REMOVING OUTLIERS TO MINIMIZE AREA AND PERIMETER*

Rossen Atanassov Pat Morin Stefanie Wuhrer

ABSTRACT. We consider the problem of removing c points from a set S of n points so that the resulting point set has the smallest possible convex hull. Our main result is an $O\left(n\left(\binom{4c}{2c}(3c)^c + \log n\right)\right)$ time algorithm that solves this problem when "smallest" is taken to mean least area or least perimeter.

1 Introduction

Motivated by the problem of removing outliers in a data set, this paper considers the following problem: Let S be a set of n points in $\mathbb{R}^2$ with convex hull denoted by $\mathrm{CH}(S)$. We consider the problem of selecting a subset $S' \subseteq S$, $|S'| = c$ such that the area, or perimeter, of $\mathrm{CH}(S \setminus S')$ is minimum. We call these problems the *area-based*, respectively, *perimeter-based*, *outlier removal problems*. We are particularly interested in the case when c (the number of outliers) is small.

Previous Work. The outlier removal problems stated above and similar problems are fairly well-studied problems in computational geometry. However, most work thus far has focused on the case when c is large. More specifically, most research has been on the problem of finding a k point subset (a k-cluster) $X \subseteq S$, $|X| = k$, such that $\mathrm{CH}(X)$ has minimum area or perimeter. These are the same problems studied in the current paper with $k = n - c$ except that our focus is on large values of k (small c) and most existing research focuses on designing efficient algorithms for small values of k.

The problem of finding a subset of S of size k that has the least perimeter convex hull was first considered over 20 years ago by Dobkin *et al* [3] who gave an $O(k^2 n \log n + k^5 n)$ time algorithm. This algorithm can be improved to run in $O(k^2 n \log n + k^4 n)$ time using techniques of Aggarwal *et al* [1]. Both algorithms are based on the fact that the k points that define the solution are a subset of the k' points that define some cell in the order k' Voronoĭ diagram, where $k' = \lceil \pi(k-1) \rceil$. This allows the problem to be solved by considering $O(k'n) = O(kn)$ subproblems each of size $k' = O(k)$.

The problem of finding a subset of S of size k that has the minimum area convex hull was considered by Eppstein *et al* [6] and later by Eppstein [5] who give $O(kn^3)$ and $O((k^3 + \log n)n^2)$ time algorithms for this problem, respectively. The second algorithm (by Eppstein) uses the first algorithm along with the fact that the k points that define the solution are among the $2k - 4$ vertical nearest neighbours of one of the $\binom{n}{2}$ line segments determined by pairs of points in S. This allows the problem to be solved by considering $O(n^2)$ subproblems each of size $O(k)$.

Note that finding a subset $X \subseteq S$ of size k whose convex hull has minimum area generalizes the problem of determining if S contains k collinear points (in which case S has a subset of size k whose

*This work was partly funded by NSERC.

convex hull has area 0). For the case $k = 3$, this problem is one of the original 3-SUM-hard problems [7] and is conjectured to have an $\Omega(n^2)$ lower-bound in many models of computation. However, for large values of k, the problem seems to become easier, and Guibas *et al* [9] have an algorithm that reports all lines containing at least k points of S in $O((n^2/k)\log(n/k))$ time. Notice that, for $k = \Omega(n)$, this gives an $O(n)$ time algorithm, which offers hope that outlier removal problems should have fast solutions for sufficiently small values of c.

New Results. For fixed values of k, the results above give $O(n \log n)$ and $O(n^2)$ time algorithms for finding the k point subset of S with minimum perimeter, respectively, area, convex hull. However, when k is close to n the above algorithms require $\Omega(n^3)$ time. In the current paper we consider specifically the case when $k = n - c$ and show that perimeter-based and area-based outlier removal problems can be solved in $O\left(n\left(\binom{4c}{2c}(3c)^c + \log n\right)\right)$ time. Thus, for any fixed c, both problems can be solved in $O(n \log n)$ time.

We observe that, in the algebraic decision tree model of computation, $\Omega(n \log n)$ is a lower bound for outlier removal problems even when $c = 1$. This is because the problem, SET-EQUALITY, of determining whether two sets A and B of real numbers are equal has an $\Omega(n \log n)$ lower bound in the algebraic decision tree model. The SET-EQUALITY problem can be mapped to an outlier removal problem on the point multiset $S = S_A \uplus S_B$ where $S_A = \{(x, x^2) : x \in A\}$, $S_B = \{(x, x^2) : x \in B\}$ and $\uplus$ denotes multiset union. The sets A and B are equal if and only if for every $p \in S$, $\mathrm{CH}(S) = \mathrm{CH}(S \setminus \{p\})$. Thus (in a very weak sense) the running time of our algorithm is optimal, at least in terms of its dependence on n.

The remainder of the paper is organized as follows: Section 2 defines the notation and some previous results that are used in subsequent sections. Section 3 describes the outlier removal algorithm. Section 4 concludes and suggests directions for future research.

2 Preliminaries

The *convex layers* $S_0, \ldots, S_k$ of S are defined as follows: S_0 is the subset of S on the boundary of $\mathrm{CH}(S)$. S_i, for $i \geq 1$ is the subset of S on the boundary of $\mathrm{CH}(S \setminus \bigcup_{j=0}^{i-1} S_j)$. The convex layers of S can be computed in $O(n \log n)$ time [2, 10] or, more simply, the first c convex layers can be computed in $O(cn \log n)$ time by repeated applications of any $O(n \log n)$ time convex hull algorithm. For the remainder of this paper we will use the notation $p_{i,j}$ to denote the $(j \bmod |S_i|)$th point of S_i, and use the convention that $p_{i,0}, \ldots, p_{i,|S_i|-1}$ occur in counterclockwise order on the boundary of $\mathrm{CH}(S_i)$.

Once the convex layers $S_0, \ldots, S_c$ have been computed, we can find, in $O(c^2 n)$ time, for each point $p_{i,j}$ on layer i and for each layer $i' > i$ and $i' \leq c$ the two points $p_{i',k}$ and $p_{i',\ell}$ such that the line through $p_{i,j}$ and $p_{i',k}$ (respectively $p_{i,j}$ and $p_{i',\ell}$) is tangent to $S_{i'}$. This is accomplished by a simple walk around S_i, updating tangents $p_{i',k}$ and $p_{i',\ell}$ as we proceed.

Consider a point $p_{0,j} \in S_0$ and refer to Figure 1. If we remove $p_{0,j}$ from S then a (possibly empty) sequence $p_{1,k}, \ldots, p_{1,\ell}$ of S_1 appears on the boundary of $\mathrm{CH}(S \setminus \{p_{0,j}\})$. When this happens we say that $p_{1,k}, \ldots, p_{1,\ell}$ is *exposed*. This exposed sequence can be obtained from the preprocessing described above by using two tangents $p_{1,k}$ and $p_{1,\ell}$ joining $p_{0,j-1}$ and $p_{0,j+1}$ to S_1. Finding the two tangent points takes $O(1)$ time and traversing the sequence takes $O(t_j)$ time, where $t_j = \ell - k + 1$.

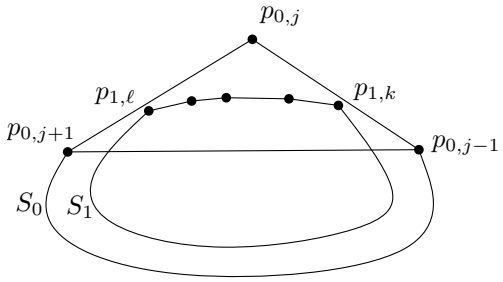

Figure 1: Removing a point $p_{0,j}$ from S_0 exposes a chain $p_{1,\ell}, \ldots, p_{1,k}$ of S_1.

Once we have removed a point $p_{0,j}$ from S_0, if we know the area (or perimeter) of $\mathrm{CH}(S)$ we can compute the area (or perimeter) of $\mathrm{CH}(S \setminus \{p_{0,j}\})$ in $O(t_j)$ time. We do this by computing the area of the triangle $\triangle p_{0,j-1}p_{0,j}p_{0,j+1}$ and subtracting from it the area of $\mathrm{CH}(\{p_{0,j-1}, p_{0,j+1}\} \cup \{p_{1,k}, \ldots, p_{1,\ell}\})$. This gives us the difference in area (perimeter) between $\mathrm{CH}(S)$ and $\mathrm{CH}(S \setminus \{p_{0,j}\})$.

3 The Algorithm

In this section we present our algorithm for solving the perimeter-based and area-based outlier removal problems. Our solution to both problems is to enumerate all the combinatorial types of solutions of size c. For each such solution type, we then use a combination of divide-and-conquer and dynamic programming to find the optimal solution of that particular solution type. Before we present the general algorithm, it will be helpful to discuss the special cases $c = 1$ and $c = 2$ to illustrate the principles involved.

3.1 Removing 1 Outlier

The case $c = 1$ asks us to remove 1 point of S so that the convex hull of the resulting set is minimum. This can be solved as follows: We first compute the two convex layers S_0 and S_1 in $O(n \log n)$ time and preprocess them for the tangent queries described in the previous section. We then determine, for each point $p_{0,j} \in S_0$ the difference in area between $\mathrm{CH}(S)$ and $\mathrm{CH}(S \setminus \{p_{0,j}\})$ using the method described in the previous section. This process takes $O(1 + t_j)$ time, where t_j is the number of vertices of S_1 exposed by the removal of $p_{0,j}$. We output the point $p_{0,j}$ that gives the largest difference in area.

To analyze the overall running time of this algorithm we observe that any point $p_{1,k} \in S_1$ appears in at most two triangles $\triangle p_{0,j-1}, p_{0,j}, p_{0,j+1}$ and $\triangle p_{0,j}, p_{0,j+1}, p_{0,j+2}$. Stated another way,

$$\sum_{j=0}^{|S_0|-1} t_j \leq 2|S_1| \leq 2n \ .$$

Thus, the overall running time of this algorithm is

$$T(n) = O(n \log n) + \sum_{j=0}^{|S_0|-1} O(1 + t_j) = O(n \log n) \ ,$$

as claimed.

3.2 Removing 2 Outliers

Next we consider the case $c = 2$. In this case, the optimal solution S' is of one of three following forms:

1. S' contains two consecutive points $p_{0,j}$ and $p_{0,j+1}$ of S_0.
2. S' contains two non-consecutive points p_{0,j_1} and p_{0,j_2} of S_0 (with $j_2 \notin \{j_1 + 1, j_1 - 1\}$).
3. S' contains one point $p_{0,j}$ of S_0 and one point $p_{1,j'}$ of S_1.

The solutions of Type 1 can be found in much the same way as the algorithm for the case $c = 1$. For each $j \in \{0, \ldots, |S_0|-1\}$ we compute the difference in area between $\mathrm{CH}(S)$ and $\mathrm{CH}(S \setminus \{p_{0,j}, p_{0,j+1}\})$. The analysis remains exactly the same as before except that, now, each point of S_1 can appear in at most 3 area computatons, instead of only 2. Thus, all solutions of Type 1 can be evaluated in $O(n \log n)$ time.

The solutions of Type 3 can also be found in a similar manner. For each point $p_{0,j} \in S_0$ we remove $p_{0,j}$ to expose a sequence $p_{1,k}, \ldots, p_{1,\ell}$ of S_1 and compute the area of $\mathrm{CH}(S \setminus \{p_{0,j}\})$. We then remove each of $p_{1,k}, \ldots, p_{1,\ell}$ in turn (exposing a chain of points from S_2) and compute the area of the resulting convex hull. To analyze the cost of all these, we observe that each point $p_{1,j} \in S_1$ appears in at most 2 subproblems because there are at most 2 points in S_0 whose removal causes $p_{1,j}$ to appear on the convex hull. Similarly, for each point $p_{2,j} \in S_2$ there are at most 2 points of S_1 whose removal causes $p_{2,j}$ to appear on the convex hull. Thus, each point in S_1 appears in at most 2 subproblems and each point in S_2 appears in at most 4 area computations. The overall running time of this algorithm is therefore bounded by

$$O\left(n \log n + |S_0| + 2|S_1| + 4|S_2|\right) = O(n \log n) \ ,$$

as required.

Finally, we consider solutions of Type 2. To find these we compute, for each $p_{0,j} \in S_0$ the difference x_j between the area of $\mathrm{CH}(S \setminus \{p_{0,j}\})$ and $\mathrm{CH}(S)$ using the technique described for the case $c = 1$. In this way, we reduce the problem to that of finding two indices $0 \leq j_1, j_2 < |S_0|$ with $j_2 \geq j_1 + 2$ such that $x_{j_1} + x_{j_2}$ is maximum. We do this by computing the following quantity

$$D_j = \max\{x_{j_1} + x_{j_2} : 0 \leq j_1, j_2 \leq j \text{ and } j_2 \geq j_1 + 2\} \ ,$$

that can be computed in $O(|S_0|)$ time using the recurrence

$$D_j = \max\{D_{j-1}, x_j + \max\{x_0, \ldots, x_{j-2}\}\} \ .$$

Since the best solution of each of the three types can be found in $O(n \log n)$ time we can find the overall best solution in $O(n \log n)$ time by keeping the best of the three.

3.3 Removing c Outliers

The solution for the case $c = 2$ illustrates all of the ideas used in our algorithm. We begin by enumerating the combinatorial types of solutions and then compute the best solution of each type. The algorithm for computing the best solution of each type is a divide-and-conquer algorithm whose merge step is accomplished by solving a dynamic programming problem (as in the Type 2 solutions described above).

For ease of exposition (to avoid treating S_0 as a special case), we describe an algorithm to find the optimal solution with the restriction that it does not include both $p_{0,-1}$ and $p_{0,0}$. To find the true optimal solution we can run this algorithm at most c times, shifting the numerical labelling of the points on S_0 by one each time.

3.3.1 The Types of Solutions

We represent the type of a solution as a rooted ordered binary tree in which each node is labeled with a positive integer and the sum of all node labels is c. We call such trees *solution trees*. The following lemma tells us that, for small values of c, there are not too many solution trees:

Lemma 1. *The number of solution trees is at most* $(1 + o(1))C(2c)$*, where* $C(r) = \binom{2r}{r}/(r+1)$ *is the* r*th Catalan number.*

Proof. Given a solution tree T, we convert it to an unlabelled rooted binary tree as follows: First, for each node N of T whose label is ℓ, we color N black, leave N's left child unchanged and replace N's right child with a (right) path of $\ell - 1$ *white* nodes, the last node of which has N's original right child as its right child. Note that this gives us a binary tree with exactly c nodes in which no white node has a left child. Next, for each black node we add a leaf, if necessary, to guarantee that each black node has a left child. This gives us an unlabelled rooted ordered binary tree T' with at most $2c$ nodes.

Observe that, given only T', we can recover T and its labels since we can recognize white nodes by the fact that they have no left child. Thus, the number of solution trees is at most equal to the number of unlabelled rooted ordered binary trees with at most $2c$ nodes. The number of such trees is

$$\sum_{i=1}^{2c} C(i) = (1 + o(1))C(2c)$$

[8], as required. □

Solution trees are interpreted as follows (refer to Figure 2 for an example): Any solution removes some elements of S_0 and the elements removed come in d groups $G_0^1, \ldots, G_0^d$ of consecutive elements with each group separated by at least one element of S_0. The sizes of these groups are given by the labels of the nodes on the rightmost path in T, in the order in which they occur. That is, the jth node, N_0^j on the right most path of T has the label $|G_0^j|$.

For some group G_0^j, let $p_{1,k}, \ldots, p_{1,\ell}$ denote the points of S_1 that appear on the boundary of $\mathrm{CH}(S \setminus G_0^j)$. Any solution removes some subset $p_{1,k}, \ldots, p_{1,\ell}$ of elements from S_1. Again, this subset can be partitioned into groups of consecutive elements with any two groups separated by at least one

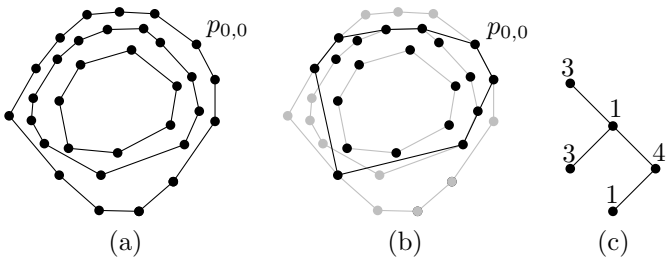

Figure 2: Examples of (a) a point set S, (b) a solution $S \setminus \{S'\}$, and (c) the solution tree for S'.

element of S_1. In the solution tree T, the sizes of these groups are given, in the order in which they occur, by the labels of the rightmost path in the subtree of T rooted at the left child of N_0^j.

This process is repeated recursively: Let $S_{<i} = \bigcup_{j=0}^{i-1} S_i$ and let $S'_{<i} = S_{<i} \cap S'$. For each consecutive group G_i^j of nodes that are removed from S_i, let $p_{i+1,k}, \ldots, p_{i+1,\ell}$ denote the vertices on S_{i+1} that appear on the boundary of $CH(S \setminus (S'_{<i} \cup G_i^j))$. In the solution tree T, the rightmost path of the left subtree of the node representing G_i^j contains nodes representing the sizes of consecutive groups of nodes that are removed from the chain $p_{i+1,k}, \ldots, p_{i+1,l}$ of S_{i+1}. In this way, any solution S' to the outlier removal problem that does not remove both $p_{0,0}$ and $p_{0,-1}$ maps to a unique solution tree.

3.3.2 Computing the Solution of a Specific Type

In this section we describe an algorithm that takes as input a solution tree T as outputs the value of the optimal solution S' whose solution tree is T.

The algorithm we describe is recursive and operates on a subchain $p_{i,j}, \ldots, p_{i,k}$ of S_i along with a solution (sub)tree T. The algorithm requires that some subset of $\bigcup_{j=0}^{i-1} S_i$ has already been removed from S so that $p_{i,j}, \ldots, p_{i,k}$ are on the boundary of the convex hull of the current point set. The algorithm finds an optimal solution of type T such that the only points removed from the convex hull of the current point set are in $p_{i,j}, \ldots, p_{i,k}$.

Let d denote the number of nodes on the rightmost path of T. The algorithm accomplishes its task by recursively solving $O(d(k-j+1))$ subproblems on the left children of these d nodes and then combining these solutions using dynamic programming. The following pseudocode gives a detailed description of the algorithm's operation with the exception of the dynamic programming component, whose description and analysis is discussed in the next subsection. Note that the algorithm below only computes the maximum amount of area (perimeter) that can be removed from $\mathrm{CH}(S)$ by a solution S' whose solution tree is T. To obtain the points in S' the algorithm can be augmented using the standard trick of remembering, whenever the algorithm takes the maximum of two values, which of the two values produced the maximum. The details of this are standard and ommitted here.

FindOptimalOfType(T, i, j, k)

1: **if** T is empty **then**
2: **return** 0
3: $d \leftarrow$ the number of nodes on the rightmost path of T
4: **for** $g = 1$ to d **do**
5: $N_g \leftarrow g$th node on the rightmost path of T
6: $c_g = \text{label}(N_g)$
7: **for** $\ell = j$ to $k - c_g + 1$ **do**
8: delete $S_{i,\ell}, \ldots, S_{i,\ell+c_g-1}$ from S_i exposing $S_{i+1,j'}, \ldots, S_{i+1,k'}$ on S_{i+1}
9: $s \leftarrow$ reduction in area (perimeter) obtained by the deletion of $S_{i,\ell}, \ldots, S_{i,\ell+c_g-1}$
10: $X_{g,\ell-j+1} \leftarrow s +$ FindOptimalOfType$(\text{left}(N_g), i+1, j', k')$
11: reinsert $S_{i,\ell}, \ldots, S_{i,\ell+c_d-1}$ into S_i
12: **return** CombineSolutions$(X, d, k-j+1, c_1, \ldots, c_d)$

The call to CombineSolutions in the last line of the algorithm is a dynamic programming subroutine described in the next section that runs in $O(d(k-g))$ time. At the topmost level, the algorithm is called as FindOptimalOfType$(T, 0, 0, |S_0| - 1)$.

To analyze the cost of FindOptimalOfType it suffices to determine, for each point $p_{i,j}$, the maximum number of times $p_{i,j}$ is deleted (in line 8) by the algorithm. All other work done by the algorithm can be bounded in terms of this quantity. For points $p_{0,j} \in S_0$, each point is deleted exactly g_0, times, where g_0 is the sum of labels of nodes on the rightmost path of T.

More generally, let g_i denote the sum of labels of all nodes N of T for which the path from the root of T to N make exactly i left turns. (These nodes correspond to groups that are deleted from S_i). Consider some point $p_{i,j} \in S_i$, for $i \geq 1$. By Carathéodory's Theorem [4], each such point is contained in some triangle $\Delta_{i,j} = \triangle p_{i-1,\ell_1}, p_{i-1,\ell_2}, p_{i-1,\ell_3}$. If $p_{i,j}$ is deleted by FindOptimalOfType then it is on the boundary of the convex hull of the current point set. However, this implies that at least one of the three vertices of $\Delta_{i,j}$ must be deleted from the current point set. Thus, if we define m_i as the maximum number of times a point of S_i is deleted by FindOptimalOfType then we have the relationships:

$$m_i \leq \begin{cases} g_0 & \text{if } i = 0 \\ 3m_{i-1}g_i & \text{if } 1 \leq i < c \\ 0 & \text{otherwise} \end{cases}$$

Using the fact that $g_i \leq c$, we obtain the (extremely loose) upper bound $m_i \leq (3c)^{i+1}$. Finally, we note the points of S_c (which are never deleted) appear in at most $3m_{c-1}$ subproblems. Putting all this together we obtain:

Lemma 2. *The algorithm* FindOptimalOfType *finds the optimal solution whose solution tree is T in $O((3c)^c n)$ time.*

3.3.3 Combining the Solutions

One aspect of the algorithm that we have not yet described is how the subroutine CombineSolutions works. This subroutine is given positive integers $c_1, \ldots, c_d$ and a $d \times m$ positive real-valued matrix X and must find indices $1 \leq i_1, \ldots, i_d \leq m - c_d$ such that

$$i_{j+1} \geq i_j + c_j + 1$$

for all $1 \leq j < d$ and such that the sum

$$h(i_1, \ldots, i_d) = \sum_{j=1}^{d} X_{j,i_j}$$

is maximum. In the terminology of the previous section, the value d is the number of nodes in the rightmost path of T, $m = k - j + 1$, and the indices $i_1, \ldots, i_d$ correspond to the indices of the first element of each group on the chain $p_{i,j}, \ldots, p_{i,k}$ considered by the algorithm. We solve this problem by filling out the $d \times m$ dynamic programming table:

$$D_{j,\ell} = \max\{h(i_1, \ldots, i_j) : 1 \leq i_1, \ldots, i_j \leq \ell, \text{ and } i_{j'+1} \geq i_{j'} + c_{j'} + 1 \text{ for all } 1 \leq j' < j\}$$

for $j = 1, \ldots, d$ and $\ell = 1, \ldots, m - c_j + 1$. We can do this in $O(cn)$ time because the table entries satisfy the recurrence

$$D_{j,\ell} = \max\{D_{j,\ell-1}, D_{j-1,\ell-c_{j-1}-1} + X_j\}$$

where we use the convention that $D_{j,\ell} = 0$ if $j = 0$ and $D_{j,\ell} = -\infty$ if $\ell < 0$. This gives the last lemma required by the algorithm:

Lemma 3. *The function* CombineSolutions *can be implemented in $O(dm)$ time.*

4 Conclusions

Together with the $O(n \log n + c^2 n)$ time preprocessing described in Section 2, Lemma 1, Lemma 2 and Lemma 3 give an algorithm for the outlier removal problem whose running time is

$$T(n, c) = O(n \log n + c^2 n) + c \times C(2c) \times O((3c)^c n) = O\left(n\left(\binom{4c}{2c}(3c)^c + \log n\right)\right) \quad .$$

Stretching the use of asymptotic notation, this completes the proof of:

Theorem 1. *For any constant c, there exists an algorithm for the (perimeter-based or area-based) outlier removal problem that runs in $O(n \log n)$.*

We close with a few open problems. We have made no attempt to optimize the dependence of our running time with respect to the value of c and, indeed, the running time of our algorithm is superpolynomial in c. Does there exist an algorithm that is polynomial in c but that still runs in $O(n \log n)$ time for any fixed value of c?

The outlier removal problems we study in this paper are two special cases of the following *general outlier removal* problem. Given any function f that maps subsets of $\mathbb{R}^2$ onto $\mathbb{R}$ we want to find, for a given set S of n points in $\mathbb{R}^2$ a subset $S' \subseteq S$, $|S'| = c$, such that $f(S \setminus S')$ is minimum. Functions f of particular interest include:

1. $f(X)$ is the radius of the smallest closed disk contain X,
2. $f(X)$ is the diameter of X ($f(X) = \max\{\|ab\| : a, b \in X\}$), and
3. $f(X)$ is the variance of X ($f(X) = \sum_{a \in X} \|am\|^2$ where $m = \sum_{a \in X} a/|X|$.

For the first two functions, it is easy to obtain $O(3^c n)$ and $O(2^c n \log n)$ time algorithms, respectively, using the fact that the smallest enclosing disk and the diameter are defined by at most 3, respectively, 2, points of S. Are there algorithms for these problems whose running time depends only polynomially on c?

References

[1] Alok Aggarwal, Hiroshi Imai, Naoki Katoh, and Subhash Suri. Finding k points with minimum diameter and related problems. *Journal of Algorithms*, 12:38–56, 1991.

[2] Bernard Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31:509–517, 1985.

[3] David Dobkin, Robert Drysdale, and Leo Guibas. Finding smallest polygons. *Computational Geometry*, 1:181–214, 1983.

[4] Jürgen Eckhoff. Helly, radon, and carathéodory type theorems. In P. M. Gruber and J. M. Wills, editors, *Handbook of Convex Geometry*, volume B, chapter 2.1, pages 389–448. North-Holland, 1993.

[5] David Eppstein. New algorithms for minimum area k-gons. In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1992.

[6] David Eppstein, Mark Overmars, Günter Rote, and Gerhard Woeginger. Finding minimum area k-gons. *Discrete and Computational Geometry*, 7:45–58, 1992.

[7] Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, 5(3):165–185, 1995.

[8] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science, Second Edition*. Addison-Wesley, 1994.

[9] Leonidas J. Guibas, Mark H. Overmars, and Jean-Marc Robert. The exact fitting problem for points. *Computational Geometry: Theory and Applications*, 6:215–230, 1996.

[10] John Hershberger and Subhash Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32:249–267, 1992.