How to Cover Most of a Point Set with a V-Shape of Minimum Width

Boris Aronov* aronov@poly.edu
John Iacono† jiacono@poly.edu
Özgür Özkan ozgurozkan@gmail.com
Mark Yagnatinsky‡ myag@cis.poly.edu

Polytechnic Institute of NYU, Brooklyn, New York

Abstract

A V-shape is an infinite polygonal region bounded by two pairs of parallel rays emanating from two vertices (see Figure 1). We describe a randomized algorithm that, given n points and an integer $k \geq 0$, finds the minimum-width V-shape enclosing all but k of the points with probability $1 - 1/n^c$ for any $c > 0$, with expected running time $O(cn^2(k+1)^4 \log n(\log n \log\log n + k))$.

1 Introduction

Motivation. The motivation for this problem comes from curve reconstruction: given a set of points sampled from a curve in the plane, find a shape approximating the original curve. It has been suggested in [AD13] that in an area where the curve makes a sharp turn, it makes sense to model the curve by a *V-shape*. The authors remark that it would be natural to investigate a variant that can handle a small number of outliers, to accommodate a few bad data points. We investigate that variant here. The problem is an instance of a large class of problems known as *geometric optimization* or *fitting* questions, (see [AS98] for a survey).

Previous work. In [AD13], the authors develop an algorithm for covering a point set in general position[1] with a V-shape of minimum width (allowing no outliers) that runs in $O(n^2 \log n)$ time and uses quadratic space. They also find a constant-factor approximation algorithm with running time $O(n \log n)$, and a $(1+\varepsilon)$-approximation algorithm with a running time of $O((n/\varepsilon)\log n + n/(\varepsilon^{3/2})\log^2(1/\varepsilon))$, which is $O(n \log n)$ for a constant $\varepsilon > 0$.

*Research supported by NSF Grants CCF-08-30691, CCF-11-17336, and CCF-12-18791, and by NSA MSP Grant H98230-10-1-0210.

†Research supported by NSF Grant CCF-1018370.

‡Research supported by NSF Grant CCF-11-17336, NSA MSP Grant H98230-10-1-0210, and GAANN Grant P200A090157 from the US Department of Education.

[1] We use the same general position assumptions as [AD13]: no vertical line goes through two points, no three points are collinear, and no lines defined by pairs of points are parallel.

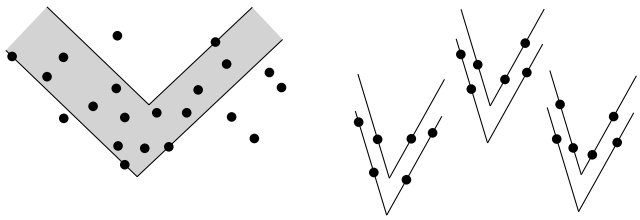

Figure 1: Left to right: a V-shape with six outliers, and both-outer, inner-outer, and both-inner V-shapes.

Our Result. Given a set P of n points in the plane and an integer $k \geq 0$, we show how to find the minimum-width V-shape enclosing all but k of the points.

Definitions and notation. A V-shape is an infinite polygonal region bounded by two pairs of parallel rays emanating from two vertices (see Figure 1). The rays on the region's convex hull are the *outer rays*. The others are the *inner rays*.

The line segment connecting the vertices of the outer and inner rays separates the V-shape V into its *left arm* and *right arm*. The *width* of an arm is the distance between its two delimiting rays. The *width* of V is the width of its wider arm. An *outlier* of V is a point of P not contained in V. Each arm has an associated *strip*, defined by the pair of directed parallel lines going through its boundary. The left and right strips together uniquely determine a V-shape. This is in fact how our algorithm works: by trying to find a pair of strips determining the thinnest V-shape.

Given a point set P, a *k-edge* (see Figure 2) of P is a directed edge between two points in the set such that exactly k points of P lie to the left of the directed line through the edge (so in general position there are $n-k-2$ points to the right). For example, a 0-edge is a directed edge on the convex hull. A k-edge is also said to be *an edge at level k*. Let $L(e,P)$ denote the level of edge e in point set P. The set $H(k,P)$ of edges at level k or less are known as the at-most-k-edges, or more concisely, the $(\leq k)$-edges.

It will also be useful to talk about levels in a line arrangement $\mathcal{A}$ (see Figure 3). We use the following definition: an edge of $\mathcal{A}$ is on the k-level if there are

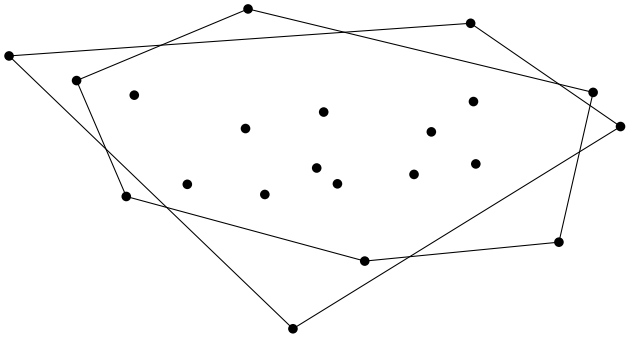

Figure 2: The edges at level 1 of a point set.

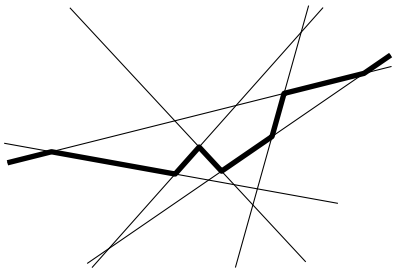

Figure 3: The 2-level of a line arrangement with six lines.

exactly k lines of $\mathcal{A}$ strictly above it.

2 The algorithm

2.1 Overview

We need only consider locally optimal V-shapes. In [AD13], it was shown that there are three types of locally optimal configurations, which they called both-outer, inner-outer, and both-inner (see Figure 1). A *both-outer* V-shape is a V-shape where both outer rays have two points on them (and the inner rays have one). A *both-inner* V-shape is a V-shape where both inner rays have two points on them (and the outer rays have one). An *inner-outer* V-shape is a V-shape where one of the outer rays and one of the inner rays has two points on it and the other two rays have one point each. (Note that even for point sets in general position, one of the arms may have its two rays coincide, and thus the V-shape will have only five points on its boundary instead of six.) A V-shape with k outliers is called a k-outlier V-shape of the point set. Our algorithm works by finding the minimum-width k-outlier V-shape of each type, and returning the one that has the smallest width of all three.

Our approach for the both-outer case and the inner-outer case was inspired by the approach of [AD13] for the inner-outer case, except we use a binary search for one step where they use total enumeration. When there are zero outliers, our algorithm for the both-outer and inner-outer cases would be easier to implement than theirs, at the cost of a logarithmic factor in the running time. However, most of the complexity of their solution was in the both-outer case, and we use their both-outer algorithm as a black box in our both-outer algorithm, by running it on random subsets of the point set.

We handle both-outer V-shapes and inner-outer V-shapes in almost the same way (see Figure 4). We begin by enumerating the $(\leq k)$-edges of the point set. Each such j-edge e is considered in turn as a candidate for one of the outer rays to go through, with $j \leq k$ outliers already accounted for. For a fixed e, we do a binary search among remaining points of P, ordered by perpendicular distance from e; this distance is the width of the first candidate strip. For each point of the search we find the second strip that has the smallest possible width and still covers the remaining points, except for the outliers. If the second strip is wider than the first, the binary search moves farther out from e so that the second strip has fewer points, otherwise it moves closer. To find the second strip, we again enumerate the edges at levels 0 through k of the remaining points. The precise definition of "remaining" here is the key difference between the both-outer and the inner-outer algorithm; see detailed discussion below. By now we have chosen three rays, and have no freedom for the fourth: it is dictated by how many more outliers we need. The running time is $O(n^2(k+1)^2 \log^2 n)$ (see Lemma 3 for proof).

To find the minimum-width both-inner k-outlier V-shape, we use a randomized algorithm that takes random samples of the given point set. For each sample, it enumerates *all* both-inner 0-outlier V-shapes using the algorithm from [AD13]. We show that with enough samples, the minimum-width both-inner k-outlier V-shape will be one of the V-shapes enumerated with probability at least $1 - 1/n^c$ for any real number $c > 0$ (given as an input parameter). The V-shapes we enumerate might have more than k outliers, so we use a range searching data structure from [CY84] to detect and discard such V-shapes. The running time of the both-inner case is $O(cn^2(k+1)^4 \log n(\log n \log\log n + k))$, which dominates the running time of the other two cases.

Theorem 1 *There is a randomized algorithm that, given n points and an integer $k \geq 0$ denoting the desired number of outliers, finds the minimum-width k-outlier V-shape for the points with probability $1 - 1/n^c$ for any $c > 0$, requiring $O(n^2)$ space with expected running time $O(cn^2(k+1)^4 \log n(\log n \log\log n + k))$.*

Proof. We find the thinnest k-outlier V-shape of each of the three types separately and return the thinnest. The both-inner algorithm dominates the running time. The running time and correctness of the algorithms handling the three cases is established in Lemmas 2, 3, and 5. □

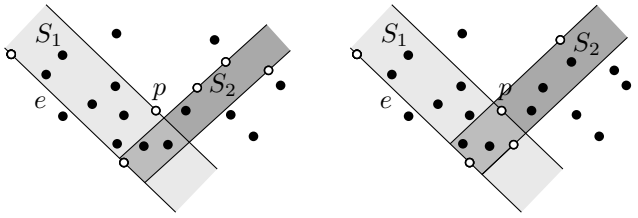

Figure 4: Snapshot of inner-outer algorithm (left) and both-outer algorithm (right).

2.2 Both-Outer and Inner-Outer

The following two algorithms find the thinnest inner-outer k-outlier V-shape and the thinnest both-outer k-outlier V-shape. The algorithms have the same structure: for each candidate first strip S_1, find the thinnest possible second strip S_2:

Algorithms 1 and 2. Part I: Finding S_1.

Input: integer k, point set P such that $|P| = n$.
For each edge $e \in H(k, P)$:
  $P' =$ points to the right of e, and the two points on e.
  Sort P' by distance from the line through e.
  //Perform binary search on P'.
  For each point p of the binary search:
    $S_1 =$ points contained in strip defined by e and p.
    Find thinnest strip S_2. //see part 2 of algorithm
    The binary search is guided by which strip is thicker:
      If S_1 is thicker, move p closer to e, else further.

It remains to explain how to find the thinnest S_2 and this depends on whether the k-outlier V-shape we seek is an inner-outer V-shape or a both-outer V-shape. First we define a function *find-line* (described in Lemma 4) that takes a directed edge e, an integer i, and point set P, and finds the $(i+1)$st furthest line from e going through a point in P right of e.

The two cases are similar, so the steps that differ are marked with an asterisk. It may help to refer to Figure 4. To find a both-outer k-outlier V-shape, we use Algorithm 1, and to find an inner-outer k-outlier V-shape, we use Algorithm 2.

Algorithm 1. Part II: Finding S_2 of a both-outer V-shape.

$i = L(e, P)$
For each edge $f \in H(k - i, P')$: *
  $j = k - i - L(f, P')$. * //number of outliers still needed
  Let $\ell =$ find-line$(f, j, P' - S_1)$.*//ℓ may not exist
  Let $S_2 =$ the strip formed by f and ℓ.
  Record the thinnest S_2 found so far.

If ℓ from Algorithm 1 does not exist, because $P' - S_1$ has less than j points, then the strip determined by S_1 is too wide, and we can proceed to the next f.

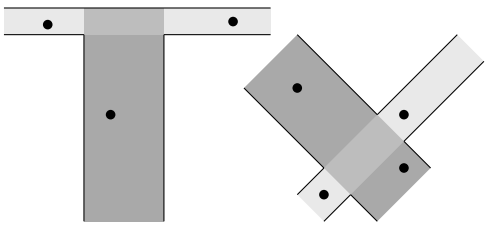

Figure 5: Invalid V-shapes that looks like a T or an X.

Algorithm 2. Part II: Finding S_2 of an inner-outer V-shape.

$i = L(e, P)$
For each edge $f \in H(k - i, P' - S_1)$: *
  $j = k - i - L(f, P' - S_1)$. * //# of outliers still needed
  Let $\ell =$ find-line(f, j, P').*
  Let $S_2 =$ the strip formed by f and ℓ.
  Record the thinnest S_2 found so far.

Lemma 2 *The above algorithms are correct.*

Proof. There are three ways these algorithms can fail. It can fail to find a valid V-shape, the V-shape it finds can have the wrong number of outliers, or it can overlook the thinnest V-shape with k outliers. How do we know that all the V-shapes we just enumerated with the above algorithms are valid? Two arbitrary strips may form a shape that looks like an X or a T instead of a V (see Figure 5). More formally, we want to avoid S_1 having points on *both* sides of S_2. The points covered by S_1 might indeed be split by S_2, but this can only happen when the points that were split off were among the k outliers. This is because only the outer ray of S_2 can split off points from S_1, and it only splits off points near the convex hull: the outliers.

The algorithms never create more than k outliers, because they keep track of how many are needed and at each step never create more than that. Do they ever create less than k? This can only happen if the algorithm counts some outlier more than once. The algorithms choose outliers three times: first when they choose e, then when choosing f, and finally when choosing ℓ. The outliers caused by e (that is, the i points to its left) are never double-counted, because they are invisible to the rest of the algorithm, which works with P' instead of P. The outliers created by f and those created by ℓ are on opposite sides of f, so they can not be counted twice either.

Lastly, can the thinest both-outer or inner-outer V-shape be overlooked? For both-outer and inner-outer V-shape, there is at least one outer ray defined by two points, and we consider all edges e that could possibly define it. For a fixed choice of e and p, we look at all

feasible choices of f. For a fixed choice of e, p and f, we have no freedom in choosing ℓ, so no wrong choice is possible. The only place where we do not look at all possibilities is in choosing p, where we do binary search. This is perfectly safe, because if, say, S_1 is thinner than S_2, moving p closer to e forces S_2 to cover more points, which can not make it any thinner. □

Lemma 3 *The running time of the above algorithms is* $O(n^2(k+1)^2 \log^2 n)$.

Proof. The algorithms are structured as a triply nested loop, so it suffices to count the number of iterations of each loop. It is well known that the set of $(\leq k)$-edges has size $O((k+1)n)$ [GP84, AG86], so the loops for e and f both iterate at most that many times. The binary search for p iterates $O(\log n)$ times. We can enumerate the j-edges, for all $j \leq k$, in sorted order along the j-level, in $O((k+1)n \log n)$ time using the algorithm in [EW86, pages 272–278][2]. By Lemma 4 we can implement find-line to run in $O(\log n)$ time. The claimed running time follows. □

Lemma 4 *After* $O((k+1)n \log n)$ *preprocessing, we can find the* $(i+1)$*st furthest point from a directed line* e *among points to the right of* e*, where* $0 \leq i \leq k$*, in* $O(\log n)$ *time.*

Proof. Finding the desired point is equivalent to finding the line ℓ parallel to e which goes through the point in P such that there are i points in P right of ℓ. (This is the line that find-line returns.) To do this, we go to the dual and let $\mathcal{A}$ be the line arrangement induced by P^*, where ℓ dualizes to a point ℓ^*. The requirement in the primal that there be k points right of ℓ means that ℓ^* must lie on an edge in the k-level or the $(|P|-k-1)$-level of $\mathcal{A}$, and the fact that ℓ must be right of e eliminates one of these two possibilities. Again using [EW86], we can compute the i-levels, and the $(|P|-1-i)$-levels, for all $i \leq k$, in sorted order by x-coordinate, in time $O((k+1)n \log n)$. Since we know the x-coordinate of ℓ^* (it is given by the slope of e in the primal), we can do binary search on the i-level to identify the two vertices that ℓ^* lies between. These two vertices lie on a line of P^*, which corresponds to a point of P in the primal. This is the desired point. □

2.3 Both-Inner

The following algorithm finds the thinnest both-inner k-outlier V-shape with high probability.

Lemma 5 *Algorithm 3 finds the thinnest both-inner* k*-outlier V-shape with probability at least* $1 - 1/n^c$ *for any* $c > 0$*, in* $O(cn^2(k+1)^4 \log n(\log n \log\log n + k))$ *expected time and* $O(n^2)$ *space.*

[2]The algorithm of [EW86] depends on a data structure for dynamic convex hull. At the time, the best available such structure was that of [OvL81, pages 169–181]. Using the one described in [J02] instead gives the claimed running time.

Algorithm 3. Finds a min-width both-inner k-outlier V-shape for P with high probability.

Input: integer k, point set P, real number $c > 0$
Let $n = |P|$, and let $K = k + 1$
Repeat $K^6 ce \ln n$ times:
  Initialize R to the empty set
  For each point in P, add it to R with probability $1/K$
  W = Find-empty-V-shapes(R) //[AD13, pp 303–304]
  Remove V-shapes with more than k outliers from W
Return the thinnest V-shape seen.

Proof. By [AD13], the above algorithm always produces the right answer if $k = 0$, albeit with needless redundant sampling of the entire point set, so we restrict our attention to the case where $k > 0$. Denote the thinnest both-inner k-outlier V-shape by V. Clearly, V is defined by (at most) six points of P. Consider a subset R of P, which contains the six points defining V but does not contain the k outliers. V is a valid both-inner 0-outlier V-shape for R, though perhaps not the thinnest one. The algorithm simply samples P over and over, in the hopes of eventually picking such a subset R. For each sample R, it enumerates all both-inner 0-outlier V-shapes using the algorithm from [AD13], and checks whether they end up having at most k outliers in P. Note that if all the V-shapes we consider end up resulting in more than k outliers, our algorithm fails to find *any* valid V-shape. However, we show that this is very unlikely: the probability that the algorithm fails to find the *optimal* V-shape is less than $1/n^c$, where c is the given positive constant.

Each point in P is independently chosen to be part of R with probability $1/K$. Thus, R has expected size n/K. Now, what is the probability that the optimal thinnest both-inner k-outlier V-shape is one of the valid both-inner 0-outlier V-shapes for R? The probability of having the required six defining points is $1/K^6$, and the probability of avoiding the k outliers is $(1-1/K)^k = (1-1/(k+1))^k > 1/e$, since $(1-1/(k+1))^k$ converges to $1/e$ from above. So, the probability of our random sample containing the six points we need and not containing the k points we should avoid is at least $p = 1/eK^6$. If we call this the probability of success, then the probability of failure is at most $1-p$. If instead of taking just one such random sample, we take $m = K^6$ samples, the probability of them all failing is at most $(1-p)^m$. Now using the fact that, for all x, $1 - x < e^{-x}$, we conclude that the probability q of all m samples failing to contain the optimum both-inner V-shape is $(1-p)^m < e^{-pm}$. Since $pm = (1/eK^6)(K^6) = 1/e$, we have $q < e^{-pm} = e^{-1/e}$. If we increase the number of samples from m to $mce \ln n$, then the probability of failure q reduces to at

most

$$e^{-pmce \ln n} = e^{-(ce \ln n)/e} = (e^{\ln n})^{-c} = 1/n^c,$$

which concludes the high-level description of the algorithm, and the proof that its probability of failure is at most $1/n^c$.

The crucial operation in the above algorithm is to check that the V-shapes returned by the algorithm from [AD13] do not have too many outliers. This can be done using range searching with wedges, which is a special case of simplex range searching, for which there are a variety of data structures with various space/time trade-offs. (A wedge is simply the convex region bounded by two rays with a common vertex.) We use a data structure that takes $O(n^2)$ space and gives $O(\log n \log\log n + k)$ query time [CY84, pages 41–45]. The time taken by the algorithm from [AD13] to enumerate all both-inner 0-outlier V-shapes of a point set with $O(n/k)$ points is $O((n^2/k^2)\log n)$. The subset may have as many as $O(n^2/k^2)$ V-shapes, each of which take $O(\log n \log\log n + k)$ time to check to make sure the number of outliers is not too high, for a total time of $O((n^2/k^2)(\log n \log\log n + k))$ per random sample.

We have glossed over a statistical subtlety here. If the expected value of a random variable X is $E[X]$, then in general $E[X^2]$ may not be $O(E[X]^2)$, or indeed, it might not even be finite. In this case, how do we know, just because the expected size of R is $O(n/k)$, that the expected number of V-shapes is $O(n^2/k^2)$? What is true for all random variables X from distributions with finite mean and variance is that $E[X^2] = E[X]^2 + \text{Variance}[X]$. The size of R follows the binomial distribution with mean n/K and variance $n(1/K)(1-1/K)$, so we have

$$E(|R|^2) = n^2/K^2 + n(1/K)(1-1/K) < n^2/K^2 + n/K,$$

which is $O(n^2/k^2)$.

Since we are taking $O(ck^6 \log n)$ random samples, finding the best both-inner k-outlier V-shape takes time $O(cn^2(k+1)^4 \log n(\log n \log\log n + k))$. □

Remark. We have calculated how many samples we need in order to find a particular both-inner k-outlier V-shape with high probability (specifically, the thinnest one). A natural question to ask is how many samples we would need to find *all* both-inner k-outlier V-shapes. If the probability of failing to find an arbitrary both-inner V-shape is q, then the probability of there being at least one both-inner V-shape we fail to find is at most q times the number of both-inner V-shapes present in the point set. Clearly, regardless of the value of k, this number is at most n^6, and we already computed $q < 1/n^c$. By choosing $c > 7$, we have $q < 1/n^7$, and thus our probability of failure is less than $1/n$.

Acknowledgments

We would like to thank Sariel Har-Peled for the randomized algorithm, and Muriel Dulieu for participating in discussions of a simpler version of the problem.

References

[AD13] B. Aronov and M. Dulieu, How to cover a point set with a V-shape of minimum width, *Comput. Geom.: Theory Appl.*, 46(3) (2013) 298–309.

[AG86] N. Alon and E. Györi, The number of small semispaces of a finite set of points in the plane, *J. Combinatorial Theory, Ser. A*, 41(1) (1986) 154–157.

[AS98] P.K. Agarwal and M. Sharir, Efficient algorithms for geometric optimization, *ACM Computing Surveys*, 30(4) (1998) 412–458.

[CY84] R. Cole and C.K. Yap, Geometric retrieval problems, *Information and Control*, 63(1–2) (1984) 39–57.

[EW86] H. Edelsbrunner and E. Welzl, Constructing belts in two-dimensional arrangements with applications, *SIAM J. Computing*, 15(1) (1986) 271–284.

[GP84] J.E. Goodman and R. Pollack, On the number of k-subsets of a set of n points in the plane, *J. Combinatorial Theory, Ser. A*, 36(1) (1984) 101–104.

[J02] R. Jacob, *Dynamic Planar Convex Hull*, PhD thesis, May 2002, University of Aarhus, Denmark, http://brics.dk/DS/02/3.

[OvL81] M.H. Overmars and J. van Leeuwen, Maintenance of configurations in the plane, *J. Computer System Sciences*, 23(2) (1981) 166–204.