Notes on the Dynamic Bichromatic All-Nearest-Neighbors Problem

Magdalene G. Borgelt* Christian Borgelt†

Abstract

Given a set S of n points in the plane, each point having one of c colors, the bichromatic all-nearest-neighbors problem is the task to find (in the set S) a closest point of different color for each of the n points in S. We consider a dynamic variant of this problem where the points are fixed but can change color. More precisely, we consider restricted problem instances, which allow us to improve over the time needed for solving the problem from scratch after each color change. In these variants we maintain, in $O(n)$ time per color change, a data structure of size $O(cn)$ or $O(n)$, with which the closest neighbor of different color of any point in S can be found in time $O(\log n)$, and the restrictions allow us to bound the number of look ups that are necessary in each step.

1 Introduction

Point proximity problems have been studied intensely over the years. One of the simplest instances is the *closest pair problem*, which consists in the task to find a closest pair of points from a set S of n points in d-dimensional space. It can be solved in $O(n \log n)$ time [5]. Its colored counterpart can be solved by computing a Euclidean minimum spanning tree, thus bounding the time complexity by the one of computing such a tree [10]. Better results are known for the special cases of 2 colors and 2 as well as 3 dimensions [1].

There are two dynamic versions of this problem. In the uncolored variant, the set S is modified by inserting or deleting points. [6] presents a data structure of size $O(n)$ that maintains a closest pair of S in $O(\log n)$ update time per insertion or deletion of a point. In the colored variant, the points in S are fixed, but they can change color dynamically. [10] showed that for an arbitrary number of colors the bichromatic closest pair can be maintained in $O(d+\log\log n)$ update time per color change with a data structure of size $O(n)$.

A natural extension of the closest pair problem is the *all-nearest-neighbors problem*, which is the task of finding the nearest neighbors for all points of a set S of n points. It can be solved, for arbitrary dimension d, in $O(n \log n)$ time [18]. The colored variant of this problem consists in the following task: given a set S of n points, each having one of c colors, find (in the set S) a closest point of different color for each of the n points in S. It can be solved in the plane in $O(n \log n)$ time for an arbitrary number of colors [2]. The static colored all-nearest-neighbor problem has applications in spatial databases and data mining [15, 13, 8, 19] and in information retrieval [7, 9].

Again there are two dynamic versions of this problem. In the uncolored variant, S is modified by inserting or deleting points. It can be solved by maintaining a Voronoi diagram for the set of points, thus bounding the time complexity to that of updating a Voronoi diagram (which in 2 dimensions takes $O(n)$ time per insertion or deletion, see Section 4). In the colored version the points in S are fixed, but they can change color dynamically. We consider this problem only in the plane ($d = 2$) and confine ourselves to restricted problem instances, which allow us to improve on the time that would be needed for solving the problem from scratch for each color change (leading to $O(n \log n)$ time per color change, since the static, colored all-nearest-neighbors problem can be solved in the plane with this time complexity).

2 A Closer Look at the Problem

Suppose that we obtained, by some algorithm, the closest bichromatic neighbor for each point in S for a given coloring of the points. What makes it difficult to maintain these closest neighbors under color changes? Obviously it is not that a point that changed from color i to color j is now eligible as the closest bichromatic neighbor of (other) points of color i. This type of update could easily be achieved in linear time: traverse all points of color i and replace their previously closest bichromatic neighbor if the point that changed color is closer. The difficulty comes from the complementary situation (see Figure 1): if the point that changed color from i to j was the closest bichromatic neighbor of some points of color j, a new bichromatic closest neighbor has to be found for each of these points. This is difficult, because all points not having color j are candidates for the new bichromatic closest neighbor, and simply searching them would take linear time per point that lost its closest neighbor.

The first idea to handle this problem is to maintain a data structure for all differently colored neighbors of a point, so that the next closest can be found quickly

*Department of Information and Computing Sciences, Utrecht University, Netherlands, magdalene@cs.uu.nl

†European Center for Soft Computing, Edificio Cientifico-Tecnologico, c/ Gonzalo Gutiérrez Quirós s/n, 33600 Mieres, Asturias, Spain, christian.borgelt@softcomputing.es

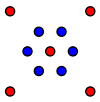

Figure 1: If the white point in the middle changes color, all black points need a new closest neighbor.

if the closest becomes invalid due to a color change. A natural choice for such a data structure would be a heap—like a binomial heap or a Fibonacci heap—for each point, which contains the bichromatic neighbors of that point. Unfortunately, this would not lead to a time complexity per color change that is better than solving the problem from scratch: in the worst case a linear number of heaps have to be updated by deleting points from them, and even the amortized time complexity of a deletion is $O(\log n)$, thus leading to a time complexity of $O(n \log n)$ per color change. In addition, the total size of all heaps would be $O(n^2)$.

3 Restricted Problem Instances

The efficiency of solutions to dynamic point proximity problems is measured by parameters like the preprocessing time, the space needed to store the data structures, the time needed to update the data structures, and the time needed to answer a user-defined query. When we studied the dynamic all-nearest-neighbor problem, we discovered that some options concerning the last two items in this list had not been investigated fully, and that there was room for improvement in certain restricted problem instances. Our rationale is that not all applications require solutions to the general problem and therefore faster solutions to restricted instances may be useful in practice.

Our ideas mainly rest on the insight that the standard problem can be seen as the task to maintain a data structure that allows a user to query for the closest, differently colored neighbor of any point in constant time. Unfortunately, this turns out to be surprisingly difficult. However, we found that it is actually fairly easy to maintain, in linear time, data structures that allow a user to query for the closest neighbor of any point in S in time $O(\log n)$ per point. Hence, if we have an application in which a user queries for the closest neighbor of, say, only a constant number of points after each color change, such a data structure is already very useful.

In addition, if we actually want to maintain, at any point in time, knowledge about the closest neighbor of each point in S, then such a solution can lead to an improvement under constraints. For example, if we face a restricted problem instance in which a color change only invalidates a sub-linear number of bichromatic neighbors (at least on average), having to look up the new neighbors still improves on the time complexity of solving the problem from scratch.

This reasoning led us to two restricted problem instances, in which improvements are possible. Suppose, in the first place, that the number of points that may have the same color is limited to $\frac{kn}{c}$, where n is the total number of points, c the number of colors, and k a real-valued constant greater than 1. Intuitively, this means that the color distribution may not deviate (upwards) more than a factor k from a uniform one. In this case we show that the update, maintaining knowledge of all closest neighbors (i.e., lookup in constant time per point), can be done in $O(\frac{n}{c} \log n)$ time per color change, while solving the problem from scratch still incurs $O(n \log n)$ time.

Alternatively, suppose that the sequence of points that change color is a concatenation of arbitrary permutations of the point set. In other words, in each section of n color changes that starts at index kn, $k \in \mathbb{N}_0$, each point appears at most once. Thus on average each point changes color every n steps, with 0 being the minimum and $2n-2$ being the maximum number of steps. For this case we show that an update takes $O(n)$ time on average, even though it can be as bad as $O(n^2)$ in the worst case. (It should be noted that, in principle, it is possible to generalize this result beyond concatenations of permutations, provided that on average a point still changes color every n steps. However, we confine ourselves to the more restricted version as this simplifies the analysis.)

4 Preliminaries

The data structures we use are always sets of Voronoi diagrams in the plane. Therefore we recall some properties of Voronoi diagrams that we need to show the properties of the update algorithms.

Given a set S of points in the plane, the *Voronoi region* $\mathrm{VR}(p)$ of a point p in S is the set of all points in the plane that are closer to the point p than to any other point in S. The *Voronoi diagram* $\mathrm{VD}(S)$ of S is the collection of all Voronoi regions of the points in S. Two points in S are said to be *Voronoi neighbors* if the closures of their regions have more than one point in common. These common points form a so-called *Voronoi edge*. Sometimes a Voronoi diagram is also defined as the union of all Voronoi edges.

Observation 1 *The number of Voronoi neighbor pairs in a Voronoi diagram for a set S of n points is at most $3n-6$ if $n > 3$.*

This observation follows immediately from the fact that the Voronoi diagram of a set S of points is closely related to the Delaunay triangulation of S: each midperpendicular of an edge in the Delaunay triangulation gives rise to at most one Voronoi edge. Therefore the neighbor graph of the Voronoi diagram (which is obtained by connecting all Voronoi neighbors in S by an edge) is a subgraph of the Delaunay triangulation. As a consequence the neighbor graph has at most as many edges as the Delaunay triangulation, which, as any triangulation, has $3n-6$ edges.

Observation 2 *The sum of degrees in the neighbor graph, over all points in S, is at most $6n - 12$.*

This observation follows trivially from the preceding one, since each edge (Voronoi neighbor pair) contributes to the degree of exactly two points.

The Voronoi diagram of n points can be computed in $O(n \log n)$ time and stored in $O(n)$ space and these bounds have been shown to be optimal in the worst case [16]. They can also easily be preprocessed for point location queries. The methods of [14] and [11], when combined with a linear time algorithm for triangulating simple polygons [17], need $O(n)$ preprocessing time. After preprocessing the Voronoi diagram, a point location query can be answered in $O(\log n)$ time [4]. It was also shown that n-point Voronoi diagrams can be updated in $O(n)$ worst case time per insertion or deletion of a point [12]. Consequently, a data structure allowing for point location queries in Voronoi diagrams can be maintained in $O(n)$ update time per insertion or deletion of a point.

5 Constrained Color Distribution

For the cases where the user queries only for the closest neighbors of a restricted number of points after each color change or the number of points having the same color is bounded by $\frac{kn}{c}$, we form the sets $S_i = \{p \in S \mid p \text{ has color } i\}$, $i = 1, \ldots, c$, and $C_i = S - S_i$ (that is, the complement sets of points for each color). Then we build the Voronoi diagrams $\mathrm{VD}(C_i)$, $i = 1, \ldots, c$, which takes $O(n \log n)$ time per Voronoi diagram and thus $O(cn \log n)$ total time. Preprocessing these Voronoi diagrams for point location adds $O(cn)$ time and thus does not change the overall time complexity. The total size of the Voronoi diagrams is $O(cn)$, since each has worst case size $O(n)$.

In order to maintain this data structure when a point p changes color from i to j, we have to update two Voronoi diagrams, namely $\mathrm{VD}(C_i)$ and $\mathrm{VD}(C_j)$. The former has to be updated by inserting p, since p now has color j and thus is in the complement of S_i, the latter by deleting p, since p is now in S_j. These two updates take $O(n)$ time each (see Section 4).

Of course, updating the two Voronoi diagrams only yields a data structure that allows us to query for the closest neighbor of each point in $O(\log n)$ time. However, suppose we queried the initial Voronoi diagrams with all points and recorded the closest bichromatic neighbor of each point. In order to update this direct knowledge about closest bichromatic neighbors, we distinguish again the two situations discussed in Section 2. For all points in C_i (except p), we check whether p is now the closest bichromatic neighbor and update accordingly. This obviously takes at most $O(n)$ time. For all points in S_j that had p as theirs closest bichromatic neighbor (which may be all points of S_j in the worst case), we have to look up new closest bichromatic neighbors. Due to the restricted problem instance we consider, we know that $|S_j| \leq \frac{kn}{c}$ and thus that we have to execute at most $O(\frac{n}{c})$ queries, each of which takes $O(\log n)$ time. Therefore the total update time is $O(\frac{n}{c} \log n)$ if we maintain a data structure in which the closest neighbor of any point can be found in constant time.

6 Constrained Update Sequence

For the case where the sequence of points that change color changes is a concatenation of arbitrary permutations of all points in S, we use the algorithm of [2] for the static colored all-nearest-neighbors problem. This algorithm builds $c + 1$ Voronoi diagrams: one for all points in S (that is, $\mathrm{VD}(S)$) and one for each of the c colors. The latter are formed for the sets $T_i = \{p \in C_i \mid p \text{ has a Voronoi neighbor in } S_i\}$, where "Voronoi neighbor" is meant w.r.t. $\mathrm{VD}(S)$. The reason is that in order to find a closest neighbor of different color for each point, it suffices to locate every point $p_i \in S_i$ in the Voronoi diagram of T_i [2]. Hence the Voronoi diagrams $\mathrm{VD}(T_i)$, $i = 1, \ldots, c$, are built. In addition, since we need this information for the update, we record for each (occurrence of a) point in each set T_i how many Voronoi neighbors it has in S_i.

Due to Observation 2 we have $\sum_{i=1}^{c} |T_i| \in O(n)$ and thus building all Voronoi diagrams $\mathrm{VD}(T_i)$, $i = 1, \ldots, c$, takes $O(n \log n)$ time. Preparing them for the queries takes $O(n)$ time, hence the overall time complexity is still $O(n \log n)$. The total size of all Voronoi diagrams is $O(n)$, and the total number of counters for the Voronoi neighbors in $\mathrm{VD}(S)$ is also $O(n)$.

In order to maintain this data structure when a point p changes color from i to j, we have to update two Voronoi diagrams, namely $\mathrm{VD}(T_i)$ and $\mathrm{VD}(T_j)$. Let $\mathrm{VN}(p)$ be the set of all Voronoi neighbors (w.r.t. $\mathrm{VD}(S)$) of p in S_i, that is, $\mathrm{VN}(p) = \{q \in S \mid q \text{ is a Voronoi neighbor of } p\}$. Since p is now in S_j, we have to add to T_j all points of $\mathrm{VN}(p)$ that are not yet in T_j (and must update $\mathrm{VD}(T_j)$ accordingly), and since p is no longer in S_i, we have to remove those points in $\mathrm{VN}(p)$ that do not have any other Voronoi neighbor in S_i from T_i (and must update $\mathrm{VD}(T_i)$ accordingly) — see Figure 2 for an example. Especially for the latter operation the Voronoi neighbor counters are important, because they make it easy to determine whether a point has another Voronoi neighbor that entitles it to stay in the set T_i.

Updating the Voronoi diagrams takes $O(n)$ time per point that has to be added or deleted. Since, in the worst case, a point can have $O(n)$ Voronoi neighbors, all of which may have to be added or deleted, the worst case time complexity of an update due to a color change is $O(n^2)$. However, one may also consider a scheme where the Voronoi diagrams are rebuild if too many points (more than $k \log n$ for some constant k)

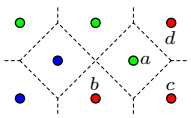

Figure 2: If a changes color from grey to black, b and c are removed from $\mathrm{VD}(T_{\mathrm{grey}})$ and c and d are inserted into $VD(T_{\mathrm{black}})$.

have to be added or deleted, thus reducing the worst case time complexity to $O(n \log n)$.

However, if the sequence of points that change color is a concatenation of arbitrary permutations of all points in S, we obtain a much better average time complexity. In order to demonstrate this, we consider one permutation of all points from the sequence of color changes, say, the section from step kn to step $(k+1)n-1$, $k \in \mathbb{N}_0$. Let p_s be the point that changed color in step s. Then we know from Observation 2 that $\sum_{s=1}^{n} \mathrm{VN}(p_s) \in O(n)$. Consequently, at most a linear number of points have to be inserted into and deleted from Voronoi diagrams in n steps. Since each of these updates take $O(n)$ time each, the total update time in n steps is bounded by $O(n^2)$. Therefore the average update time is linear per color change.

Of course, this procedure only maintains a data structure with which the closest bichromatic neighbor can be found in $(\log n)$ time. In order to improve the situation for a constant time lookup, we have to draw again on the restriction of the number of points that may have the same color. However, the advantage of this procedure is the smaller size of the data structure that needs to be maintained (namely $O(n)$ space instead of $O(cn)$).

7 Conclusions

We described two restrictions that may be introduced for the dynamic version of the all-nearest neighbor problem, in which points are fixed, but can change color. These restrictions allowed us to improve the time complexity of the update operations compared to that needed for solving the problem from scratch after each color change.

Acknowledgments

Partially funded by the Netherlands Organization for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.065.503.

References

[1] P. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean Minimum Spanning Trees and Bichromatic Closest Pairs. *Discrete and Computational Geometry* 6:407–422. 1991

[2] A. Aggarwal, H. Edelsbrunner, P. Raghavan, and P. Tiwari. Optimal Time Bounds for Some Proximity Problems in the Plane. *Information Processing Letters* 42:55–60. 1992

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

[4] F. Aurenhammer and R. Klein. Voronoi Diagrams. In: J.-R. Sack and J. Urrutia, eds. *Handbook of Computational Geometry*, 210–290. Elsevier Science, Amsterdam, Netherlands, 2000

[5] J. Bentley and M. Shames. Divide and Conquer in Multidimensional Space. *Proc. 8th Ann. ACM Symp. Theory of Computing*, 220–230. 1976

[6] S. Bespamyatnikh. An Optimal Algorithm for Closest Pair Maintenance. *Proc. 11th Ann. Symp. Computational Geometry*, 152–161. 1995

[7] W. Burkhard and R. Keller. Some Approaches to Best-match File Searching. *Communications of the ACM*, 230–236. 1973

[8] F. Cazals. Effective Nearest Neighbours Searching on the Hyper-cube, with Applications to Molecular Clustering. *Proc. 14th Ann. ACM Symp. Computational Geometry.* 1998

[9] L. Devroye and T. Wagner. Nearest Neighbor Methods in Discrimination. In: P.R. Krishnaiah and L.N. Kanal, eds. *Handbook of Statistics, Vol. 2.* North-Holland, Netherlands, 1982

[10] A. Dumitrescu and S. Guha. Extreme Distances in Multicolored Point Sets. *Proc. Int. Conf. Computational Science (ICCS 2002), Part III*, LNCS 2331, 14–25. Springer-Verlag, Heidelberg, Germany 2002

[11] H. Edelsbrunner, L. Guibas, and J. Stolfi. *Optimal Point Location in a Monotone Subdivision.* Report 2, Digital Systems Research Center, Palo Alto, California, USA 1984

[12] I. Gowda, D. Kirkpatrick, D. Lee, and A. Naamad. Dynamic Voronoi Diagrams. *Information Theory, IEEE Transactions* 29:724–731. 1983

[13] T. Hastie and R. Tibshirani. Discriminant Adaptive Nearest Neighbor Classification. *Proc. 1st Int. Conf. Knowledge Discovery and Data Mining*, 142–149. 1995

[14] D. Kirkpatrick. Optimal Search in Planar Subdivisions. *SIAM Journal Comp. Sci.* 28–35. 1983

[15] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, USA 1989

[16] M. Shamos and D. Hoey. Closest Point Problems. *Proc. 16th Ann. IEEE Symp. Foundations of Computer Science*, 151–162. 1975

[17] R. Tarjan and C. van Wyk. A Linear-time Algorithm for Triangulating Simple Polygons. *Proc. 18th Ann. ACM Symp. Theory of Computing.* 1985

[18] P. Vaidya. An $O(n \log n)$ Algorithm for the All-Nearest-Neighbors Problem. *Discrete and Computational Geometry* 4:101–115. 1989

[19] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao. All-Nearest-Neighbors Queries in Spatial Databases. *Proc. 16th Int. Conf. Scientific and Statistical Database Management*, 297–306. 2004