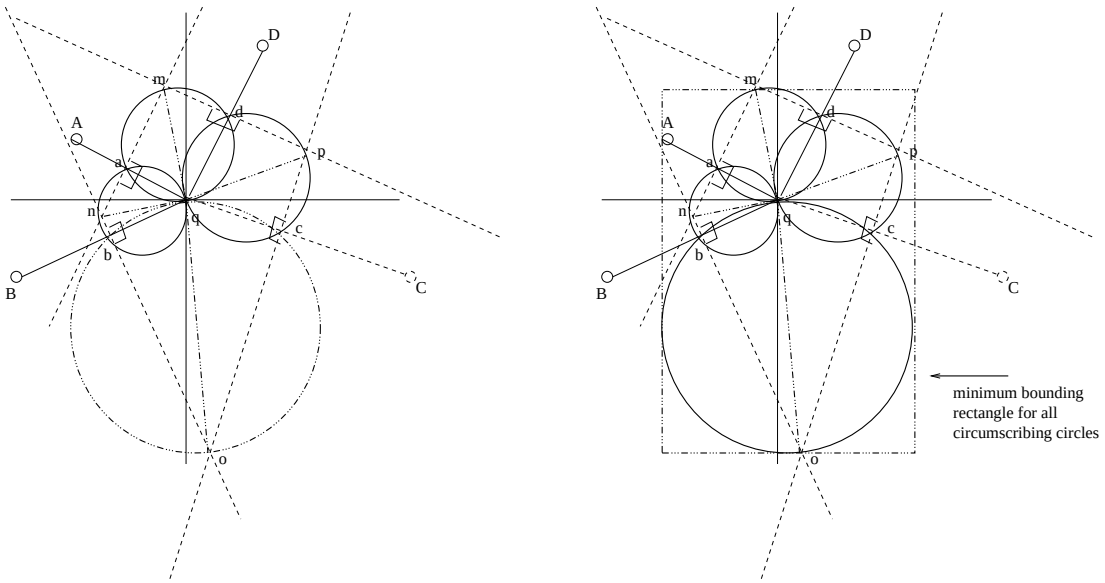


Figure 8: Find the Boundaries of $Refine(I_q)$

close to the boundary of the data space. One such case is when there are no points in a quadrant, and therefore a nearest neighbor cannot be found. Then the data space boundaries of the corresponding quadrant will be considered as bisectors, and the Voronoi cell is computed accordingly. Another special case is if the minimum bounding rectangle of the region $Refine(I_q)$ is greater in a dimension d than the limit of the data space; then for dimension d, $Refine(I_q)$ be restricted by the data space.

1. find nearest neighbors $\{NN_1, NN_2, \cdots NN_k\}$ in each quadrant
2. calculate approximation to I_q, $Approx(I_q) = VoronoiCell(q, NN_1, NN_2, \cdots NN_k)$
3. define $C_1, \cdots C_i, \cdots C_n$ as the circumscribing circles centered at the middle of each segment between q and the vertices $v_1, \cdots v_i, \cdots v_n$ of $Approx(I_q)$. The radius r_i of a circumscribing circle C_i is the distance from its center to q, $r_i = d(v_i, q)/2$.
4. calculate boundaries of minimum bounding rectangle around all circumscribing circles
5. $Refine(I_q) = (2 \times low_x - q_x, 2 \times high_x - q_x, 2 \times low_y - q_y, 2 \times high_y - q_y)$

Figure 9: Calculating the Boundaries of $Refine(I_q)$

5 Experimental Results

In the following, we describe several experiments performed to test the efficiency of the proposed RNN method. First, we compared our algorithm for RNN queries with a brute-force approach. Although the results are intuitive, this comparison is relevant because there is no other existing algorithm based on on-the-fly computation. Second, we implemented and compared with the method of [KM00]. Finally, we show the cost distribution of the different components of our algorithm.

In our experiments we used the Montgomery County data set representing the longitude/ latitude of road crossings. From the initial database of 60,000 points, we created two data sets to represent DB_{sites} and DB_{points}, and indexed them in an R*-tree. We varied the size ratio between DB_{sites} and DB_{points} in order to better analyze a wide variety of applications. The 60000 point road crossing set was divided into two data sets whose relative sizes are varied between 1/5 (for a ratio of 10000 sites to 50000 points), 2/4, 3/3, 4/2, and 5/1. In each of these cases, we ran 5000 RNN queries, where query sites were randomly chosen from DB_{sites}. For the nearest neighbor method incorporated into our RNN algorithm, we used the solution proposed by [HS95] that was proven by [BBKK97] to be optimal.

A naive attempt to answer $RNN(q)$ queries with no pre-computation is to compare, for each point p_i in DB_{points} its distance to all sites s, $d(p_i, s)$, versus its distance $d(p_i, q)$ to the query site q. If, as a result of the comparison, data point p_i is the closest to q ($d(p_i, q) \leq d(p_i, s)$), then $p_i \in RNN(q)$. However, it is easy to note that if the size of the data in DB_{sites} is n_1 and the size of the DB_{points} is n_2, then the complexity of the brute force approach is that of computing nearest neighbors for each point in DB_{points}, i.e., $O(n_2 \times log(n_1))$. By contrast, our algorithm incurs the cost of several nearest neighbor queries as well as range queries in both DB_{sites} and DB_{points} In order to make the comparison independent of cache size and to give the naive algorithm the best possible setup, we assumed that leaf pages can be stored in a cache after being read once. However, we did not make the same assumption for our algorithm. As expected, the results of our experiments also showed that the method we proposed greatly outperforms the naive approach. The average number of leaf page accessed by queries based on the brute-force approach was 240 pages, while our method accessed only 5 to 6 leaf pages on average. While in most cases, our method takes advantage of the proximity of the nearest neighbors retrieved, the naive approach accesses all pages in DB_{points} and most pages in DB_{sites}.

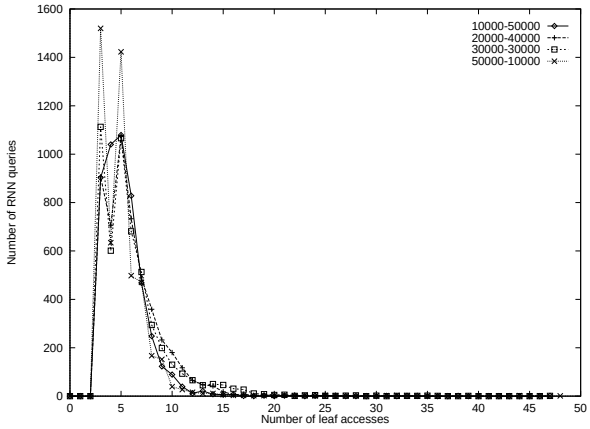


Figure 10: Number of Leaf Accesses for Different Data Set Sizes, Queries Based on Our Algorithm

We also compared our method with the solution proposed in [KM00].The distribution of its query cost is shown in Figure 11 Note that most queries access around 5 pages, similarly to the results obtained by using our method without pre-computation (Figure 10). The cost of updates using our technique is that of a single insert into the underlying indexing tree. Note however that RNN queries do not impose any update overhead with respect to the already existing data management cost. If reverse nearest neighbors are facilitated by pre-computed results stored in an $RNN - tree$, then an insert update requires the following sequence of steps:

1. query of the $RNN - tree$. The cost of such a query is equivalent to that shown in Figure 11

2. changing of all affected radii in $RNN-tree$. Considering a lower bound on the update overhead, this step might access pages that are already in the cache and do not cause additional I/O.
3. query in NN-tree. This step is comparable with that of updating the underlying indexing tree if using our method.
4. insert of a point into NN-tree.
5. insert of a circle into RNN-tree.

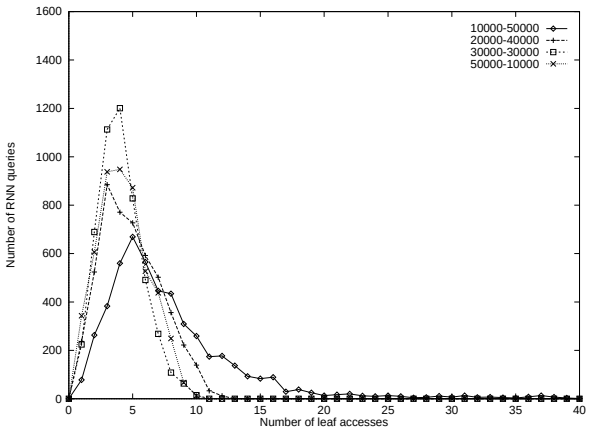


Figure 11: Number of Leaf Accesses for Different Data Set Sizes, Queries Based on [KM00]

As a result, an advantage of using our method is that we do not need to query the RNN tree during updates. Additionally, we only need to store point data. By comparison, precomputed NN circles are stored as spatial objects, which leads to more page overlap and /or more complex index structures. Finally, an important advantage of our method is that the size of the data set does not influence the cost of RNN queries; what matters is the distribution of points into pages. Since the algorithm we propose pertains mostly to 2-dimensional data (for location aware applications), finding quadrant nearest neighbor, or defining $Refine(I_q)$, will usually access a single page. Although comparable query performance, our method has the advantage of eliminating update overhead.

In Figure 12 we illustrate the cost of the various components of our RNN method. Note that the x-axes represent the size of DB_{sites} only (the size of DB_{points} can be inferred), while the y-axes corresponds to the number of leaf page accesses of the R*-tree. The results were obtained for the same set of 5000 queries. We show the cumulative number of page accesses at every step of the algorithm: following the retrieval of the quadrant nearest neighbors, following the computation of the first approximation $Approx(I_q)$, and the total number of page accesses necessary for the range query $Refine(I_q)$. The overall cost is not greater than a total of 5 to 6 page accesses. It is interesting to note that although the cost of nearest neighbor queries involved is lower than that of the two range queries, the difference is not significant. As expected, nearest neighbor queries access on average one to two leaf pages. The number of page accesses is very low because the boundaries of the range query in most cases falls within the boundaries of a few leaf pages. The algorithm is shown to perform very well in finding a good approximation to the exact Voronoi cells.

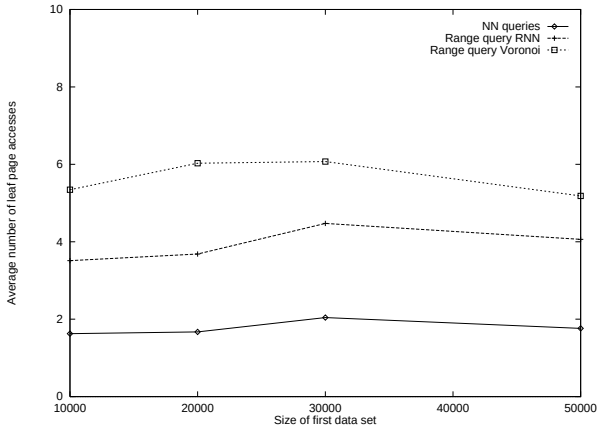


Figure 12: The Costs of Various Components in the Proposed Method

6 Conclusion

Emerging applications necessitate increasing sophistication in ways to express and provide location information. Being able to track moving objects introduces new challenges for data management that should be able to support frequent updates as well as fast query processing. In this paper we described a first method to compute on demand influence sets around a query site, by reducing them to nearest neighbor and range queries. Our approach also facilitates integration with more complex location queries, using the already exiting indexing and retrieval framework. Since it does not create any additional data structures for the specific support of RNN queries, the method we propose does not impose any overhead with respect to database maintenance.

As spatial queries in general are going to gain in importance due to the increasing market for location aware services, we plan to analyze in more depth the modifications that should be brought not only to spatial queries, but also to the appropriate indexing structures. Additionally, we plan to analyze the behavior of RNN queries for applications that require higher dimensional data. In this case, an algorithm should make use of approximate nearest neighbors and avoid the cost [BBK$^+$01] of high-dimensional nearest neighbor queries.

7 Acknowledgements

The authors of this paper would like to thank Costin Iancu and Christian Lang for their helpful comments and patience.

References

[BBK00] C. Bohm, B. Braunmuller, and H. P. Kriegel. The pruning power: theory and heuristics for mining databases with multiple k-nearest neighbor queries. In *Proceedings of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK)*, 2000.

[BBK+01] S. Berchtold, C. Bohm, D. A. Keim, F. Krebs, and H. P. Kriegel. On optimizing nearest neighbor queries in high-dimensional data spaces. In *Proceedings of 8th International Conference on Database Theory (ICDT)*, 2001.

[BBKK97] S. Berchtold, C. Bohm, D. A. Keim, and H. P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proceedings of the ACM PODS Conference*, 1997.

[BEKS00] B. Braunmuller, M. Ester, H. P. Kriegel, and J. Sander. Efficiently supporting multiple similarity queries for mining in metric databases. In *Proceedings of the 16th International Conference on Data Engineering (ICDE)*, 2000.

[BKKS98] S. Berchtold, D. A. Keim, H. P. Kriegel, and T. Seidl. Fast nearest neighbor seearch in high-dimensional data space. In *Proceedings of 14th International Conference on Data Engineering (ICDE)*, 1998.

[BKOS00] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. Computational geometry: Algorithms and applications. In *Springer-Verlag*, 2000.

[HS95] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Proceedings of the 4th International Symposium on Large Spatial Databases*, 1995.

[KM00] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2000.

[SAA00] I. Stanoi, D. Agrawal, and A. El Abbadi. Reverse nearest neighbor queries for dynamic databases. In *Proceedings of the ACM SIGMOD Workshop on Data Mining and Knowledge Discovery (DMKD)*, 2000.

[Tha00] John Thackara. Doors of perception. In *www.doorsofperception.com and CHI2000*, April 2000.

[WJ96] D. A. White and R. Jain. Similarity indexing: algorithms and performance. In *Proceedings of Storage and Retrieval for Image and Video Databases (SPIE)*, 1996.