On Reverse Nearest Neighbor Queries

(extended abstract)

Anil Maheshwari* Jan Vahrenhold† Norbert Zeh‡

Abstract

In this paper, we discuss static and dynamic data structures for answering *reverse nearest neighbor* queries in two dimensions.

1 Introduction

A well-know proximity concept used, e.g., in clustering applications is the notion of a point's nearest neighbor according to some given metric. The problem of finding nearest neighbors, formally defined below, has received considerable attention in the past, and a variety of algorithms have been devised for this problem and its variants—see the recent survey by Smid [7].

Problem 1 (Nearest Neighbor) *Given a set $\mathcal{S}$ of n points in $\mathbb{R}^d$, a distance metric $\mathbf{d}$, and a query point p in $\mathbb{R}^d$, report a point $q \in \mathcal{S}$, for which $\mathbf{d}(p,q) = \min\{\mathbf{d}(p,r) \mid r \in \mathcal{S}\}$.*

The property of being a point's nearest neighbor, however, is not symmetric: while a point p can be the nearest neighbor of a point q, the reverse is not necessarily true. This asymmetry has led to defining the so-called *reverse nearest neighbor* problem [4], which, given a point p, asks for all points for which p is the nearest neighbor.

*School of Computer Science, Carleton University, Ottawa, Canada. Email: `maheshwa@scs.carleton.ca`.

†Westfälische Wilhelms-Universität Münster, Institut für Informatik, 48149 Münster, Germany. Email: `jan@math.uni-muenster.de`. Part of this work was done while on leave at the University for Health Informatics and Technology Tyrol, 6020 Innsbruck, Austria.

‡School of Computer Science, Carleton University, Ottawa, Canada. Email: `nzeh@scs.carleton.ca`.

Problem 2 (Reverse Nearest Neighbors) *Given a set $\mathcal{S}$ of n points in $\mathbb{R}^d$, a distance metric $\mathbf{d}$, and a query point p in $\mathbb{R}^d$, report all points $q \in \mathcal{S}$, for which $\mathbf{d}(q,p) = \min\{\mathbf{d}(q,r) \mid r \in \mathcal{S} \cup \{p\},\ r \neq q\}$. (see Figure 1).*

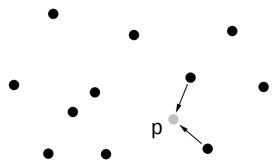

Figure 1: Reverse nearest neighbors for p.

Motivation and Related Work: The *Reverse Nearest Neighbor* problem (Problem 2) has been introduced in the database setting by Korn and Muthukrishnan [4] along with several applications, e.g., in decision support systems, continuous referral systems, or profile based marketing. Korn and Muthukrishnan presented static and dynamic solutions and also discussed a bichromatic setting: each point has one out of two possible colors, and reverse nearest neighbors are only sought among points of the other color. In this paper, we only consider the monochromatic problem. The algorithm of [4] performs a preprocessing step to solve the *All Nearest Neighbor* problem for $\mathcal{S}$, that is, to determine for each point in $\mathcal{S}$ its nearest neighbor. Each point q and its nearest neighbor r define a so-called $\mathcal{NN}$-ball centered at q with radius $\mathbf{d}(q,r)$. Then, all $\mathcal{NN}$-balls are stored in a data structure that can be used to report, given a query point p, all $\mathcal{NN}$-balls containing p. It is easy to verify that the points corresponding to the $\mathcal{NN}$-balls that contain p

are exactly the points having p as their nearest neighbor in $\mathcal{S} \cup \{q\}$.

Korn and Muthukrishnan approximate each $\mathcal{NN}$-ball by its axis-aligned bounding box and then use an R-tree [3] to maintain these approximations. Insertions and deletions can be handled in logarithmic time, but the worst-case complexity for point-containment queries is linear—even if no containing boxes are reported. Moreover, approximating the $\mathcal{NN}$-balls by their bounding boxes can lead to wrong answers (unless appropriate postprocessing is done).

Our Results: We present efficient data structures for the static and dynamic version of the reverse nearest neighbor problem in two dimensions that avoid using bounding boxes. The static data structure for a set of n points uses linear space, can be constructed in $\mathcal{O}(n \log n)$ time, and can be used to answer a reverse nearest neighbor query in $\mathcal{O}(\log n)$ time. The dynamic data structure requires $\mathcal{O}(n \log n)$ space, it can be updated $\mathcal{O}(Q(n) + \log^2 n \log\log n)$ time, and a query can be answered in $\mathcal{O}(\log^2 n \log\log n)$ time. Here, the quantity $Q(n)$ denotes the complexity of adding or removing a point to resp. from $\mathcal{S}$ while at the same time maintaining for each point in the $\mathcal{S}$ the *distance* to its nearest neighbor using only $\mathcal{O}(n \log n)$ space.

2 The static case

Definition 1 *For a set $\mathcal{S}$ of points in the plane and any point $p \in \mathcal{S}$, let its $\mathcal{NN}$-disc $D(p, N(p))$ be the disc centered at p with radius $N(p) := \mathbf{d}(p, p')$, where p' is the nearest neighbor of p with respect to $\mathcal{S} \setminus \{p\}$.*

The main observation in [4] is that the points in $\mathcal{RNN}(q)$ are exactly those points $p \in \mathcal{S}$ for which $q \in D(p, N(p))$. Hence, to answer a reverse nearest neighbor query, it is sufficient to find all $\mathcal{NN}$-discs containing q.

Observation 1 *Empty disc property: For any point $p \in \mathcal{S}$, the $\mathcal{NN}$-disc $D(p, N(p))$ does not contain any other point of $\mathcal{S}$.*

In the following, we will discuss the arrangement $\mathcal{A}$ of the set of n $\mathcal{NN}$-discs. The complexity of an arrangement of discs in two dimensions is the total number of its 0-dimensional faces (vertices) and 1-dimensional faces (circular arcs). Although, in general, the complexity of an arrangement of n discs can be $\Theta(n^2)$ [2], we show in the following lemma that the complexity of the arrangement $\mathcal{A}$ of the $\mathcal{NN}$-discs in our setting is only $\mathcal{O}(n)$.

Lemma 1 *The complexity of the arrangement $\mathcal{A}$ of all $\mathcal{NN}$-discs for a set $\mathcal{S}$ of n points is $\mathcal{O}(n)$.*

If we store for each 2-dimensional face of the arrangement $\mathcal{A}$ the $\mathcal{NN}$-disc(s) overlapping it, we can answer a reverse nearest neighbor query for a point q by simply performing a point location query on $\mathcal{A}$. The answer to the reverse nearest neighbor query then consists of the points whose $\mathcal{NN}$-discs overlap (the face containing) q.

Next we show how the arrangement $\mathcal{A}$ can be constructed and at the same time preprocessed to efficiently support point location queries. The main ingredient of our algorithm is a modification of the well-known line segment intersection algorithm by Bentley and Ottmann [1]. The original plane-sweep algorithm maintains all segments intersecting the sweep-line ordered by the y-coordinate of their intersection point. Intersections are detected by comparing neighboring segments in this y-structure, and the overall complexity of this algorithm is $\mathcal{O}((n + k) \log n)$ where k is the number of intersections.

Our algorithm implicitly computes the arrangement $\mathcal{A}$ by also sweeping the plane by a vertical sweep line from left to right, by increasing x-coordinate, but additionally maintaining the circles intersecting the sweep-line in a persistent y-structure [6]. The event points are the extremal left and right end points of each disc as well as points corresponding to intersection points of pair of discs. More specifically, we split each circle into an upper and a lower half-circle such that the resulting circular arcs are monotone with respect to the sweeping direction. At each extremal left or right point of a circle we insert resp. delete the two half-circles into resp. from the y-structure, and at each intersection point of two half-circles, we exchange the intersecting circular arcs in the y-structure.

From Lemma 1 we know that in all there are $\mathcal{O}(n)$ event points. Since the circular arcs defining the boundary of cells can be expressed by a simple second order function, it is easy to maintain the order of circular arcs along the sweep line as well as to insert an arc, delete an arc, or compute the intersection point of a pair of neighboring arcs. All of these operations can be performed in $\mathcal{O}(\log n)$ time. Moreover, for each face f in the arrangement $\mathcal{A}$, we maintain a list of all the discs overlapping face f. (Exploiting a standard packing argument and using Lemma 1, we know that there are only a constant number of discs overlapping any given face.) This additional information can be stored without asymptotically increasing the space complexity bounds. We conclude that the arrangement of n discs satisfying the empty disc property can be computed in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space.

To answer a reverse nearest neighbor query for a query point $q = (x_q, y_q)$, we use the persistent data structure to recover the status of the sweep-line at position $x = x_q$. By searching along the y-coordinate in the recovered y-structure, we locate the face f containing the query point q, and retrieve all $\mathcal{NN}$-discs overlapping f. As discussed above, the center points defining these $\mathcal{NN}$-discs are the solution to the the reverse nearest neighbor query. It takes $\mathcal{O}(\log n)$ time to recover the y-structure corresponding to a specific x-coordinate using the persistent data structure. It takes an additional $\mathcal{O}(\log n)$ time to locate the face f containing the query point q. Since there are only a constant number of $\mathcal{NN}$-discs overlapping a face, and they are also stored with the face, these $\mathcal{NN}$-discs can be retrieved in additional constant time. We summarize our discussion in the following theorem.

Theorem 1 *There exists a linear-space data structure that can be used to solve the (static) reverse nearest neighbor problem in two dimensions. Such a data structure for a set of n points can be constructed in $\mathcal{O}(n \log n)$ time. A query can be answered in $\mathcal{O}(\log n)$ time.*

3 The dynamic case

In this section, we make an attempt to design a data structure for a dynamically changing point set to answer reverse nearest neighbor queries efficiently. We wish to support the following operations:

Insert(p,T): Insert the point p (and the associated disc) into the data structure T.

Delete(p,T): Delete the point p (and the associated disc) from the data structure T.

Search(q,T): Search for all the discs in T that contain the query point q.

To state our results, we assume that a data structure (using $\mathcal{O}(S(n))$ space) exists that requires $\mathcal{O}(Q(n))$ time to update $\mathcal{S}$ while at the same time maintaining for each point *the distance* to its nearest neighbor. One of the main problems, however, is that designing a data structure for maintaining *nearest neighbors* in a dynamic setting using $\mathcal{O}(n \log^{\mathcal{O}(1)} n)$ space and $\mathcal{O}(\log^{\mathcal{O}(1)} n)$ update time is an open problem [7]. Consequently, the relevance of our result is somewhat dependent on a solution to this open problem, and we state all complexities including the terms $S(n)$ and $Q(n)$.

Our approach for designing the dynamic data structure for the reverse nearest neighbor problem is as follows: we split a disc into four quarter-discs of equal size using coordinate axis through the center of the disc. The boundary of each of these quarter-discs is monotone with respect to both coordinate axes. We build four data structures, one for each kind of quarter-discs. As the situations are symmetric, we only describe the data structure for maintaining north-east quarter-discs.

We can show that the set of quarter-discs has a very useful property:

Theorem 2 *For every quarter-disc Q, all quarter-discs Q_1, Q_2, ..., Q_q of the same type that overlap Q are pairwise disjoint.*

The dynamic data structure is a two-level tree structure whose base tree is a segment tree on the x-projections of the quarter-discs. Any node u of the primary tree corresponds to a vertical slab S_u, and all the quarter-discs that span the width of S_u, and not of its parent slab, are stored in the secondary structure associated with u. The secondary structure associated with the node u is an interval tree defined on the following intervals ordered according to y-coordinates. For all the quarter-discs stored at node u, intervals are the intersection of quarter-discs with the left boundary of S_u. The global (two-level) data structure is augmented such as to support dynamic fractional cascading [5].

To insert or delete a point (and its quarter-disc), we proceed as if we are doing that in the standard segment tree, except that we need to insert/delete in each of the interval trees along the search path. It is a straightforward task to modify the dynamic fractional cascading data structure of [5] to accommodate interval trees in place of "catalogue" at the expense of increasing the time complexity of these operations by a factor of $\log n$.

Reverse nearest neighbor queries are answered as follows. For a query point q use the primary segment tree structure to locate a node, say u, containing a set of quarter-discs whose x-projections encompass x-coordinate of q. To search in the secondary interval tree, we locate all the intervals that are stabbed by y-coordinate of q. We will show that there are at most two such intervals, and hence we report at most two quarter-discs in each slab. Therefore the query procedure is the same as that in dynamic segment tree with the exception that the search in each node of the tree is done in an interval tree at the expense of increasing the query time by a factor of $\log n$.

As a consequence of Theorem 2, any point in the plane will be covered by at most two quarter-discs of the same type. Consider the slab S_u associated with any node u of the segment tree. Let Z be the set of quarter-discs associated with the interval tree of node u such that the y-coordinate of the query point q stabs the intervals formed by the intersections of all discs in Z with the left boundary of S_u. Let Q be the quarter-disc among the discs in Z whose horizontal axis h passing through its center is immediately below the query point q. Let α be the intersection point of h and the left boundary of S_u. As all discs in Z span the whole width of S_u, it is easy to see that the intersection point α is in all the discs in the set Z, hence, by Theorem 2, $|Z| \leq 2$. The number of discs in Z actually containing q, however, might be smaller, as can be seen from Figure 2: In this situation, the candidate set Z consists of the quarter-discs Q and Q', but only Q' actually contains the query point q.

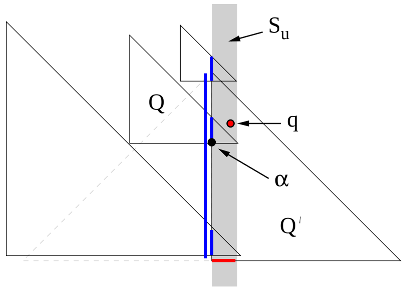

Figure 2: Querying a secondary interval tree.

We summarize the result in the following theorem.

Theorem 3 *A set $\mathcal{S}$ of n points in the plane can be maintained under insertions and deletions such that a reverse nearest neighbor query can be answered in $\mathcal{O}(\log^2 n \log\log n)$ time. The time complexity of each update operation is in $\mathcal{O}(Q(n) + \log^2 n \log\log n)$, and the space requirement for maintaining $\mathcal{S}$ is $\mathcal{O}(S(n) + n\log n)$.*

References

[1] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on Computers*, C-28(9):643–647, September 1979.

[2] K. L. Clarkson, H. Edelsbrunner, L. J. Guibas, M. Sharir, and E. Welzl. Combinatorial complexity bounds for arrangements of curves and spheres. *Discrete & Computational Geometry*, 5(2):99–160, 1990. A preliminary version appeared in *Proceedings of the 29th Symposium on Foundations of Computer Science* (1988), pages 568–579.

[3] A. Guttman. R-trees: A dynamic index structure for spatial searching. In B. Yormark, editor, *SIGMOD '84, Proceedings of Annual Meeting*, volume 14.2 of *SIGMOD Record*, pages 47–57. ACM Press, June 1984.

[4] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In W. Chen, J. Naughton, and P. A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, volume 29.2 of *SIGMOD Record*, pages 201–212, New York, June 2000. ACM Press.

[5] K. Mehlhorn and S. Näher. Dynamic fractional cascading. *Algorithmica*, 5(2):215–241, 1990.

[6] N. I. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, July 1986.

[7] M. Smid. Closest-point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 20, pages 877–935. Elsevier, Amsterdam, 2000.