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* [4] 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* [7] and later by Eppstein [6] 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 $O(k)$.

New Results. For fixed values of k, the results above give $O(n \log n)$ and $O(n^2 \log n)$ 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.

Due to space constraints, many details and all proofs have been ommitted from this abstract. Full details can be found in the accompanying technical report [2].

The remainder of the paper is organized as follows: Section 2 presents definitions, notation, and 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 [3, 8] 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 mod $|S_i|$)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.

[*]School of Computer Science. Carleton University. This work was partly funded by NSERC.

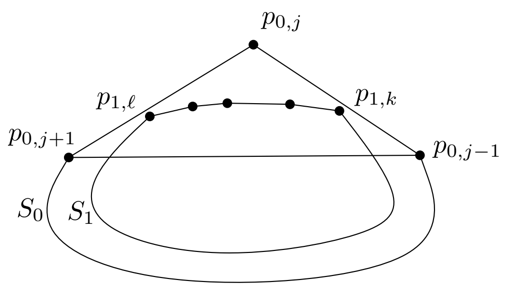

Figure 1: Removing a point $p_{0,j}$ from S_0 exposes a chain $p_{1,\ell},\ldots,p_{1,k}$ of S_1.

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 $t_j=\ell-k+1$.

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 particular 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,

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

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' has 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 $0 \le j_1, j_2 < |S_0|$ with $j_2 \ge j_1 + 2$ such that $x_{j_1} + x_{j_2}$ is maximum. We do this by computing the following quantity

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

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

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

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 $O(C(2c))$, where $C(r) = \binom{2r}{r}/(r+1)$ is the rth Catalan number.*

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 $G_0^1, \dots, G_0^d$ 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 rightmost path of T has the label $|G_0^j|$.

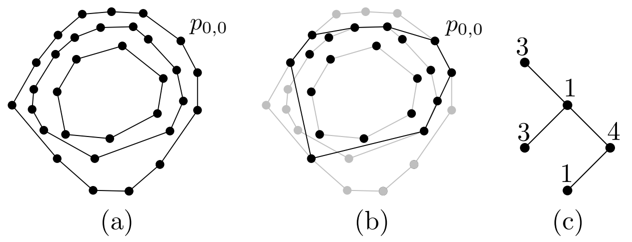

Figure 2: Examples of (a) a point set S, (b) a solution $S \setminus \{S'\}$, and (c) the solution tree for S'.

For some group G_0^j, let $p_{1,k}, \dots, p_{1,\ell}$ denote the points of S_1 that appear on the boundary of $\mathrm{CH}(S \setminus G_0^j)$. Any solution removes some subset $p_{1,k}, \dots, p_{1,\ell}$ 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 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 $p_{i+1,k}, \dots, p_{i+1,\ell}$ 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 $p_{i+1,k}, \dots, p_{i+1,l}$ 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 and outputs the value of the optimal solution S' whose solution tree is T.

The algorithm we describe is recursive and operates on a subchain $p_{i,j}, \dots, p_{i,k}$ 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 $p_{i,j}, \dots, p_{i,k}$ 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 $p_{i,j}, \dots, p_{i,k}$.

Let d denote the number of nodes on the rightmost path of T. The algorithm accomplishes its task by re-

```
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($X, d, k-j+1, c_1, \ldots, c_d$)
```

Figure 3: Pseudocode for the FindOptimalOfType algorithm.

cursively solving $O(d(k-j+1))$ subproblems on the left children of these d nodes and then combining these solutions using dynamic programming. The pseudocode for the algorithm is given in Figure 3. The call to CombineSolutions in the last line of the algorithm is a dynamic programming subroutine whose description we omit but that runs in $O(d(k-g))$ time. The CombineSolutions subroutine computes the optimal locations of the groups of points represented by $N_1, \ldots, N_d$ in T. 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. Using Carathéodory's Theorem [5], it is possible to show that a point in S_i is deleted at most $(3c)^{i+1}$ times. This implies that 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.*

4 Conclusions

Taken together, the preceding discussion completes the proof of our main theorem:

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

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?

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] Rossen Atanassov, Pat Morin, and Stefanie Wuhrer. Removing outliers to minimize area and perimeter. Technical Report TR-06-07, Carleton University School of Computer Science, 2006.

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

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

[5] 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.

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

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

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