TWO ALGORITHMS FOR THE SUM OF DIAMETERS PROBLEM AND A RELATED PROBLEM

MUHAMMAD H. ALSUWAIYEL

Department of Information and Computer Science
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
e-mail: facp079@saupm00.bitnet

ABSTRACT

Given a set S of points in the plane, we consider the problem of partitioning S into two subsets such that the sum of their diameters is minimized. We present two algorithms with time complexities $O(n \log^2 n / \log \log n)$ and $O(n \log n/(1-\epsilon))$, where $\epsilon, 0 < \epsilon < 1$, is a real number that is dependent on the density of the point set. In almost all practical instances, the second algorithm runs in optimal $O(n \log n)$ time, improving all previous results in the case of nonsparse point sets. These bounds follow immediately from two corresponding algorithms with the same time complexities for the following problem: given a set of points $S = \{p_1, p_2, \ldots, p_n\}$ in the plane sorted in increasing distance from p_1, compute the sequence of diameters $d_1, d_2, \ldots, d_n$, where $d_i = Diam\{p_1, \ldots, p_i\}$ is the diameter of the first i points, $1 \leq i \leq n$.

1 Introduction

Let S be a set of n points in the plane. We consider the problem of partitioning S into two disjoint subsets whose sum of diameters is minimum. More precisely, given a set $S = \{p_1, p_2, \ldots, p_n\}$ of n points in the plane, we are concerned with the problem of finding a bipartition $\{S_1, S_2\}$ of S such that $Diam(S_1) + Diam(S_2)$ is minimum among all possible bipartitions. In general, the problems of partitioning a point set into k partitions, or covering a point set with k convex objects, e.g. disks, are intractable. When the number of partitions or convex objects is restricted to two, a number of algorithms have been developed in order to meet a given criterion. For the case when the problem is partitioning into two partitions and the criterion is to minimize the sum of their diameters, Monma and Suri [6] gave an $O(n^2)$ time algorithm to solve this problem. A subquadratic algorithm was first given by Hershberger [4] who proposed an algorithm to solve this problem in $O(n \log^2 n/\log\log n)$ time. He has also shown that if the ratio between the diameter and the minimum inter-point distance is polynomial in n, then a refinement of the algorithm improves the bound to $O(n \log n)$. Equivalently, the refined algorithm finds an approximate solution to within a factor of $(1+O(n^{-c}))$ in $O(n \log n)$ time.

In this paper, we propose two algorithms to solve this problem. The first algorithm is a simple divide-and-conquer algorithm that runs in time $O(n \log^2 n/\log\log n)$. Although, as stated above, an algorithm for this problem with the same time complexity has been given in[4], the merit of this algorithm is its simplicity and parallelizability. Next, we develop another algorithm that runs in time $O(n \log n/(1-\epsilon))$ for some $\epsilon, 0 < \epsilon < 1$, that is characteristic of the density of the point set. In almost all practical instances, the value of ϵ is reasonably small so that the time complexity reduces to $O(n \log n)$. In both algorithms, the problem is divided into a number of subproblems ($\lceil \log n \rceil$ subproblems in the first algorithm and $\lceil n^c \log n \rceil$ subproblems in the second algorithm, for some constant $c, 0 < c < 1$, that is dependent on the density of the point set).

These two bounds follow immediately from two corresponding bounds for the following problem, which we will refer to as *the sequence of diameters problem*: given a set of points $S = \{p_1, p_2, \ldots, p_n\}$ in the plane sorted in increasing distance from p_1, compute the sequence of diameters $d_1, d_2, \ldots, d_n$, where $d_i = Diam\{p_1, \ldots, p_i\}$ is the diameter of the first i points, $1 \leq i \leq n$. As will be shown later, the problem of minimizing the sum of diameters trivially reduces to the sequence of diameters problem (in $O(n \log n)$ time). However, it turns out that the latter is interesting in its own right and may be used as a subroutine in other clustering algorithms. The rest of the paper is organized as follows. Section 2 has the preliminaries. Section 3 is devoted for the sequence of diameters problem. In Section 4, we apply the main results of Section 3 to obtain a revised version of Hershberger's algorithm[4]. Finally, Section 5 concludes with some discussions and remarks.

2 Preliminaries

Let S be a set of points in the plane. The Euclidean distance between any two points $u, v \in S$ is denoted by $d(u, v)$. The diameter of S, denoted by $Diam(S)$, is the maximum distance realized by any two points in S. Two points $u, v \in S$ are called *diametral* if $d(u, v) = Diam(S)$.

For any set of points S, $CH(S)$ is the set of points on the convex hull of S and $\mathcal{FVD}(S)$ will denote the farthest-point Voronoi diagram of S. The algorithms presented in this work rely heavily on the fact that two Voronoi diagrams can be merged in linear time. We present here the very basic lemma due to Kirkpatric[5]

Lemma 1 *If S_1 and S_2 are arbitrary planar point sets, then $\mathcal{FVD}(S_1 \cup S_2)$ can be constructed from $\mathcal{FVD}(S_1)$ and $\mathcal{FVD}(S_2)$ in $O(|S_1| + |S_2|)$ time.*

An important implication of Lemma 1 is that S_1 and S_2 do not have to be separable. Let $S = \{p_1, p_2, \ldots, p_n\}$ be an *ordered* set of points in the plane sorted in increasing distance from p_1. For any point $p_i \in S$ and any subset $T \subseteq S$, define $f(p_i, T)$ as follows:

$$f(p_i, T) = \begin{cases} p_1 & \text{for } i = 1 \\ \text{the farthest neighbor of } p \text{ in } T \cap \{p_1, p_2, \ldots, p_{i-1}\} & \text{for } 2 \leq i \leq n. \end{cases}$$

It will be appropriate if we call $f(p, S)$ the *restricted* farthest point (or neighbor) of p.

3 Computing the sequence of diameters

In this section, we confine our attention to the sequence of diameters problem. We describe two algorithms for solving this problem; the first algorithm, which we will refer to as $SEQD1$, is a straightforward divide-and-conquer algorithm that runs in time $O(n \log^2 n / \log\log n)$. Next, we present Algorithm $SEQD2$ which is a refinement of $SEQD1$ in which we exploit the fact that in almost every point set of size n, the number of its hull vertices is no more than n^ϵ for some reasonably small $\epsilon, 0 < \epsilon < 1$. This is supported by the well-known result that if a set of n points are chosen uniformly and independently in the plane from a convex r-gon, then as n approaches ∞, the expected number of points on their convex hull is $(2r/3)(\gamma + \log_e n) + O(1)$, where γ is Euler's constant[8].

3.1 The first algorithm

In this subsection, we describe the first algorithm, $SEQD1$. It is a simple divide-and-conquer algorithm: the point set S is partitioned into $k \geq 2$ groups of approximately $\lceil |S|/k \rceil$ points each, which give rise to k subproblems that are recursively solved. The merge step consists of a sweep over the k subsets $S_1, S_2, \ldots, S_k$, in which point locations and merging of farthest-point Voronoi diagrams are performed in their order of increasing indices.

First, assuming that $\mathcal{FVD}(S_j)$ and $f(p, S_j)$ for each point $p \in S_j, 1 \leq j \leq k$, have all been computed as a result of k previous calls to $SEQD1$, we do an interleaved sequence of point locations and merging of farthest-point Voronoi diagrams as follows. For each point $p \in S_2$, we compute

$q = f(p, S_1)$. If $d(p, q) > d(p, f(p, S_2))$, then we set $f(p, S) = q$; otherwise we set $f(p, S) = f(p, S_2)$. Next $\mathcal{FVD}(S_1 \cup S_2)$ is constructed by merging $\mathcal{FVD}(S_1)$ and $\mathcal{FVD}(S_2)$. In general, in the jth step, $2 \leq j \leq k$, we do a point location on $\mathcal{FVD}(S_1 \cup S_2, \ldots, \cup S_{j-1})$ to find $q = f(p, S_1 \cup S_2, \ldots, \cup S_{j-1})$ for each point $p \in S_j$. If $d(p, q) > d(p, f(p, S_j))$, then we set $f(p, S) = q$; otherwise we set $f(p, S) = f(p, S_j)$. This is immediately followed by merging $\mathcal{FVD}(S_1 \cup S_2, \ldots, \cup S_{j-1})$ and $\mathcal{FVD}(S_j)$ to produce $\mathcal{FVD}(S_1 \cup S_2, \ldots, \cup S_j)$.

Procedure $SEQD1$ below finds $f(p, S)$ for each point $p \in S$. After this procedure terminates, a scan over the set of *sorted* points in S together with their respective restricted farthest neighbors gives the desired sequence of diameters $d_1, d_2, \ldots, d_n$. Specifically, $d_1 = 0$ and for each $j, 1 < j \leq n, d_j = \max\{d_{j-1}, d(p_j, f(p_j, S))\}$. The value of k used to partition the given point set is to be determined later.

```
**procedure** $SEQD1(S)$

  1. If $|S| < k$, then use a straightforward method to find $f(p, S)$ for each point $p \in S$. Construct
     $\mathcal{FVD}(S)$. Return $f(p, S)$ for each $p \in S$, and $\mathcal{FVD}(S)$.

  2. Divide $S$ into $k$ subsets $S_1, S_2, \ldots, S_k$ of $\lceil |S|/k \rceil$ points each (except possibly $S_k$).

  3. For each $j, 1 \leq j \leq k$, recursively call $SEQD1(S_j)$ to obtain $\mathcal{FVD}(S_j)$ and $f(p, S_j)$ for each
     point $p \in S_j$.

  4. Let $\mathcal{FVD}_1 = \mathcal{FVD}(S_1)$. For $j = 2, 3, \ldots k$, do the following:

     (a) For each point $p \in S_j$ do a point location on $\mathcal{FVD}_{j-1}$ to compute $q = f(p, S_1 \cup S_2 \cup
         \ldots \cup S_{j-1})$. If $d(p, q) > d(p, f(p, S_j))$, then set $f(p, S) = q$, else set $f(p, S) = f(p, S_j)$.
     (b) Merge $\mathcal{FVD}_{j-1}$ with $\mathcal{FVD}(S_j)$ to obtain $\mathcal{FVD}_j$.

  5. Set $\mathcal{FVD}(S) = \mathcal{FVD}_k$. Return $\mathcal{FVD}(S)$ and $f(p, S)$, for each point $p \in S$.

**end** $SEQD1$
```

Clearly, the above procedure computes $f(p, S)$ for each point $p \in S$. This is immediate from the fact that for each pair of points p_i and $p_j, 1 \leq i < j \leq n$, p_i is tested for the possibility of being the restricted farthest point of p_j. Specifically, for each pair of points p_i and $p_j, 1 \leq i < j \leq n$, if $f(p_j, S) = p_i$, then there is a subset $T \subseteq \{p_1, p_2, \ldots, p_{j-1}\}$ such that $p_i \in T$ and a point location is performed on $\mathcal{FVD}(T)$ to find the farthest neighbor of p_j in T. The running time of the algorithm is computed as follows. Steps 1 and 2 take $O(n)$ time. Step 3 takes $O(kT(n/k))$ time. In Step 4.a, exactly n point locations are performed for a total cost of $O(n \log n)$. By Lemma1, the cost of Step 4.b, the time needed to compute all $\mathcal{FVD}_j$'s, $2 \leq j \leq k$, is proportional to

$$\begin{aligned} 2\lceil |S|/k \rceil + 3\lceil |S|/k \rceil + \ldots + k\lceil |S|/k \rceil &= O(k^2|S|/k) \\ &= O(k|S|) \\ &= O(kn). \end{aligned}$$

Finally, Step 5 takes $O(1)$ time. If we assume for simplicity that n is a multiple of k, then

$$T(n) = kT(n/k) + O(n \log n) + O(kn).$$

If we choose $k = 2$, we obtain $T(n) = \Theta(n \log^2 n)$. If we choose $k = \lceil \log n \rceil$, we obtain $T(n) = \Theta(n \log^2 n / \log \log n)$. Increasing the value of k beyond $\lceil \log n \rceil$ does not lead to any reduction in the asymptotic running time of the algorithm, since doing so increases the cost of merging the farthest-point Voronoi diagrams. This proves the following lemma.

Lemma 2 *The sequence of diameters corresponding to a set of n points in the plane can be computed in time $O(n \log^2 n / \log \log n)$ in the worst case.*

3.2 The second algorithm

In Algorithm $SEQD1$ above, if we let $k = \lceil \log n \rceil$, then as many as $\lceil \log_{\lceil \log n \rceil} n \rceil = \Theta(\log n / \log \log n)$ point locations are performed in order to compute $f(p, S)$ for each point $p \in S$. Consequently, the total number of point locations performed by the algorithm is $O(n \log n / \log \log n)$. In what follows, we show that the number of point locations can be reduced drastically in almost all practical instances.

Since for any point set $S, \mathcal{FVD}(S) = \mathcal{FVD}(CH(S))$[8], then by Lemma 1, the cost of merging two farthest-point Voronoi diagrams $\mathcal{FVD}(S_1)$ and $\mathcal{FVD}(S_2)$ of two point sets S_1 and S_2 is proportional to $|CH(S_1)| + |CH(S_2)|$. In the following, we exploit this fact to improve the worst case running time to $O(n \log n / (1 - \epsilon))$ for some $\epsilon, 0 < \epsilon < 1$.

Let $\gamma = 1/\sqrt{\log n}$. Suppose that the number of hull vertices of any subset $T \subseteq S$ of size n^γ or more is at most $\lceil |T|^\epsilon \rceil$ for some $\epsilon, 0 < \epsilon < 1$. Then, a worst case bound of $O(n \log n / (1 - \epsilon))$ can be achieved by partitioning the point set into $\alpha = \lceil n^{(1-\epsilon)/(2-\epsilon)} \rceil$ disjoint subsets and proceeding as in Algorithm $SEQD1$. Algorithm $SEQD1'$ below implements this idea.

procedure $SEQD1'(S)$

```
1. If $|S| \leq \lceil n^{\gamma} \rceil$ then call $SEQD1(S)$ and halt; otherwise continue.
2. Divide $S$ into $\alpha$ subsets $S_1, S_2, \ldots, S_\alpha$. Each subset( except possibly $S_\alpha$) consists of $\lceil n/\alpha \rceil = \lceil n / \lceil n^{(1-\epsilon)/(2-\epsilon)} \rceil \rceil \leq \lceil n^{1/(2-\epsilon)} \rceil$ points.
3. For each $j, 1 \leq j \leq \alpha$, recursively call $SEQD1'(S_j)$ to obtain $\mathcal{FVD}(S_j)$ and $f(p, S_j)$ for each point $p \in S_j$.
4. Let $\mathcal{FVD}_1 = \mathcal{FVD}(S_1)$. For $j = 2, 3, \ldots, \alpha$ do the following
    (a) For each point $p \in S_j$ use $\mathcal{FVD}_{j-1}$ to compute $q = f(p, S_1 \cup S_2 \cup \ldots \cup S_{j-1})$. If $d(p,q) > d(p, f(p, S_j))$, then set $f(p,S) = q$, else set $f(p,S) = f(p,S_j)$.
    (b) Merge $\mathcal{FVD}_{j-1}$ with $\mathcal{FVD}(S_j)$ to obtain $\mathcal{FVD}_j$.
5. Set $\mathcal{FVD}(S) = \mathcal{FVD}_\alpha$. Return $\mathcal{FVD}(S)$ and $f(p,S)$, for each point $p \in S$.
end $SEQD1'$
```

Let $T(n)$ be the worst case running time of Algorithm $SEQD1'$. By Lemma 2, each call of procedure $SEQD1$ in Step 1 costs

$$\begin{aligned}
O(\lceil n^{\gamma} \rceil \log^2 \lceil n^{\gamma} \rceil / \log\log \lceil n^{\gamma} \rceil) &= O(\gamma^2 n^{\gamma} \log^2 n / \log\log n^{\gamma}) \\
&= O(\gamma^2 n^{\gamma} \log^2 n / \log(\gamma \log n)) \\
&= O((1/\sqrt{\log n})^2 n^{\gamma} \log^2 n / \log(\sqrt{\log n})) \\
&= O(n^{\gamma} \log n / \log\log n).
\end{aligned}$$

Hence, the overall time taken by this step is $O(n \log n / \log\log n)$. Step 2 takes $O(n)$ time. In Step 4.a, the time needed for point locations is $O(n \log n)$. Since the size of a farthest-point Voronoi diagram of a set S_j is proportional to $|CH(S_j)| \leq |S_j|^{\epsilon}$, the time needed to compute all $\mathcal{FVD}_j$'s, $2 \leq j \leq \alpha$, in Step 4.b is proportional to

$$\begin{aligned}
2\lceil n^{1/(2-\epsilon)} \rceil^{\epsilon} + 3\lceil n^{1/(2-\epsilon)} \rceil^{\epsilon} + \ldots + \alpha \lceil n^{1/(2-\epsilon)} \rceil^{\epsilon} &= O(\alpha^2 \lceil n^{1/(2-\epsilon)} \rceil^{\epsilon}) \\
&= O(\lceil n^{(1-\epsilon)/(2-\epsilon)} \rceil^2 \lceil n^{1/(2-\epsilon)} \rceil^{\epsilon})) \\
&= O(n^{2(1-\epsilon)/(2-\epsilon)} n^{\epsilon/(2-\epsilon)})) \\
&= O(n^{(2-2\epsilon)/(2-\epsilon) + \epsilon/(2-\epsilon)}) \\
&= O(n).
\end{aligned}$$

Let c denote the recursion depth. Then

$$\begin{aligned} c &\leq \lceil \log_{\alpha}\ n \rceil \\ &= \lceil \log_{\lceil n^{(1-\epsilon)/(2-\epsilon)} \rceil}\ n \rceil \\ &\leq \lceil \log_{n^{(1-\epsilon)/(2-\epsilon)}}\ n \rceil \\ &= \lceil \frac{2-\epsilon}{1-\epsilon} \rceil . \end{aligned}$$

It follows that $T(n) = O(cn \log n) = O(n \log n/(1-\epsilon))$. This result is summarized in the following lemma.

Lemma 3 *If for each subset $T \subseteq S$ with $|T| \geq n^{1/\sqrt{\log n}}$ the number of points on the convex hull of T is at most $\lceil T^{\epsilon} \rceil$ for some $\epsilon, 0 < \epsilon < 1$, then the sequence of diameters corresponding to S can be computed in time $O(n \log n/(1-\epsilon))$.*

The above lemma is not constructive in the sense that only the existence of an $O(n \log n/(1-\epsilon))$ algorithm is shown. In what follows, we show how to calculate a value of ϵ which works well for nonsparse point sets. Suppose that the points in S are sorted in increasing distance from p_1. Suppose also that for any subset T of $\lceil n^{\gamma} \rceil$ *consecutive* points, the number of points on the convex hull of T is at most $\lceil \lceil n^{\gamma} \rceil^{\epsilon} \rceil = \lceil n^{\epsilon\gamma} \rceil$. Let T' be any subset whose size is a multiple of $\lceil n^{\gamma} \rceil$, say $k\lceil n^{\gamma} \rceil$ for some $k \geq 2$. Then, it is also the case that the number of hull vertices in T' is upperbounded by $k\lceil n^{\epsilon\gamma} \rceil$. This suggests that if we restrict the size of each subset in Algorithm $SEQD1'$ to be a multiple of $\lceil n^{\gamma} \rceil$, then in order to guarantee an $O(n \log n/(1-\epsilon))$ upper bound, it suffices to find some $\epsilon, 0 < \epsilon < 1$, such that any *consecutive* $\lceil n^{\gamma} \rceil$ points in S have at most $\lceil n^{\epsilon\gamma} \rceil$ hull vertices. A straightforward approach for calculating ϵ is by scanning the points in their sorted order. We maintain a queue Q of size $\lceil n^{\gamma} \rceil$. We start by pushing all the first $\lceil n^{\gamma} \rceil$ points into the queue and find their convex hull. Next, by scanning the remaining points in their sorted order, each time a point is pushed into Q and another one is deleted. At the same time, the convex hull of the current $\lceil n^{\gamma} \rceil$ points is updated. This procedure, however, does not lead to an efficient algorithm if, for instance, the distribution of the point set is irregular; it may happen that, although S is dense, there is a subset T of $\lceil n^{\gamma} \rceil$ consecutive points whose convex hull is T itself. This suggests that a better computation of ϵ must apply not only to uniformly distributed point sets, but also to those sets that are highly irregular. To achieve this, we choose to calculate such an ϵ in the following way. We partition S into a number of subsets constructed as follows. To construct subset S_j, we keep adding points to it until either the size of its convex hull becomes $\lceil n^{\gamma} \rceil$ or its size reaches some predefined limit. Thus, in both cases the number of hull vertices of S_j is at most $\lceil n^{\gamma} \rceil$. Specifically, we partition S into k_1 subsets, $S_1, S_2, \ldots, S_{k_1}$, having the following two properties:

1. For each $j, 1 \leq j \leq k_1, |CH(S_j)| \leq \lceil n^\gamma \rceil$.

2. Each subset $S_j, 1 \leq j \leq k_1$, consists of at most $\lceil n^{m\gamma} \rceil$ consecutive points, where $p_1 \in S_1$ and $p_n \in S_{k_1}$. Here, $m \geq 2$ is a positive integer constant. However, m can be as large as $\lceil 2 + \sqrt{\log \log n} \rceil$ (see the proof of Lemma 5).

Finally, to determine ϵ, we solve the equation $h = n^\epsilon$ for ϵ, where $h = \sum_{j=1}^{k_1} |CH(S_j)|$.

For brevity, let us call each one of the subsets $S_1, S_2, \ldots, S_{k_1}$ a γ-subset. Before we give the second algorithm, which we will call $SEQD2$, we first dispose of the problem of partitioning S into γ-subsets satisfying the abovementioned two properties. This is outlined in the following *preprocessing step.*

Preprocessing step. Starting from an empty convex hull, scan the points in S in their sorted order of increasing distance from p_1 to build the first γ-subset, S_1. Each time a point is encountered, the convex hull and its size are updated. Let r be the *minimum* index between 1 and $\lceil n^{m\gamma} \rceil$ such that either $|CH(\{p_1, p_2, \ldots, p_r\})| = \lceil n^\gamma \rceil$ or $|\{p_1, p_2, \ldots, p_r\}| = \lceil n^{m\gamma} \rceil$ or $r = n$. Set $S_1 = \{p_1, p_2, \ldots, p_r\}$. Now, if $r < n$, then starting at point p_{r+1}, repeat the same procedure to compute the second γ-subset S_2. We continue this procedure of partitioning S until S_{k_1}, which contains p_n, is finally computed. Note that the size of the convex hull of each γ-subset will never exceed $\lceil n^\gamma \rceil$ as the insertion of one point increases the size of the current convex hull by at most one. Clearly, this preprocessing step involves the use of a convex hull maintenance algorithm that supports insertions and deletions in logarithmic time, and hence its overall time complexity is $O(n \log n)$[8].

At this point, it will be appropriate if we make precise the notion of a "dense point set". The following definition is good enough for the sake of analyzing the algorithm.

Definition 1 *A point set is said to be dense if for some $\epsilon, 0 < \epsilon < 1$, that is sufficiently small, $h = O(n^\epsilon)$, where $h = \sum_{j=1}^{k_1} |CH(S_j)|$. Here $S_j, 1 \leq j \leq k_1$, are the γ-subsets constructed by partitioning the point set S as described in the preprocessing step.*

After computing the γ-subsets, Algorithm $SEQD2$ proceeds by processing each $\gamma-$subset for "local" restricted farthest points in time $O(n^{m\gamma} \log^2 n^{m\gamma} / \log \log n^{m\gamma}) = O(n^{m\gamma} \log n / \log \log n)$ (as will be shown later) using procedure $SEQD1$(recall that $\gamma = 1/\sqrt{\log n}$). The algorithm then proceeds by merging each α (a number to be determined later) γ-subsets into a new larger subset. This process of merging and building larger subsets continues until there is only one subset left, namely S. We note that this procedure is inherently iterative and is best implemented using a queue to store (pointers to) the current subsets. Thus, the algorithm proceeds in *stages* with the merging of *all* subsets of the same size indicating the end of a stage and the beginning of the next. The number α is chosen so that it satisfies the following two conditions:

1. The time taken to merge the farthest-point Voronoi diagrams of all subsets in one stage is $O(n \log n)$.

2. At least $\lceil \log n \rceil$ subsets are used in each merge.

The first condition ensures that the algorithm will take $O(n \log n)$ in the case of dense point sets, whereas the second condition guarantees that the overall running time is $O(n \log^2 n / \log \log n)$ in the extreme case when $h \approx n$, where, as stated before, h is the total number of hull vertices of the γ-subsets. In the latter case, the performance of Algorithm $SEQD2$ *almost* reduces to that of $SEQD1$.

Given the preprocessing step outlined above, the following is a description of Algorithm $SEQD2$.

```
procedure $SEQD2$

1. Let $m = \lceil 2 + \sqrt{\log \log n} \rceil$ and do the preprocessing step to partition $S$ into $k_1$ $\gamma$-subsets
   $S_1, S_2, \ldots, S_{k_1}$.

2. For $j = 1, 2, \ldots, k_1$, call $SEQD1(S_j)$ to find $f(p, S_j)$ for all points $p \in S_j$. Algorithm
   $SEQD1$ will also compute $\mathcal{FVD}(S_j)$, the farthest-point Voronoi diagram of subset $S_j$, for
   each $j, 1 \leq j \leq k_1$.

3. Let $h = \sum_{j=1}^{k_1} |CH(S_j)|$. Set $\epsilon = \log_n h$ and $\alpha = \lceil \, n^{1-\epsilon} \log n \rceil$. Label each $\gamma$-subset $S_j$ with
   $\lambda(S_j) = 1, 1 \leq j \leq k_1$. Push all $\gamma$-subsets into an empty queue $Q$. Set $k = k_1$.

4. While $Q$ contains more than one item do the following steps.

   (a) Let $T_i$ be the subset at the front of $Q$. Let $m$ be the number of subsets in $Q$ whose label
       is $\lambda(T_i)$ and $l = \min\{\alpha, m\}$. Pop $T_i$ from the front of $Q$. Set $k = k + 1, T_k = T_i$ and
       $\mathcal{FVD}(T_k) = \mathcal{FVD}(T_i)$.

   (b) For $j = i + 1, i + 2, \ldots, i + l - 1$, do the following:

       i. Pop $T_j$ from the front of $Q$.

       ii. For each point $p \in T_j$ use $\mathcal{FVD}(T_k)$ to compute $q = f(p, T_k)$. If $d(p, q) > d(p, f(p, (T_j)))$,
           then set $f(p, T_k) = q$, else set $f(p, T_k) = f(p, T_j)$.

       iii. Set $T_k = T_k \cup T_j$ and update $\mathcal{FVD}(T_k)$ by merging $\mathcal{FVD}(T_k)$ with $\mathcal{FVD}(T_j)$.
```

(c) At this point, $T_k = T_i \cup T_{i+1} \cup \ldots \cup T_{i+l-1}$ and $\mathcal{FVD}(T_k) = \mathcal{FVD}(T_i \cup T_{i+1} \cup \ldots \cup T_{i+l-1})$. Set $\lambda(T_k) = \lambda(T_i) + 1$ and enter T_k at the back of Q.

end *SEQD*2

It is not hard to see that Algorithm *SEQD*2 correctly computes $f(p, S)$ for each $p \in S$. We observe that the labels of subsets in the queue are ordered in increasing order with the lowest being of the subset in the front of the queue. We also observe that Q cannot have 3 subsets of pairwise different labels. These observations are direct consequence of how the algorithm operates. To this end, let stage i denote the time period in which the queue has more than one subset *and* one or more subsets in the queue have label i. The following lemma bounds the number of stages.

Lemma 4 *The number of stages in Algorithm SEQD2 is at most* $\lceil 1/(1-\epsilon+\log\log n/\log n)\rceil$.

Proof. Clearly, an upper bound (that is achievable) for k_1 is $\lceil n/\lceil n^\gamma \rceil \rceil \le \lceil n^{1-\gamma}\rceil$.

Hence, the number of stages is

$$\begin{aligned}
c &\le \lceil \log_\alpha k_1 \rceil \\
&= \lceil \log_{\lceil n^{1-\epsilon}\log n\rceil} k_1 \rceil \\
&\le \lceil \log_{\lceil n^{1-\epsilon}\log n\rceil} \lceil n^{1-\gamma}\rceil \rceil \\
&= \lceil \log_{\lceil n^{1-\epsilon}\log n\rceil} n^{1-\gamma} \rceil \\
&\le \lceil \log_{(n^{1-\epsilon}\log n)} n^{1-\gamma} \rceil \\
&= \lceil \frac{\log n^{1-\gamma}}{\log(n^{1-\epsilon}\log n)} \rceil \\
&= \lceil \frac{(1-\gamma)\log n}{(1-\epsilon)\log n + \log\log n} \rceil \\
&= \lceil \frac{1-\gamma}{1-\epsilon+\log\log n/\log n} \rceil \\
&< \lceil \frac{1}{1-\epsilon+\log\log n/\log n} \rceil .
\end{aligned}$$

Corollary 1 *The number of stages in Algorithm SEQD2 is at most* $\lceil 1/(1-\epsilon)\rceil$.

Lemma 5 *The time complexity of Algorithm SEQD2 is* $O(n\log n/(1-\epsilon))$.

Proof. Step 1, the preprocessing step, takes $O(n\log n)$ time[8]. The time taken by Step 2 is computed as follows. The cost of each call $SEQD1(S_j)$ when $|S_j|$ is largest, i.e., when $|S_j| = n^{m\gamma} =$

$n^{\lceil 2+\sqrt{\log\log n}\rceil\gamma}$, is at most

$$\begin{aligned}
& O(|S_j| \log^2 |S_j| / \log\log |S_j|) \\
= \ & O(n^{m\gamma} \log^2 n^{m\gamma} / \log\log n^{m\gamma}) \\
= \ & O(m^2\gamma^2 \ n^{m\gamma} \log^2 n / \log(m\gamma \log n) \\
= \ & O(m^2 n^{m\gamma} \log n / \log(m\sqrt{\log n}) \\
= \ & O(m^2 n^{m\gamma} \log n / \log\log n) \\
= \ & O(\lceil 2 + \sqrt{\log\log n}\rceil^2 n^{m\gamma} \log n / \log\log n) \\
= \ & O(n^{m\gamma} \log n).
\end{aligned}$$

It follows that the time taken by each call $SEQD1(S_j)$ is $O(|S_j| \log n)$, and hence the overall time taken by Step 2 is at most $\sum_{j=1}^{k_1} O(|S_j| \log n) = O(n \log n)$. Step 4.a takes a total of $O(n)$ time. Now we compute the time taken by Step 4.b.ii to construct the Voronoi diagrams *in the first stage.* Let $h_j, 1 \leq j \leq k_1$, denote $|CH(S_j)| \leq n^\gamma$ and $w = \lceil k_1/\alpha \rceil$. There are w Voronoi diagrams to be constructed in the first stage, namely $\mathcal{FVD}(T_{k_1+1}), \mathcal{FVD}(T_{k_1+2}), \ldots, \mathcal{FVD}(T_{k_1+w})$. The total cost of computing these Voronoi diagrams is proportional to

$$\begin{aligned}
& (h_1 + h_2) + (h_1 + h_2 + h_3) + \ldots + (h_1 + h_2 + \ldots + h_\alpha) \\
+ \ & (h_{\alpha+1} + h_{\alpha+2}) + (h_{\alpha+1} + h_{\alpha+2} + h_{\alpha+3}) + \ldots + (h_{\alpha+1} + h_{\alpha+2} + \ldots + h_{2\alpha}) \\
& \vdots \\
+ \ & (h_{(w-1)\alpha+1} + h_{(w-1)\alpha+2}) + (h_{(w-1)\alpha+1} + h_{(w-1)\alpha+2} + h_{(w-1)\alpha+3}) + \ldots \\
& +(h_{(w-1)\alpha+1} + h_{(w-1)\alpha+2} + \ldots + h_{k_1}) \\
< \ & \alpha \sum_{j=1}^{k_1} h_j = \alpha h \\
= \ & \lceil n^{1-\epsilon} \log n \rceil n^\epsilon \\
= \ & O(n \log n).
\end{aligned}$$

Clearly, the cost of computing the Voronoi diagrams in the ith stage, $i > 1$, is no more that their computation in the first stage as the number of subsets decreases monotonically when going from one stage to the next higher stage. Consequently, the overall time taken by Step 4.b.ii to compute the Voronoi diagrams is $O(cn \log n)$, where c is the number of stages. The cost of point locations is $O(n \log h) = O(\epsilon n \log n)$ per stage for a total of $O(c\epsilon n \log n)$. Finally, The overall time taken by Step 4.c is clearly $O(n)$. By Corollary 1, it follows that the running time of the entire algorithm is $O(cn \log n) = O(n \log n/(1 - \epsilon))$.

By Lemma 5, the running time of Algorithm $SEQD2$ is optimal in the case of dense point sets. Figure 1 shows an extreme instance in which $h = n$ or , equivalently, $\epsilon = 1$. Applying Algorithm $SEQD2$ to this instance results in a running time of $\Theta(n \log^2 n / \log \log n)$ since, by Lemma 4, the number of stages in Algorithm $SEQD2$ is at most $\lceil 1/(1 - \epsilon + \log \log n / \log n) \rceil = \lceil \log n / \log \log n \rceil$. This exemplifies our remark before that the performance of Algorithm $SEQD2$ reduces to that of $SEQD1$ in the worst case. In this instance, for each point p_i in $S - \{p_1, p_n\}, f(p_i, S) = p_{i-1}$. Thus, the behavior of the algorithm *may* degrade when applied to sparse point sets, i.e., sets in which h (as computed by the algorithm) and n are comparable in magnitude. However, this is never the case in almost all practical instances in which the existence of a reasonably small ϵ is very natural. In the bound given by Lemma 4, we observe that the larger the value of n the smaller the ratio $\log \log n / \log n$. However, naturally, as n increases the value of ϵ decreases. As a result, for sufficiently large n, the bound on the number of stages given by Corollary 1 becomes very small in almost all practical instances, which implies that the algorithm runs in optimal $O(n \log n)$ time.

As to the work space needed, Algorithm $SEQD2$ clearly uses no more than $O(n)$ space. The following theorem summarizes our main result regarding the computation of the sequence of diameters of a given point set.

Theorem 1 *The sequence of diameters of a set of n points in the plane can be computed in $O(n \log^2 n / \log \log n)$ time and $O(n)$ space. If for some $\epsilon, 0 < \epsilon < 1, h = O(n^\epsilon)$, where h is as defined in Algorithm $SEQD2$, then the time complexity is $O(n \log n / (1 - \epsilon))$. Thus, if ϵ is reasonably small, the algorithm runs in optimal $O(n \log n)$ time.*

Proof. Direct from Lemma 5 and the discussion following Lemma 5.

4 The sum of diameters problem

We now turn our attention to the sum of diameters problem. The algorithm presented in this section is basically a refinement of the basic algorithm of Hershberger[4] that runs in $O(n \log^2 n / \log \log n)$.

Let S be a set of n points in the plane and p and q a diametral pair in S. A bipartition $\{S_1, S_2\}$ of S will be referred to as *feasible* if $Diam(S_1) + Diam(S_2) \leq d(p, q)$.

The following Lemma, which is due to Hershberger[4], provides the basis for an efficient algorithm.

Lemma 6 *Let p and q be a diametral pair of S. In any feasible bipartition of S, the subsets are contained in two disjoint disks centered on p and q.*

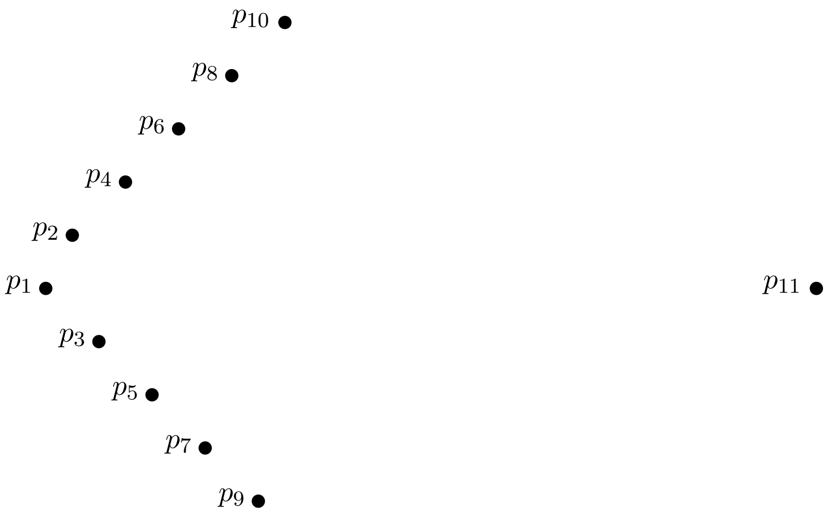

Figure 1: An extreme instance in which the running time is $\Omega(n \log^2 n / \log\log n)$.

4.1 Review of Hershberger's algorithm

Hershberger's algorithm for computing an optimal bipartition $\{S_p, S_q\}$ is summarized as follows. First, two points p and q realizing the diameter of S are computed. Next, the points in $S - \{p, q\}$ are sorted into two lists L_p and L_q, one sorted by increasing distance from p and the other by increasing distance from q. Lemma 6 implies that for any feasible bipartition $\{S_p, S_q\}$, the points in S_p must form a prefix of L_p and a suffix of L_q. The algorithm identifies all prefixes of L_p whose elements form a suffix of L_q by first marking each element of L_p with its rank in L_q and then, starting with an empty array corresponding to the list L_p, the algorithm marches through the elements of L_p, at each step marking the array entry given by the element's rank in L_q. Whenever a suffix of the array is marked, the algorithm detects it using the union-find algorithm. Finally, Using the farthest-point Voronoi diagram as the basis for Bently and Saxe logarithmic method [1], the elements of $L_p(L_q)$ are inserted into $S_p(S_q)$ in order, recording the diameter of $S_p(S_q)$ as it changes. For each prefix of L_p whose elements form a suffix of L_q, the two corresponding diameters are added and a bipartition with a minimum diameter sum is chosen. This results in a total cost of $O(n \log^2 n)$. Using a general technique of Mehlhorn and Overmas [7] that allows amortized query and insertion times of $O(\log^2 n / \log\log n)$, the bound is improved to $O(n \log^2 n / \log\log n)$. The remaining part of the paper, which constitutes the bulk of it, is devoted to the refinement of the basic algorithm. In this refinement, it is shown that if the ratio between the diameter and the minimum inter-point distance is polynomial in n, then a bound of $O(n \log n)$ is achievable. Equivalently, the refined algorithm finds an approximate solution to within a factor of $(1 + O(n^{-c}))$ in $O(n \log n)$ time.

4.2 The revised algorithm

From the description of Hershberger's algorithm, it is easy to see that if the two diameter sequences corresponding to the two sorted lists L_p and L_q are computed *a priori*, then the minimum diameter sum can be computed directly. In other words, finding the minimum diameter sum reduces (in $O(n \log n)$ time) to the computation of two diameter sequences. Using Algorithm $SEQD1$ to find these diameter sequences results in a simple algorithm that has a worst case running time of $O(n \log^2 n / \log \log n)$ without the use of dynamic data structures. On the other hand, if Algorithm $SEQD2$ is used instead, then the bound becomes optimal $O(n \log n)$ in almost all practical instances as explained in the previous section. We also state a simple observation that eliminates the need for the union-find algorithm. The revised algorithm is as follows.

Let p and q be a diametral pair in S. Let $L_p = p_1, p_2, \ldots, p_n$ be the set of points in S sorted in increasing distance from p. Let $L_q = q_1, q_2, \ldots, q_n$ be the set of points in S sorted in decreasing distance from q, where $p_1 = q_1 = p$ and $p_n = q_n = q$. Let $\{S_p, S_q\}$ be a feasible bipartition of S. By Lemma 6, any feasible bipartition of S into S_p and S_q is such that S_p is a prefix of *both* L_p and L_q. It follows that for some $k, 1 \leq k \leq n-1$,

$$S_p = \{p_1, p_2, \ldots, p_k\} = \{q_1, q_2, \ldots, q_k\}$$

and

$$S_q = \{p_{k+1}, p_{k+2}, \ldots, p_n\} = \{q_{k+1}, q_{k+2}, \ldots, q_n\}.$$

To test for this set equality, we use the following simple observation:

Observation 1

$$\{p_1, p_2, \ldots, p_k\} = \{q_1, q_2, \ldots, q_k\} \text{ if and only if } \max\{i \mid q_j = p_i, 1 \leq j \leq k\} = k.$$

Therefore, it suffices to keep track of the maximum rank of the elements $\{q_1, q_2, \ldots, q_k\}$ in L_p instead of employing the union-find algorithm as in[4]. The revised algorithm, which we will call $SUMD$, is described below:

```
procedure $SUMD$

1. Find a diametral pair $p$ and $q$ in $S$.
```

```
2. Let $L_p = p_1, p_2, \ldots, p_n$ be the list of points in $S$ sorted in increasing distance from $p$. Let $L_q = q_1, q_2, \ldots, q_n$ be the list of points in $S$ sorted in decreasing distance from $q$, where $p_1 = q_1 = p$ and $p_n = q_n = q$. Let $L'_q = q'_1, q'_2, \ldots, q'_n$ be the reverse of list $L_q$,i.e., the list of points in $S$ sorted in increasing distance from $q$.

3. Compute the two diameter sequences $d_1, d_2, \ldots, d_n$ and $d'_1, d'_2, \ldots, d'_n$ corresponding to the two sorted lists $L_p$ and $L'_q$, respectively.

4. For each $j, 1 \leq j \leq n$, find $\lambda(q_j) = i$ such that $q_j = p_i$. Here, $\lambda(q_j)$ is the rank of $q_j$ in $L_p$.

5. Set $minSum = d'_{n-1}, \lambda_{max} = 1$.

6. For each $k, 2 \leq k \leq n-1$, do the following:

    (a) Set $\lambda_{max} = \max\{\lambda_{max}, \lambda(q_k)\}$.
    (b) If $\lambda_{max} = k$, then set $minSum = \min\{minSum, d_k + d'_{n-k}\}$.

7. Return $minSum$.

end SUMD
```

Step 4 computes the rank of each point q_j in the sorted list L_p. Step 5 initializes the bipartition to $\{\{p_1\}, S - \{p_1\}\}$. Step 6 scans the list L_p detecting ordered subsets of L_p that are prefixes of both L_p and L_q. The maximum rank, λ_{max}, is updated in each iteration. By Observation 1, if $\lambda_{max} = k$, then this is an indication that a prefix has been detected, in which case the minimum sum of diameters is updated. Since the ordering of the points in L_q is the reverse of that in L'_q, the bipartition

$$\{\{p_1, p_2, \ldots, p_k\}, \{p_{k+1}, p_{k+2}, \ldots, p_n\}\}$$

can be rewritten as

$$\{\{p_1, p_2, \ldots, p_k\}, \{q_{k+1}, q_{k+2}, \ldots, q_n\}\}$$

or

$$\{\{p_1, p_2, \ldots, p_k\}, \{q'_{n-k}, q'_{n-k}, \ldots, q_1\}\}$$

which is

$$\{\{p_1, p_2, \ldots, p_k\}, \{q'_1, q'_2, \ldots, q'_{n-k}\}\}.$$

Therefore, the corresponding diameter sum for this bipartition is $d_k + d'_{n-k}$, which justifies Step 6.b.

The correctness of Algorithm *SUMD* follows directly from Lemmas 6, Observation 1 and the above discussion. The running time of the algorithm is dominated by Step 3 of finding the two diameter sequences, which is the bound given by Theorem 1. Consequently, a worst case bound that is superior to $O(n \log^2 n/\log\log n)$ is achieved in almost all practical instances in which the point set is not sparse. The only situation in which the algorithm runs in $\Omega(n \log^2 n/\log\log n)$ is in the very special case in which $h \approx n$, where h is as computed in Algorithm $SEQD2$. If this is not the case, then the time complexity is $O(n \log n/(1-\epsilon)), 0 < \epsilon < 1$, with a bound of $O(n \log n)$ being guaranteed for the case of dense point sets. As to the work space needed, the algorithm clearly uses no more than $O(n)$ space. To see that this problem is $\Omega(n \log n)$, we note that the problem of computing the diameter of a point set, which is $\Omega(n \log n)$, trivially reduces to this problem. Specifically, to find the diameter of a set of points S, we may apply the above algorithm to the instance $S \cup \{p\}$, where p is a point at infinity, in the obvious way. The following theorem summarizes our main result regarding the bipartitioning of a point set to minimize the sum of their diameters.

Theorem 2 *The bipartition of a set of n points S in the plane into two subsets S_p and S_q such that $Diam(S_p)+Diam(S_q)$ is minimum can be computed in $O(n \log^2 n/\log\log n)$ and $O(n)$ space. If for some $\epsilon, 0 < \epsilon < 1, h = O(n^\epsilon)$, where h is as defined in Algorithm $SEQD2$, then the time complexity is $O(n \log n/(1-\epsilon))$. Thus, if ϵ is reasonably small, the algorithm runs in optimal $O(n \log n)$ time.*

5 Conclusion

We have given two algorithms, $SEQD1$ and $SEQD2$, for the sequence of diameters problem (SEQDP) that have running times of $O(n \log^2 n/\log\log n)$ and $O(n \log n/(1-\epsilon))$, respectively, in the worst case. This implies two algorithms with the same time complexities for the sum of diameters problem (SUMDP). The number $\epsilon, 0 < \epsilon < 1$, that appears in the time complexity of Algorithm $SEQD2$ is a function of the density of the point set. In the extreme case, when $\epsilon \approx 1$, the bound degenerates into $O(n \log^2 n/\log\log n)$. However, the bound is optimal $O(n \log n)$ in the case of nonsparse point sets. It remains open whether these problems can be solved in $O(n \log n)$ time regardless of the density of the point set. Hershberger's $O(n \log n)$ time algorithm[4] gives a correct solution to SUMDP only if the ratio between the diameter and the minimum inter-point distance of the point set is polynomial in n. In other words, if this condition is not met, then it only gives an approximation to the optimal solution. The inability of both this algorithm and ours to achieve optimality regardless of the precision of the point coordinates and the density of the point set indicates that, if an $O(n \log n)$ algorithm exists without restrictions, another totally different approach may be inevitable. Since problem SUMDP reduces (in $O(n \log n)$ time) to SEQDP, an

$O(n \log n)$ algorithm for the latter implies an algorithm with the same time complexity for the former. However, it does not seem obvious how an $O(n \log n)$ algorithm for problem SEQDP *without restrictions* can be achieved, if one exists. An interesting question is the following: given a set S of n points in the plane together with $\mathcal{FVD}(S)$, the farthest-point Voronoi diagram of S, is it possible to find a farthest neighbor for each point in S in $O(n)$ time? This is a restricted version of the points-in-regions problem[2] in which all regions are unbounded. If this is possible, then an $O(n \log n)$ time complexity for problem SEQDP (and hence SUMDP) is straightforward. We leave it as a conjecture that the sequence of diameters problem is $\Omega(n \log^2 n / \log \log n)$ and this bound is achievable (see Figure 1.)

References

[1] J. Bentley and J. Saxe, "Decomposable searching problems I. Static-to-dynamic transformation," J. Alg., 1, 1980, 301-358.

[2] G. Blankenagel and R. H. Guting, "Internal and external algorithms for the points-in-regions problem," Lecture notes in computer science 333, Springer, Berlin, 1988.

[3] B. Chazelle, " An optimal algorithm for intersecting three-dimensional convex polyhedra," Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, 1989, 586-591.

[4] J. Hershberger , "Minimizing the sum of diameters efficiently," *Computational Geometry: Theory and Applications* 2, 1992, 111-118.

[5] D. Kirkpatric, "Efficient computation of continuous skeleton," Proceedings of the 20th Annual IEEE Syposium on Foundations of Computer Science, 1979, 18-27.

[6] C. Monma and S. Suri , "Partitioning points and graphs to minimize the maximum or the sum of diameters," *Proceedings of the Sixth International Conference on the Theory and Applications of Graphs* (Wiley, New York, 1989).

[7] M. Overmas, "The design of dynamic data structures," Lecture notes in computer science 156, Springer, Berlin, 1983.

[8] F. P. Preparata and M. I. Shamos, "Computational Geometry - An Introduction ," (Springer. Berlin, 1985).