Minimizing the Sum of Diameters Efficiently

John Hershberger
DEC Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301
johnh@src.dec.com

Abstract

This note considers the problem of partitioning a planar set of n points into two subsets so that the sum of the diameters of the subsets is minimized. We present a simple algorithm that runs in $O(n \log^2 n / \log\log n)$ time, improving the previous $O(n^2)$ algorithm. In the case in which the ratio between the diameter and the minimum inter-point distance of the set is polynomial in n, a refinement of our algorithm runs in optimal $O(n \log n)$ time.

1 Introduction

Clustering problems are of fundamental importance in operations research. These problems specify a set of points S, a parameter k, a set measure μ, and a k-argument function f; the solution to the problem is a partition of S into k subsets $S_1, \ldots, S_k$ such that $f(\mu(S_1), \ldots, \mu(S_k))$ is minimized. Such problems are generally NP-hard for arbitrary k, even for planar point sets and for simple instances of μ and f (μ = diameter and f = maximum, for example) [7, 9, 11]. Therefore research has concentrated on the case of fixed k, for which polynomial algorithms are available [2, 8, 14].

This note focuses on the case in which S is planar, μ is the diameter, k is 2, and f is the sum. That is, we seek a bipartition of S that minimizes the sum of the diameters of the two subsets. This problem was considered by Monma and Suri, who gave an $O(n^2)$ algorithm [12]. We give an algorithm that improves their bound to $O(n \log^2 n / \log\log n)$. Our algorithm can be improved to $O(n \log n)$ when the ratio between the diameter of S and its minimum inter-point separation is polynomial in n. This is the case, for example, whenever the point coordinates are specified using fixed-precision machine arithmetic. The algorithm also gives subquadratic bounds for points in any fixed-dimensional Euclidean space.

2 Definitions

We need a few definitions before we can describe the algorithm. Let S be a set of n points in the plane, let the distance between two points p and q be $d(p,q)$, and let the diameter of a set X be denoted by $Diam(X)$. The closed disk with radius r centered on a point p is denoted by $D(p,r)$. For brevity, we refer to the sum of the diameters of the components of a bipartition as the *diameter sum* of the bipartition. We say that any bipartition with diameter sum less than $Diam(S)$ is *good*. Our algorithm finds the best of the good partitions, if any good partitions exist.

3 The basic algorithm

This section describes a simple algorithm to find a bipartition of S that minimizes the diameter sum. The algorithm is based on the following straightforward lemma:

Lemma 3.1 *Let a and b be a diametral pair of S. In any good bipartition of S, the subsets are contained in two disjoint disks centered on a and b.*

Proof: Let $\Delta \equiv d(a,b)$. Any good bipartition of S must have a and b in different sets, which we denote by S_a and S_b. Let r be the distance from a to the point in S_a farthest from it. The set S_a lies in the disk $D(a,r)$. Because $Diam(S_a) \geq r$, we must have $Diam(S_b) < \Delta - r$, and hence $S_b \subset D(b, \Delta - r - \epsilon)$ for some positive ϵ. ∎

The algorithm begins by computing a diametral pair a and b of S, which takes $O(n \log n)$ time [13]. Next it sorts the points of $S \setminus \{a,b\}$ into two lists L_a and L_b, one sorted by increasing distance from a and the other by increasing distance from b. Lemma 3.1 implies that for any good bipartition, the points in S_a must be a prefix of L_a and a suffix of L_b (their order may differ in the two lists). The algorithm identifies all prefixes of L_a whose elements form a suffix of L_b. To do this, it first marks each element of L_a with its rank in L_b, then prepares an empty array corresponding to the list L_b.

The algorithm marches through the elements of L_a, at each step marking the array entry given by the element's rank in L_b. Whenever a suffix of the array is marked, the algorithm detects it using union-find. This takes $O(n)$ total time [6], or $O(n \log n)$ time using a simpler algorithm based on a static binary tree.

To compute the diameter sum for all potentially good partitions, we use the semi-online algorithm of Dobkin and Suri for diameter maintenance [3]. We insert the elements of L_a into S_a in order, recording the diameter of the set as it changes. We do the same for L_b. For each prefix of L_a whose elements form a suffix of L_b, we add the two corresponding diameters. The minimum sum gives the best partition. Because the algorithm of Dobkin and Suri takes $O(n \log^2 n)$ time, we have the following theorem.

Theorem 3.2 *Given a planar set S of n points, the bipartition $S = S_1 \cup S_2$ that minimizes $Diam(S_1) + Diam(S_2)$ can be found in $O(n \log^2 n)$ time.*

As shown in the next section, this time bound can be improved to $O(n \log^2 n / \log\log n)$ by more careful exploitation of Dobkin and Suri's method.

4 A precision-sensitive improvement

The algorithm of the previous section works in the Real RAM model of computation, in which the point coordinates are specified with arbitrary precision. This section shows how to improve the running time of the algorithm to $O(n \log n)$ under the mild restriction that the ratio between the diameter and the minimum inter-point distance is polynomial in n. Alternatively, the algorithm can approximate the optimal bipartition to within a factor of $(1 + O(n^{-c}))$ in $O(n \log n)$ time.

The previous section shows how to identify values of r for which $S \subset D(a,r) \cup D(b, \Delta - r)$, where $\Delta = d(a,b)$. We will show that these values of r can be grouped into intervals such that the full complexity of Dobkin and Suri's algorithm is necessary only if there are many intervals. The lower bound of the ith interval increases exponentially with i, and hence if there are many intervals, the points of S must be specified to very high precision.

Without loss of generality, suppose that the segment $\overline{ab}$ is horizontal, with a at the left end. Divide the plane to the right of a into three 60° sectors T, M, and B as shown in Figure 1. As r increases, new points are inserted into S_a on the boundary of $D(a,r)$. After an insertion, the new $Diam(S_a)$ is the maximum of the old diameter and the distance from the new point to its farthest neighbor in S_a. For any new point in the sector M, the farthest point in the current set S_a is a or is in T or B; it cannot lie in M.

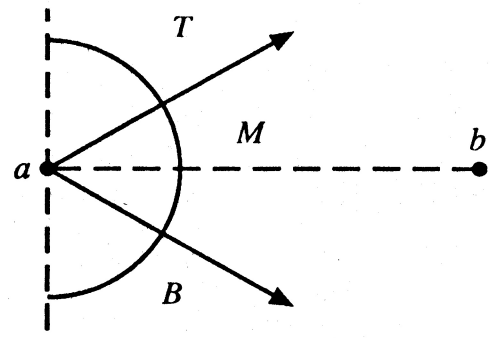

Figure 1: Partition the plane into sectors

Any point in T or B is evidence that the algorithm doesn't need to check diameters for a substantial range of values of r. Suppose that a point q lies in T or B at distance ρ from a. Then no value of r in $\frac{\sqrt{3}}{2}\rho \le r < \rho$ can give a good partition: for such radii, q is outside both $D(a,r)$ and $D(b, \Delta - r)$. With one pass through L_a, we can identify a sequence of radius values $r_1 < r'_1 < r_2 < r'_2 < \cdots$ such that no value of r outside the intervals $[r_1, r'_1), [r_2, r'_2), \ldots$ can give a good partition. Within each interval, the first insertion to S_a lies in T or B, but all the rest lie in M. Let k be the total number of such intervals.

Our algorithm exploits the facts that (1) within each interval $[r_i, r'_i)$, no new point except the first can be the farthest neighbor of any other, and (2) outside those intervals, no diameters need to be computed. Following the example of Dobkin and Suri, we partition the current S_a into disjoint subsets and maintain the farthest-point Voronoi diagram of each subset [1, 3, 4, 13]. To determine the farthest neighbor of a new point, we perform point location in each of the Voronoi diagrams. For purposes of the high-level algorithm below, let us arbitrarily define r_0 and r'_0 such that $r_0 < r'_0 < 0$.

For $i := 1$ to k do

1. Merge the points in the radial interval $(r_{i-1}, r_i]$ into the Voronoi diagram structure. (Recall that $r_{i-1} < r'_{i-1} < r_i < r'_i$.)
2. For each point p in $[r'_{i-1}, r'_i)$ in radial order do
   - Compute the farthest neighbor of p in the Voronoi diagram structure; call it q.
   - Update the diameter $Diam(S_a) := \max(Diam(S_a), d(p,q))$.

Lemma 4.1 *The preceding algorithm maintains the correct value of $Diam(S_a)$ in every interval $[r_i, r'_i)$, that is, in those intervals in which it is needed.*

Proof: The proof is by induction. The claim is vacuously true for $i = 0$. At the ith step, statement 1 updates the Voronoi diagram structure to include all the points in the interval $[0, r_i]$. The for-loop of statement 2 processes points in $[r'_{i-1}, r_i]$ before those in (r_i, r'_i). It first finds the farthest neighbor of each point in $[r'_{i-1}, r_i]$ among the points in $[0, r_i]$. The computed value of $Diam(S_a)$ may be larger than the true value during this period, but it does not matter. By the time the point at radius r_i is inserted, the computed value of $Diam(S_a)$ is correct once again. All the points in the interval (r_i, r'_i) lie in region M, and so their farthest neighbors are in the Voronoi diagram structure computed in statement 1. Thus $Diam(S_a)$ is correctly maintained for these points. ∎

We need to describe the Voronoi diagram structure more carefully. Let $t = \lceil \log k / \log\log n \rceil$, and let $\alpha = k^{1/t} \leq \log n$ (the reason for these choices will be apparent later). We maintain an array $V[1..t]$ of farthest-point Voronoi diagrams. Let $S[j]$ be the subset of S_a of which $V[j]$ is the Voronoi diagram. We maintain the invariant that the size of $S[j]$ is greater than $n\alpha^{j-1}/k$ and at most $n\alpha^j/k$ for $1 < j \leq t$, and is at most $n\alpha/k$ for $j = 1$.

At step 1 of the algorithm above, we determine the set P_i of points in the interval $(r_{i-1}, r_i]$. Let j be the index such that $n\alpha^{j-1}/k < |P_i| \leq n\alpha^j/k$, or $j = 1$ if $|P_i| \leq n\alpha/k$. We compute the convex hull of P_i, set $S[j] := S[j] \cup P_i$, and compute the convex hull of the new $S[j]$. If the new $S[j]$ is too big, with $|S[j]| > n\alpha^j/k$, we merge it into $S[j+1]$, set $j := j + 1$, and repeat until $S[j]$ is small enough. Finally we build the farthest-point Voronoi diagram of $S[j]$ and preprocess it for point location in $O(|S[j]|)$ time [1, 5, 10].

Lemma 4.2 *The total cost of maintaining the data structure described above is* $O(n \log n \lceil \log k / \log\log n \rceil)$.

Proof: Computing the convex hull of P_i may take $O(|P_i| \log |P_i|)$ time, but merging this convex hull with that of the old $S[j]$ takes only $O(|P_i| + |S[j]|)$ time. We can bound this by $O(\alpha|P_i| + n\alpha/k)$, since $S[j]$ is at most α times larger than P_i unless $j = 1$. The possible merges of $S[j]$ into $S[j+1]$ take $O(\alpha|S[j]|)$ time apiece by a similar argument. To obtain a global bound on the latter merging cost, note that as sets merge, no point ever moves into a set with a smaller index, and each merge of a smaller set into a larger takes time proportional to at most α times the size of the smaller set. We can charge each merge to the points whose set index increases, and hence the total cost of the merges that combine some $S[j]$ and $S[j+1]$ is $O(t\alpha)$ per point, or $O(t\alpha n)$ overall. The cost of building the Voronoi diagram is proportional to the cost of the merges that precede it, so we do not need to account for it separately. Putting it all together, we obtain a total bound of

$$\begin{aligned} O\left(t\alpha n + \sum_{i \leq k} (\alpha|P_i| + |P_i| \log |P_i| + n\alpha/k)\right) &= O(t\alpha n + n \log n + \alpha n) \\ &= O\left(n \log n \left\lceil \frac{\log k}{\log\log n} \right\rceil\right). \end{aligned}$$

∎

Because each farthest neighbor computation does a point location in each of t Voronoi diagrams, the total cost of step 2 of the algorithm is $O(nt \log n) = O(n \log n \lceil \log k / \log\log n \rceil)$. (The parameter t was chosen to balance this cost with that of maintaining the Voronoi diagrams.)

The preceding discussion shows how to compute all the required values of $Diam(S_a)$. We apply the same algorithm to compute $Diam(S_b)$, and thus establish the following theorem.

Theorem 4.3 *Let S be a planar set of n points, and let k be the parameter defined above, maximized over S_a and S_b. Then we can find a bipartition of S that minimizes the diameter sum in $O(n \log n \lceil \log k / \log\log n \rceil)$ time.*

Notice that this improves the bound of Theorem 3.2 to $O(n \log^2 n / \log\log n)$, since k is always less than n.

Corollary 4.4 *Let S be a planar set of n points such that the ratio of the diameter and the minimum inter-point distance is $O(n^c)$ for some constant c. Then we can find a bipartition of S that minimizes the diameter sum in $O(n \log n)$ time.*

Proof: We argue that k is $O(\log n)$ for S_a; the argument for S_b is symmetric. By the definition of the intervals $[r_i, r'_i)$, we have $r_{i+1} \geq (2/\sqrt{3})r'_i$ for any $i \geq 1$. If $k > 1$, there is a point of S at distance $r_2 > 0$ from a (r_1 might be 0), and so the minimum inter-point distance of S is at most r_2. We have $Diam(S) \geq r_k \geq r_2(2/\sqrt{3})^{k-2}$, and so $(2/\sqrt{3})^{k-2} = O(n^c)$, which implies that $k = O(\log n)$. ∎

The arguments above also imply that if we wish to approximate the optimal bipartition to within a factor of $(1+\epsilon)$, we need to process points in at most $O(\log(1/\epsilon))$ intervals, which takes only $O(n \log n)$ time if ϵ is fixed or $\epsilon = \Omega(n^{-c})$ for some constant c.

5 Extensions

The algorithms given in this note also work for points in higher dimensions. The time bounds degrade, because

the algorithms for computing and searching Voronoi diagrams in higher dimensions are more expensive than algorithms for two dimensions. The bound of Theorem 3.2 degrades to $O(n^{3/2}\log n)$ in three dimensions and $O(n^{2-1/(d(d+3)+4)}\log n)$ for dimension $d \geq 4$ [3, 15]. Because the cost of computing Voronoi diagrams is superlinear in three dimensions and above, the refinement of Theorem 4.3 no longer applies, but a separate argument shaves a logarithmic factor from the time bounds of Dobkin and Suri, giving the bounds quoted here [15].

Subhash Suri has observed that the algorithm of Theorem 3.2 also applies when the diameter is computed in the L_1 or L_∞ metric. In that case the problem is easier, and requires only $O(n \log n)$ time.

The appearance of a term dependent on the precision of the point coordinates in Theorem 4.3 is unusual in computational geometry, especially since no direct manipulation of coordinates (such as scaling) is performed. It may be that further analysis or algorithmic refinement will remove that dependency and result in an $O(n \log n)$ algorithm without restrictions.

Acknowledgment

Thanks to Subhash Suri for several helpful and enjoyable discussions about this work.

References

[1] A. Aggarwal, L. Guibas, J. Saxe, and P. Shor. A linear time algorithm for computing the Voronoi diagram of a convex polygon. *Discr. Comput. Geom.*, 4:591–604, 1989.

[2] T. Asano, B. Bhattacharya, M. Keil, and F. Yao. Clustering algorithms based on minimum and maximum spanning trees. In *Proceedings of the 4th ACM Symposium on Computational Geometry*, pages 252–257, June 1988.

[3] D. Dobkin and S. Suri. Dynamically computing the maxima of decomposable functions, with applications. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 488–493, 1989.

[4] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987.

[5] H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15:317–340, 1986.

[6] H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. Comput. Syst. Sci.*, 30:209–221, 1985.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco, 1979.

[8] J. Hershberger and S. Suri. Finding tailored partitions. In *Proceedings of the 5th ACM Symposium on Computational Geometry*, pages 255–265, 1989. To appear in *J. Algorithms*.

[9] D. S. Johnson. The NP-completeness column. *J. Alg.*, 3, 1982.

[10] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12:28–35, 1983.

[11] N. Meggido and K. Supowit. On the complexity of some common geometric location problems. *SIAM J. Comput.*, 13(1):182–196, 1984.

[12] C. Monma and S. Suri. Partitioning points and graphs to minimize the maximum or the sum of diameters. In *Proceedings of the Sixth International Conference on the Theory and Applications of Graphs*. John Wiley & Sons Publishers, 1989.

[13] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer-Verlag, New York, 1985.

[14] G. Rote and G. Woeginger. Geometric clusterings. Technical Report (Serie B—Informatik) B-89-04, Freie Universität Berlin, April 1989.

[15] M. Smid. A worst-case algorithm for semi-online updates on decomposable problems. Technical Report A 03/90, Fachbereich Informatik, Universität des Saarlandes, 1990.