Efficient computation of location depth contours by methods of computational geometry

Kim Miller* Suneeta Ramaswami† Peter Rousseeuw‡ J. Antoni Sellarès§
Diane Souvaine* Ileana Streinu¶ Anja Struyf‖

Abstract

The concept of location depth was introduced as a way to extend the univariate notion of ranking to a bivariate configuration of data points. It has been used successfully for robust estimation, hypothesis testing, and graphical display. The depth contours form a collection of nested polygons, and the center of the deepest contour is called the Tukey median. The only available implemented algorithms for the depth contours and the Tukey median are slow, which limits their usefulness. In this paper we describe an optimal algorithm which computes all bivariate depth contours in $O(n^2)$ time and space, using topological sweep of the dual arrangement of lines. Once these contours are known, the location depth of any point can be computed in $O(\log^2 n)$ time with no additional preprocessing or in $O(\log n)$ time after $O(n^2)$ preprocessing. We provide fast implementations of these algorithms to allow their use in everyday statistical practice.

Keywords: Bagplot, Bivariate Median, Graphical Display, Robust Estimation, Tukey Depth

1 Introduction

The location depth of a given point θ relative to a bivariate data set first occurred (without being given a name) as a test statistic of Hodges (1955) for the hypothesis that θ is the center

*Department of Electrical Engineering and Computer Science, Tufts University, Medford, MA 02155. `<kmillr,dls>@eecs.tufts.edu`. Partially supported by NSF grant EIA-9996237 and by *Forward in SEM*.

†Department of Computer Science, Rutgers University, Camden, NJ 08102. `rsuneeta@crab.rutgers.edu`

‡Department of Mathematics and Computer Science, U.I.A., Universiteitsplein 1, B-2610 Antwerp, Belgium. `Peter.Rousseeuw@ua.ac.be`

§Institut d'Informàtica i Aplicacions, Universitat de Girona, Spain. `sellares@ima.udg.es`. Partially supported by grants TIC2001-2392-C03-01 of the Spanish Government and DURSI 2001SGR-00296 of the Generalitat de Catalunya.

¶Department of Computer Science, Smith College, Northampton, MA 01063. `streinu@cs.smith.edu`. Partially supported by NSF grant CCR-9731804

‖Contact author. Department of Mathematics and Computer Science, U.I.A., Universiteitsplein 1, B-2610 Antwerp, Belgium. `Anja.Struyf@ua.ac.be`. Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium).

of the probability distribution from which the data was drawn. More recently, Rousseeuw and Struyf (2000) have put this test in a broader context by proving that location depth actually characterizes angular symmetry. Liu and Singh (1997) constructed other statistical tests based on location depth.

Definition. Let $Z = \{z_1, \ldots, z_n\}$ be a finite set of data points in $\mathbb{R}^p$ and let θ be an arbitrary point, not necessarily in Z. The *location depth* of θ relative to Z is the smallest number of points of Z lying in any closed halfspace determined by a line through θ.

The location depth thus varies between 0 (when θ lies outside the convex hull of Z) and n (when all points of Z coincide with θ). Figure 1 shows a point θ with location depth 1. The more θ is centrally located, the higher its depth. For sets Z in general position, the depth can be at most $\lfloor n/2 \rfloor$ (this occurs when Z is symmetric about θ). Throughout this paper, we will concentrate on bivariate data sets ($p = 2$).

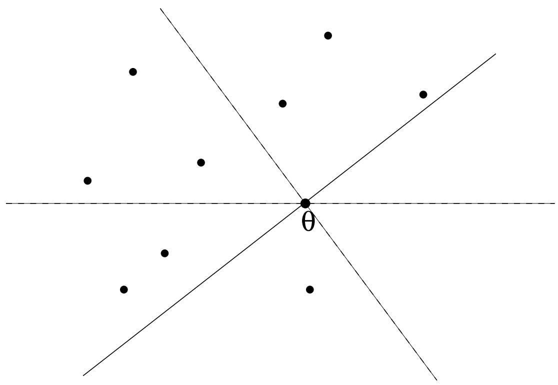

Figure 1: The point θ (which is not a data point) has location depth 1.

Definition. For a fixed positive integer k, the set of points in the plane with location depth $\geq k$ is a convex polygonal region, called the k-th *depth contour* (referred to as the k-hull in the computational geometry literature).

Tukey (1975) proposed to use depth contours in a graphical display of the data. The k-th contour is the intersection of all halfplanes containing $n - k + 1$ data points (see Figure 2). Note that the vertices of depth contours are either points from the original set or new points from the intersection of lines through two data points. The maximum depth (over all θ) of Z is denoted by k^*, and from Helly's theorem (see Wenger 1997) it follows that always $k^* \geq \lfloor n/3 \rfloor$ (this fact is also known as the existence of at least one centerpoint of Z).

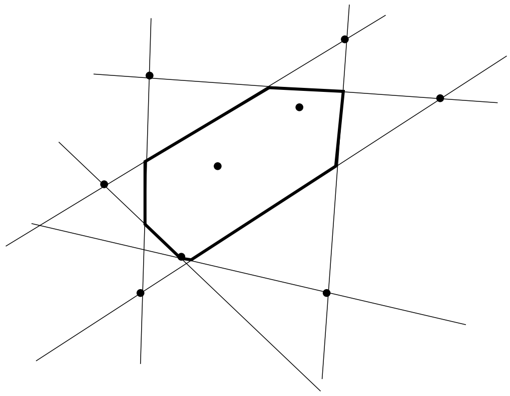

Figure 2: The boundary of depth contour 2 is drawn in bold.

Definition. The Tukey Median, T^*, is the center of gravity of the deepest contour.

Donoho and Gasko (1992) showed that T^* is a location estimator with several desirable properties. They define the finite-sample breakdown value of an estimator as the smallest fraction of contamination that needs to be added to a data set to make the estimator arbitrarily large. Thus,

$$\epsilon^*(T^*; Z_n) = \min\{\frac{m}{n+m}; \sup_{Z_{n+m}} \|T^*(Z_{n+m}) - T^*(Z_n)\| = \infty\},$$

where Z_{n+m} is a data set formed by adding m observations to Z_n. The Tukey median has a good breakdown value (making it more robust than the median based on Liu's (1990) simplicial depth) with $\epsilon^*(T^*; Z) \geq 1/(p+1)$ for any sample in general position. When the data set is drawn from an angularly symmetric distribution, $\epsilon^*(T^*; Z)$ tends to $1/3$ in any dimension. Hence, when at least $2/3$ of the points come from such a distribution, then the median remains in a bounded region, regardless of the position of the other points. Moreover, the location depth is invariant under affine transformations, making T^* affine equivariant (unlike the spatial median, which is based on distances). The asymptotics of the Tukey median have been studied by Bai and He (1999).

The bagplot proposed by Rousseeuw, Ruts, and Tukey (1999) is a two-dimensional generalization of Tukey's univariate boxplot (Tukey 1977). It is based on location depth and provides a visual representation of the location, spread, correlation, skewness, and tails of the data. Figure 3 shows the bagplot of the spleen weight versus the liver weight of 73 hamsters (Cleveland 1993). The Tukey median T^* is depicted by a cross, and the dark grey area around the median is the bag which contains 50% of the data. The bag is constructed as an interpolation between two subsequent depth contours and gives us an idea of the

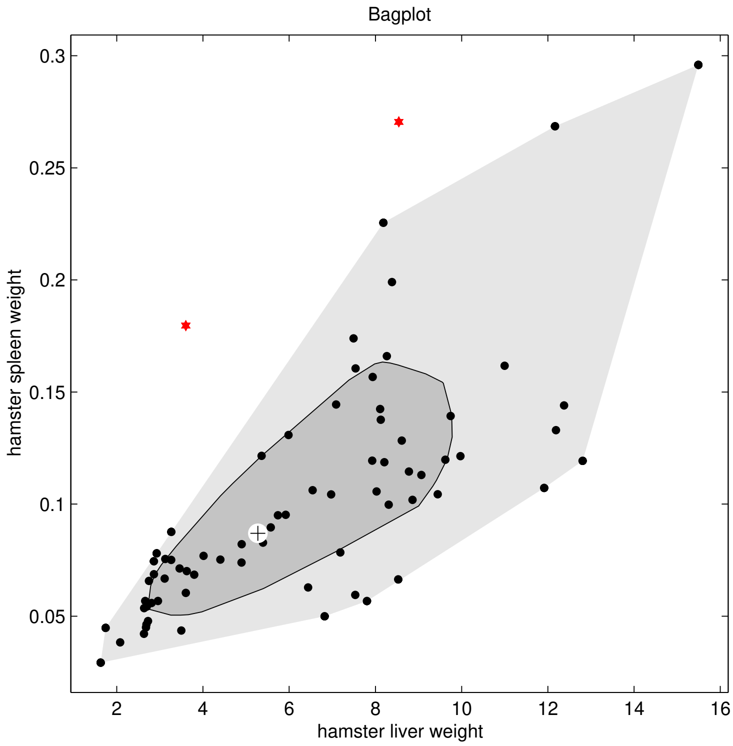

Figure 3: Bagplot of the spleen weight versus the liver weight of 73 hamsters.

shape of the data cloud. Here we see that there is a positive correlation between the two variables. A data point is considered an outlier if it lies outside of the fence (not displayed in the bagplot), which is obtained by inflating the bag by a factor 3 relative to T^*. The light grey area is the convex hull of the bag and the non-outliers. The computation of the fence and the light grey area are immediate once the Tukey median and the bag are known. Here we have two outlying observations, depicted by stars. These two hamsters have a large spleen relative to the weight of their liver.

1.1 Previous implementations and our results

Theoretical complexity results on depth contours, or k-hulls, have been known for some time (Cole, Sharir and Yap 1987). The best theoretical result for computing the Tukey median is an $O(n \log^5 n)$ algorithm by Matoušek (1991), but its complex structure makes it an unlikely candidate for actual implementation. Fast implementations of depth contours are critical for several applications. An overview of statistical methods based on the location depth is given by Liu, Parelius and Singh (1999). Not many implemented algorithms were available.

The program ISODEPTH (Ruts and Rousseeuw 1996) computes the k-th depth contour in $O(n^2 \log n)$ time, and hence needs $O(n^3 \log n)$ time to compute all depth contours. The programs HALFMED (Rousseeuw and Ruts 1998) and BAGPLOT (Rousseeuw, Ruts and Tukey 1999) give $O(n^2 \log^2 n)$ implementations to compute the Tukey median and the bag. These programs become impractically slow for large data sets.

We present an $O(n^2)$ time algorithm for computing *all* the depth contours, as well as the location depth of all the data points. Consequently, we compute the Tukey median and the bag in the same time bound. We achieve our results by using duality and topological sweep of an arrangement of lines. All our algorithms are numerically stable. Our code allows collinearities in the data set, but currently assumes that no two data points have the same x-coordinate (which can always be achieved by rotating the data first). Preliminary code which does not have this assumption is also available. We also give empirical results on the dramatic speed-up (by a factor of more than 300 for data sets of size 700 and higher) provided by our algorithm over ISODEPTH and HALFMED.

In the paper, we will use standard notations to describe the time and space complexities of our algorithms. For functions $f(n)$ and $g(n)$, $f(n) = O(g(n)$ if there exists a $c > 0$ and an $N_0 > 0$ such that for all $n > N_0$, $f(n) \leq cg(n)$. Conversely, $f(n) = \Omega(g(n)$ if there exists a $c > 0$ and an $N_0 > 0$ such that for all $n > N_0$, $f(n) \geq cg(n)$. Moreover, $f(n) = \Theta(g(n))$ denotes that both $f(n) = O(g(n)$ and $f(n) = \Omega(g(n))$. Finally, $f(n) = o(g(n))$ if $\lim_{n\to\infty} f(n)/g(n) = 0$.

2 The Algorithms

2.1 Preliminaries

The planar partition induced by a set L of non-vertical lines in the plane is referred to as the *arrangement* induced by L. An arrangement consists of vertices, edges, and cells. Some of the edges and cells are unbounded. We give below some standard definitions (see e.g. de Berg et al. 1997, Preparata and Shamos 1985) that are relevant to the description of our algorithm.

Definition. Given an arrangement of lines L in the plane, the *lower envelope* (*upper envelope*) of L is the boundary of the "lowermost" ("uppermost") unbounded cell. This is the cell with the property that if a downward (upward) vertical ray is drawn from any point in the cell, it will not intersect the arrangement anywhere. See Figure 4(a).

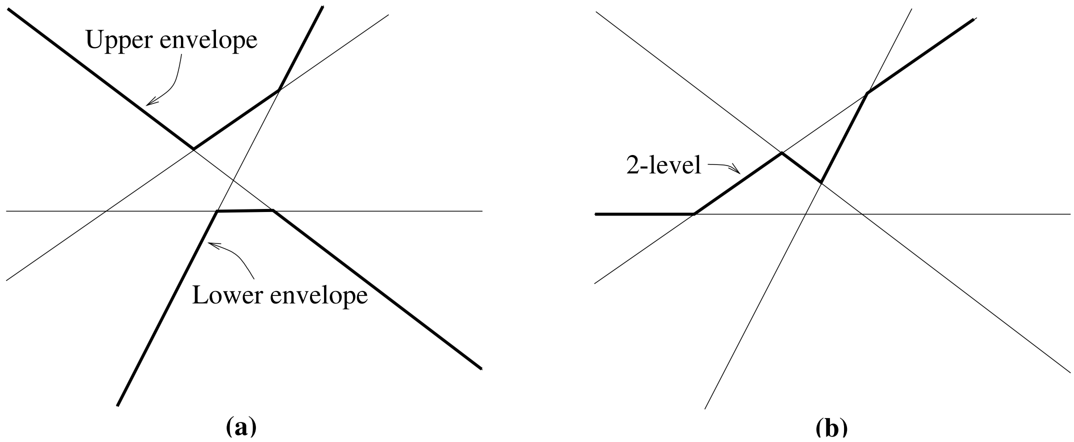

Figure 4: (a) Upper and lower envelopes of an arrangement of lines; (b) the 2-level.

Definition. Given an arrangement of n lines in the plane, the *k-level* of the arrangement is the set of points of the arrangement that have at most $k-1$ lines strictly above it, and at most $n-k$ lines strictly below it. See Figure 4(b). Note that the 1-level is the upper envelope and the n-level is the lower envelope.

Definition. The *upper hull* (*lower hull*) of a planar set of points Z is defined to be the set of edges of the convex hull of Z that have Z *below* (*above*) the supporting line. See Figure 5.

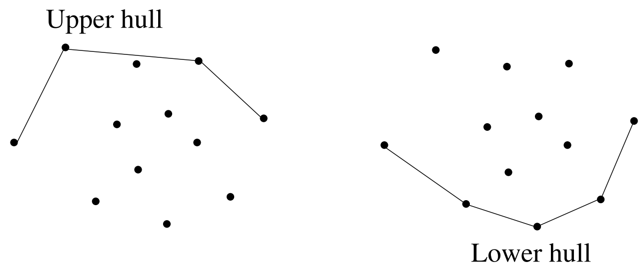

Figure 5: The upper and lower hulls of a set of points.

Definition. We define a *k-divider* of the point set Z as a line having exactly $k-1$ points of Z on one side and at most $n-k$ points of Z on the other side (so the line must contain at least one point of Z). A *special k-divider* is a k-divider that contains at least two points of Z. A *special k-halfplane* is a closed halfplane bounded by a special k-divider and containing $n-k+1$ points of Z.

The k-th depth contour is the convex polygon obtained by intersecting all the special k-halfplanes of Z (Cole, Sharir and Yap 1987). A special k-halfplane is an upper (resp. lower) halfplane if it is bounded from below (resp. above). Therefore the k-th depth contour is the intersection of the lower envelope of the lines that bound the lower special k-halfplanes, with the upper envelope of the lines that bound the upper special k-halfplanes.

2.2 Determining the Depth Contours

Constructing the depth contours involves three major steps. The first step describes a conceptual tool and provides the framework on which steps 2 and 3 are based. A brief description of each step is given below. Explanatory and algorithmic details appear later in the section.

1. First we use a duality to map every data point to a line. In particular, we map the point $z_i = (x_i, y_i)$ to the line $b = (-x_i)a + y_i$ in the (a, b)-space. Hence the set of data points in the primal defines an arrangement of lines in the dual. An intersection point of two lines in the dual (i.e., a vertex of the arrangement) corresponds to a line between the respective points in the primal.

2. Then we use topological sweep to find all vertices of the arrangement of lines efficiently. As we find each vertex in the dual, we can determine the level of that vertex. This gives us the depth contour to which the corresponding line (and its associated halfplane) potentially contributes in the primal.

3. Finally, each convex depth contour is given by the intersection of the halfplanes determined in the previous step. This is equivalent to finding the upper and lower convex hulls of the corresponding vertices in the dual. The intersection of the halfplanes determined by the hull vertices gives the final depth contour.

A high-level description of the algorithm to construct the depth contours for a data set Z of n points may be given as follows.

```
Algorithm DepthContours(Z, n)
{
  L ⟵ dual(Z);

  // TopoSweep returns, for each k, the sorted list of arrangement
  // vertices that potentially contribute to the k-th depth contour
  Levels ⟵ TopoSweep(L, n);

  // FindHulls finds the upper and lower hulls of the arrangement
  // vertices identified above as potential contributors to a given
  // contour
  for (i = 1 to n/2)
      contours[i] ⟵ FindHulls(Levels[i]);

  return contours;
}
```

This method was first suggested by Cole, Sharir and Yap (1987), but without using topological sweep. Also for the least median of squares regression estimator (Rousseeuw 1984), efficient algorithms have been constructed by means of duality and topological sweep (Edelsbrunner and Souvaine 1990, Souvaine and Steele 1987). In the remainder of this section, we provide a detailed description of each step of the algorithm. The discussion assumes that no two data points have the same x-coordinate.

The Arrangement. We use the standard dual mapping that maps each point $z = (x, y)$ to the non-vertical line $\mathcal{D}(z) : b = (-x)a + y$, and each non-vertical line $l : y = ax + b$ to the point $\mathcal{D}(l) = (a, b)$ (see e.g. Dobkin and Souvaine 1987, Edelsbrunner 1987). This duality has a number of interesting properties. It preserves the *above/below* relationship: point z lies above/on/below line l in the primal plane if and only if line $\mathcal{D}(z)$ lies above/on/below point $\mathcal{D}(l)$. It is *intersection preserving*: lines l_1 and l_2 intersect at point z if and only if line $\mathcal{D}(z)$ passes through points $\mathcal{D}(l_1)$ and $\mathcal{D}(l_2)$, and line l passes through points z_1 and z_2 if and only if lines $\mathcal{D}(z_1)$ and $\mathcal{D}(z_2)$ intersect at point $\mathcal{D}(l)$. The line between two points in the primal dualizes to the intersection point of the corresponding lines in the dual.

Given a set L of non-vertical lines in the dual, we use $\mathcal{D}(L)$ to denote the set of its primal points. Since the slope of a line in the dual is the negative x-coordinate of the primal point, the left-to-right list of edges of the lower envelope of L corresponds exactly to the right-to-left list of points of the lower hull of the points of $\mathcal{D}(L)$. Similarly, the left-to-right list of edges of the upper envelope of L corresponds to the left-to-right list of points of the upper hull of $\mathcal{D}(L)$. So, the lower (resp. upper) envelope of a set of lines in the dual can be found by computing the lower (resp. upper) hull of the primal set of points. See Figure 6.

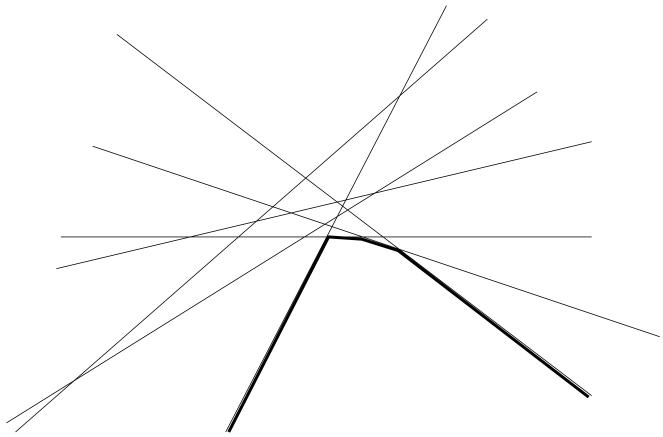

Figure 6: The lower envelope of lines in the dual is the lower hull of the primal set of points.

Consider the arrangement induced by the set of dual (non-vertical) lines of Z. Since no two points of Z have the same x-coordinate, no two lines of the arrangement are parallel. Let l be a line through two points in the primal corresponding to the intersection point i in the dual. The vertical line v through i in the dual intersects every line of the arrangement, some above i and some below i. The number of lines intersected above (resp. below) i exactly equals the number of points lying above (resp. below) l in the primal. Let m be the number of crossings above i and let r be the number of crossings below i. Define $k = \min\{m, r\} + 1$. It follows that l is a special k-divider and its associated special k-halfplane potentially contributes to the k-th depth contour.

During the topological sweep of the arrangement (the next step of the algorithm), in order to make the future computation of the depth contour easier, when a vertex of the arrangement is identified, it is classified as *upper* or *lower*. Using the notation from above, if m was the minimum, more points lie below the corresponding line in the primal and we say that the vertex belongs to the *k-th upper set*. Conversely, if r was the minimum the vertex is said to belong to the *k-th lower set*. See Figure 7. After the sweep, since all vertices have been examined, for each k we have a list of arrangement vertices, sorted by x-coordinate, that form the upper and lower sets. These vertices correspond to lines in the primal that potentially contribute to the k-th depth contour. Observe that the points of the k-th upper set are the vertices of the k-level of the arrangement of dual lines, whereas the points of the k-th lower set are the vertices of the $(n - k + 1)$-level of the arrangement.

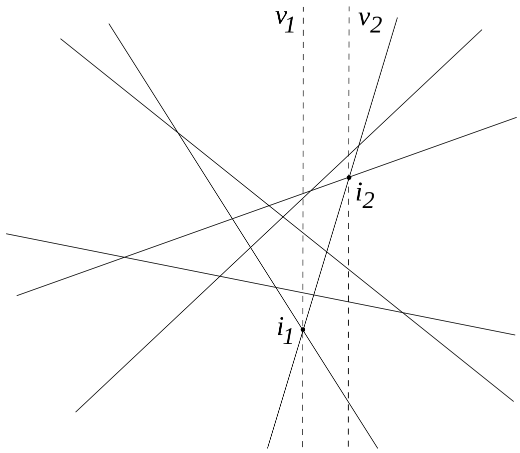

Figure 7: Intersection point i_1 has 4 lines above and 0 line below; hence it belongs to the lower set of the first contour. Intersection point i_2 has 1 line above and 3 lines below; hence it belongs to the upper set of the second contour.

The lower envelope of the lines that bound the lower special k-halfplanes in the primal corresponds exactly to the lower hull of the k-th upper set in the dual. Similarly, the upper envelope of the lines that bound the upper special k-halfplanes in the primal corresponds exactly to the upper hull of the k-th lower set in the dual. In particular, an edge of the k-th lower (or upper) hull that is not included in a line of the arrangement provides a vertex of the k-th depth contour that is not a point from Z. Once the k-th upper and lower hulls are computed in the dual, the k-th depth contour is found by intersecting the corresponding set of upper and lower special k-halfplanes in the primal. If the intersection is empty, the depth contour does not exist.

Our implementation uses the above approach of computing the upper and lower hulls in the dual, followed by halfspace intersection in the primal. However, it is also possible to determine the k-th depth contour entirely in the dual, as follows: The convex hull of a set of points in the primal corresponds to the upper envelope and the lower envelope of the dual arrangement. Note that the leftmost edge of the upper envelope and the rightmost edge of the lower envelope are colinear. Similarly, the rightmost edge of the upper envelope and the leftmost edge of the lower envelope are colinear. Consider the inner common tangents of the two envelopes. These consist of two finite segments of those same two common lines. Therefore we can think of the convex hull of a set of points in the primal as corresponding to the finite edges of the upper envelope, the finite edges of the lower envelope, and the finite inner common tangents that join them. Consider now the lower hull of the k-th upper set and the upper hull of the k-th lower set. We need to calculate the inner common tangents between these two convex chains. Then the finite portions of these tangents and portions of the convex chains that join the endpoints correspond to the boundary of the depth contour in the primal. If these inner common tangents do not exist, then the lower hull of the upper set and the upper hull of the lower set intersect, and no depth contour at this level exists. These inner common tangents correspond to the points of intersection between the chains in the primal that bound the intersections of the lower special k-halfplanes and the upper special k-halfplanes respectively. If these intersections do not exist, the boundary of intersection of the lower halfplanes is below the boundary of intersection of the upper halfplanes and thus there is no intersection region.

The main motivation for solving the depth contours problem in the dual is that the technique known as *topological sweep*, described in the next step, allows us to efficiently find all arrangement vertices and their levels. While topological sweep may be simulated in the primal, it is more easily described and implemented in the dual.

Topological Sweep. The typical line sweep method is to use a vertical sweepline to sweep through the arrangement vertices in increasing order of their first coordinate. This is done by maintaining a priority queue (heap) of potential next points for the vertical sweep. It can be shown that this requires only a linear amount of space, but at a cost of $O(\log n)$ per update when a new vertex is swept, resulting in a total time complexity of $O(n^2 \log n)$. In many problems, it is not necessary to obtain the arrangement vertices in sorted order. It is enough simply to visit each of them in the following partial order: the vertices of each level of the arrangement are visited in increasing order of their first coordinate. Such an approach is used in *topological sweep*, which is a variation of vertical line sweep that gives an improved runtime of $O(n^2)$ while retaining the $O(n)$ space bound. The topological sweep algorithm was introduced by Edelsbrunner and Guibas (1989). For the sake of completeness, we provide a brief description of their sweep technique here.

Topological sweep sweeps a curved line, rather than a straight line, through the arrangement. The curved line retains an important property of the vertical sweepline, which is that each line of the arrangement is intersected exactly once by the sweepline. The set of edges of the arrangement intersected by the sweepline is called a *cut*. The sweep algorithm starts at the leftmost cut (which consists of all unbounded edges of the arrangement with unbounded left vertices) and ends when it is at the rightmost cut. A cut is maintained as an array of indices of the lines in the order they are intersected. When a *consecutive* pair of edges of the cut intersect at a common vertex before intersecting any other line, that vertex becomes a potential next candidate for the topological sweep (note that every cut must always have such a pair of edges). If more than one such vertex exists, any one of them may be chosen next for the sweep step. See Figure 8(a). Observe that after the chosen vertex has been swept, the new cut simply swaps the order of the pair of lines that gave rise to the vertex. See Figure 8(b).

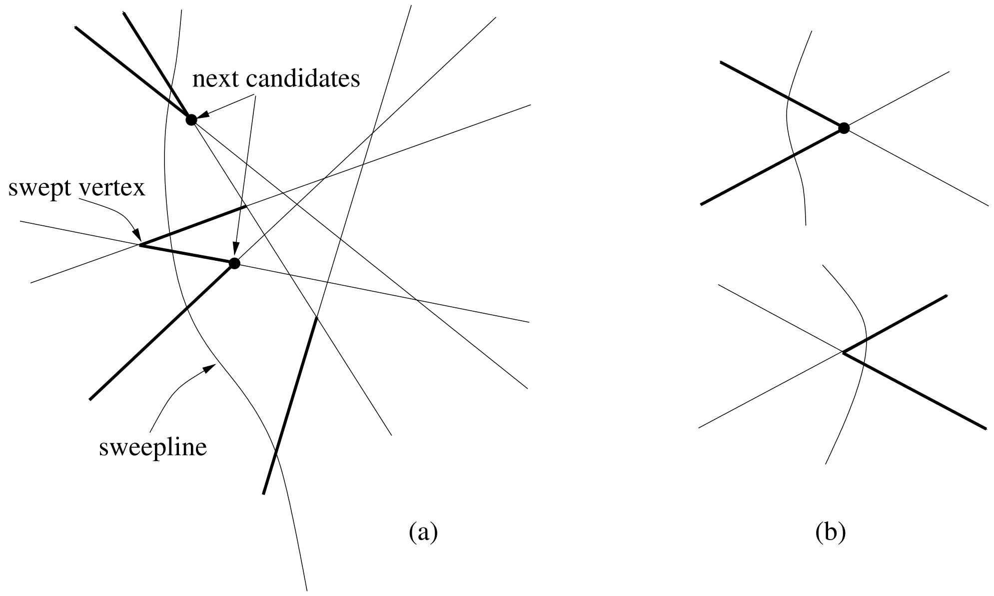

Figure 8: (a) Cut edges are shown as thick lines. Potential candidate vertices for the next sweep step are highlighted. (b) A single sweep step.

It would not be difficult to find the desired pair for each cut in $O(\log n)$ time. However, this would cause the total time complexity of the algorithm to be $O(n^2 \log n)$. Instead, the algorithm in (Edelsbrunner and Guibas 1989) achieves its $O(n^2)$ runtime by efficiently finding a consecutive pair of edges of a cut that intersect at a common vertex. In particular, an amortized cost of $O(1)$ per pair is achieved by maintaining data structures called the *upper horizon tree* and the *lower horizon tree* (efficiently implemented using arrays). The horizon trees are used to find all consecutive pair of edges of a cut that share a common vertex, and all such pairs are stored on a stack. The next vertex to be swept is given by the pair at the top of the stack. A single sweep step consists of popping the stack, updating the cut and the horizon trees, and pushing any new consecutive cut edges with shared endpoints onto the stack. Note that the stack size remains linear at all times since the size of the cut is linear. For the runtime analysis, the crucial facts are that (a) the initial horizon trees can be constructed in linear time, and (b) updating the horizon trees takes $O(1)$ amortized time (i.e., a single update may take more than $O(1)$ time, but the total update time for the entire sweep is $O(n^2)$).

During the topological sweep, we can easily determine the level of each arrangement vertex. When a vertex given by a pair of cut edges is swept, the level of that vertex is simply the position of the upper cut edge in the cut array. Observe that all the vertices at the k-level, for any k, are in fact swept in sorted order of their first coordinate. Our implementation of this step returns, for each of $k = 1 \dots n/2$, a sorted list of arrangement vertices at level k, and at level $n-k+1$ (as discussed in the previous step, these are the vertices that potentially contribute to the k-th depth contour). Thus, the upper and lower sets for each depth contour are determined during the topological sweep at no extra cost. If we are computing all depth contours, all $\Theta(n^2)$ intersection points of the arrangement are saved in order to compute the depth contours (step 3 of our algorithm), producing quadratic space complexity.

Intersection of Halfplanes. The lower and upper hulls of a set of points can be computed in linear time in the size of the set if the points are given in increasing order of their first coordinate. For example, this can be done by using an incremental algorithm, which can be found in standard textbooks in computational geometry (de Berg et al. 1997, Preparata and Shamos 1985). After the topological sweep, we have the sorted list of arrangement vertices of the k-th upper and lower sets. Note that while the size of the lower and upper hulls is always $O(n)$ (Matoušek 1991), the best known upper bound on the size of k-levels of the arrangement of dual lines is $O(nk^{1/3})$ (Dey 1998).

For each k we find the lower hull of the upper set and the upper hull of the lower set in time linear in the sizes of the k-level and $(n-k+1)$-level. The hulls represent the upper envelope and lower envelope of the corresponding halfplanes in the primal.

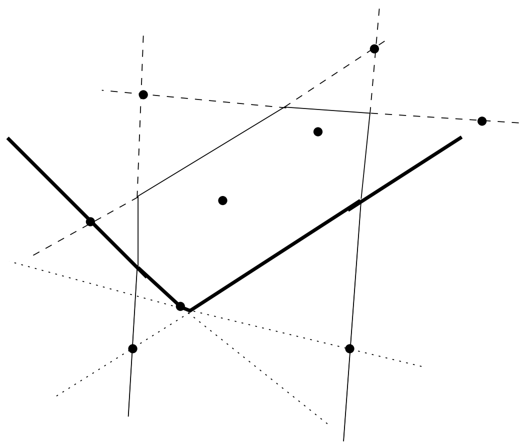

Figure 9: The dashed (dotted) lines are the halfplanes associated with the upper (lower) set of intersection points in the arrangement.

See Figure 9. The halfspaces bounded by the lower and upper envelopes are then intersected in the primal. This is done in time linear in the sum of the sizes of the two envelopes, which is $O(n)$, which gives us the k-th depth contour (if it exists). So, we obtain the k-th depth contour in time linear in the sizes of the k and $(n-k+1)$-levels. Therefore, the total runtime of this step is the sum of the sizes of all the levels of the arrangement of dual lines, which is $O(n^2)$. This step is summarized below.

Algorithm FindHulls(VertexList)
{
  $U \longleftarrow$ upper set of VertexList;
  $L \longleftarrow$ lower set of VertexList;

  UH $\longleftarrow$ lower hull of U;
  LH $\longleftarrow$ upper hull of L;

  // $\mathcal{U}$ and $\mathcal{L}$ represent the upper and lower
  // envelopes of $\mathcal{D}(U)$ and $\mathcal{D}(L)$ respectively
  $\mathcal{U} \longleftarrow$ UH in the primal;
  $\mathcal{L} \longleftarrow$ LH in the primal;

  return $(\mathcal{U} \cap \mathcal{L})$;
}

2.3 Location Depth of a Single Point

After Contour Calculations. After calculating all the depth contours, the depth of a given point may be computed efficiently in two different ways.

- A simple and easily implementable algorithm yields the depth of a point in $O(\log^2 n)$ time as follows. The initial contour calculations provide a set of nested convex polygons representing each depth contour. The depth contours subdivide the plane into depth regions; the region between contours k and $k+1$ has depth k. Therefore, the depth of the given point is simply the depth region in which it lies. Determining if a point lies inside or outside a convex n-gon takes $O(\log n)$ time. To determine the depth of a single point we can thus perform a binary search on the depth contours to compute the result in $O(\log^2 n)$ time.
- A more efficient algorithm computes the depth of a point in $O(\log n)$ time using any of the optimal algorithms for planar point location (e.g. Kirkpatrick 1983; Sarnak and Tarjan 1986; and Edelsbrunner, Guibas, and Stolfi 1986). It is easy to connect each interior nested convex polygon to the next larger polygon by adding a line segment from its vertex with minimum x-coordinate to the comparable vertex on the larger polygon and a second line segment connecting the vertices of maximum x-coordinate. All bounded regions are now monotone, meaning that the remaining preprocessing for any of the three cited algorithms can be completed in linear-time in the size of the subdivision. The collection of depth contours is a planar subdivision of size $O(n^2)$. Hence, after $O(n^2)$ preprocessing time, computing the depth of a point takes $O(\log n)$ time. A version using the Edelsbrunner *et al.* approach has been implemented by Mitchell (2002).

Without Contour Calculations. The depth contours need not be calculated first to find the depth of a single point. An alternative $O(n \log n)$ time algorithm is described in Rousseeuw and Ruts (1996). Recently it was shown (Langerman and Steiger 2000) that any algorithm for computing the location depth of a point needs $\Omega(n \log n)$ time, so the above algorithm has optimal time complexity. This algorithm has also been extended to three dimensions in $O(n^2 \log n)$ time (Rousseeuw and Struyf 1998).

2.4 The Bag

There are three main steps in the construction of the bag. Recall that the bag contains at most half of the original data points, and lies between the contour containing less than or equal to half and the contour containing more than half of the points.

1. The first step in the construction is to identify these two contours. Let D_k and D_{k-1} be two consecutive contours containing respectively n_k and n_{k-1} data points, such that $n_k \leq \lfloor n/2 \rfloor < n_{k-1}$. Since the contours have already been calculated, start at the deepest and count the number of original points lying on each contour until the count exceeds $\lfloor n/2 \rfloor$. The contour at which we stop counting is D_{k-1} and the previous one is D_k.

2. Next, we calculate the value of a parameter λ, which determines the relative distance from the bag to each of the two contours. This is given by $\lambda = (50 - J)/(L - J)$, where D_k contains $J\%$ of the original points and D_{k-1} contains $L\%$ of the original points.

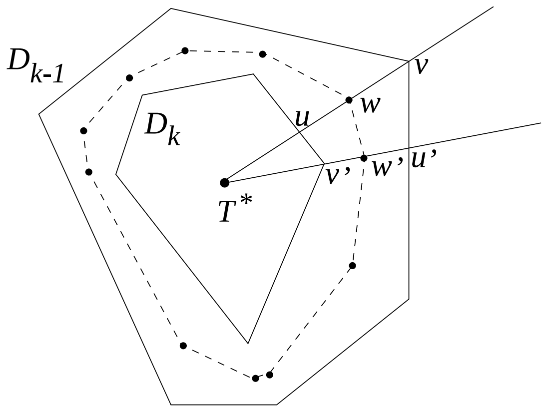

Figure 10: Determining the vertices of the bag.

3. Finally, the bag is constructed by interpolating between the two contours by using the λ value. For each vertex v on D_k or D_{k-1} let l be the line going through v and the Tukey median T^*. Let u be the intersection of l with the other contour (D_{k-1} or D_k). Each vertex w of the bag lies on l, interpolated between v and u, i.e., $w = \lambda v + (1-\lambda)u$. The value of λ weights the position of the bag towards D_k or D_{k-1}. See Figure 10.

3 The Implementations

The algorithm described in the previous section has theoretical running time faster than any existing implementation for depth contours. As our empirical results show, this is borne out in practice as well.

3.1 Complexity

The overall complexity of the implementation is $O(n^2)$ time and $O(n^2)$ space to compute all the depth contours for a planar set of n points. Topological sweep in itself uses linear space. Nonetheless, the space complexity is $\Theta(n^2)$ when all of the contours are found, because all intersection points of the arrangement need to be saved. Note that the space complexity would be linear if only the k-th depth contour is needed, because the convex hull of the k-set can be computed incrementally and, as pointed out earlier, the size of any depth contour is always $O(n)$.

The complexity of each step is as follows. Specifying the arrangement takes $O(n)$ time and space. Topological sweep takes $O(n^2)$ time and $O(n)$ space. Since we compute all the contours for the testing of the code, our runs use $O(n^2)$ space. Computing the upper and lower hull of the intersection points in the arrangement for all contours requires $O(n^2)$ time and space.

3.2 The code

The main focus of the implementation is the computation of all depth contours and the bag. Our program is implemented in `C++` using the LEDA libraries (Library of Efficient Data structures and Algorithms, `www.mpi-sb.mpg.de/LEDA`). The bivariate data points may be entered interactively or by reading from a file, and the contours can be output either in graphical form or as a file containing lists of vertices for each contour. See Figures 11 and 12 for examples of the graphical output produced by our program.

Once the contours are available, computing the bag requires only linear time. Our implementation referred to the `C` code for topological sweep given in (Rosenberger 1990), but we rewrote the code in order to make it compatible with the LEDA libraries. A linear time incremental algorithm for computing the upper and lower hulls of points given in sorted order was implemented as well. This step also identifies the location depth of the original data points, which is required for the bag computation. Our implementation allows collinearities in the data set, but currently assumes that no two points have the same x-coordinate. There is preliminary code that makes no assumptions about the input set. All primitive geometric computations rely on LEDA and inherit their numerical stability.

3.3 Significance

The testing phase consists of runtime comparisons for computing the Tukey median. To check the correctness of our code, we first ran it on data sets for which the Tukey median

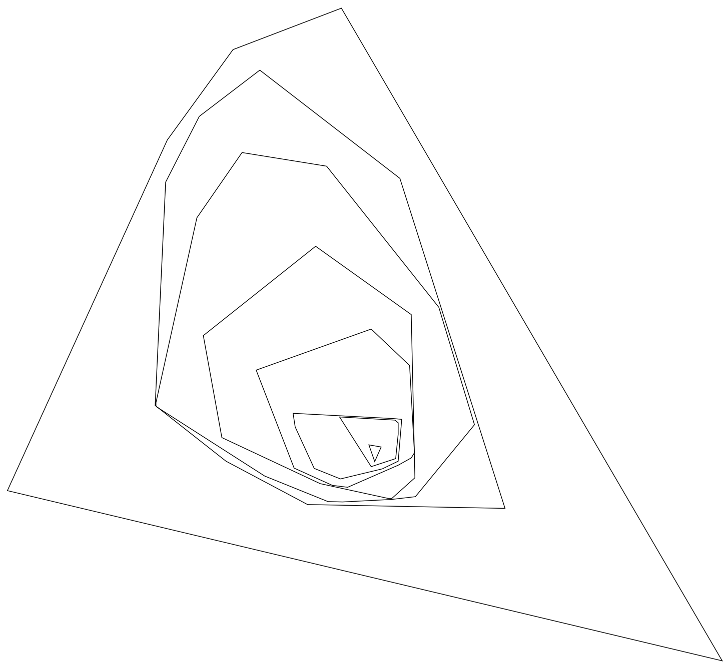

Figure 11: Depth contours produced by our program for a set of 20 points.

was already known. The previously best-known algorithm for computing the Tukey median is HALFMED (Rousseeuw and Ruts 1998), which uses an $O(n^2 \log n)$ time algorithm to compute a single contour. It then uses binary search to repeatedly apply this algorithm until the deepest contour is found, giving an $O(n^2 \log^2 n)$ algorithm for computing the Tukey median (the center of gravity of the deepest contour). Our algorithm finds the Tukey median by computing all the contours, as described in Section 2, in $O(n^2)$ time.

Our program was tested on a Sun Ultrasparc 167 MHz workstation, running SunOS, and was compiled using the GNU `C++` compiler. HALFMED is written in Fortran, and was compiled using the GNU Fortran compiler. The testing was done on randomly generated point sets of various sizes. Ten data sets of each size were generated, and the total runtime for each size was recorded using perl benchmark scripts. The average runtimes for HALFMED and our code are shown in Table 1.

As Table 1 shows, our algorithm provides a dramatic speed-up over HALFMED for data sets of size greater than 40. For smaller data sets, the overhead of computing the arrangement and using more sophisticated data structures makes the algorithm slower than HALFMED. However, the improvement in runtime for larger data sets is notable.

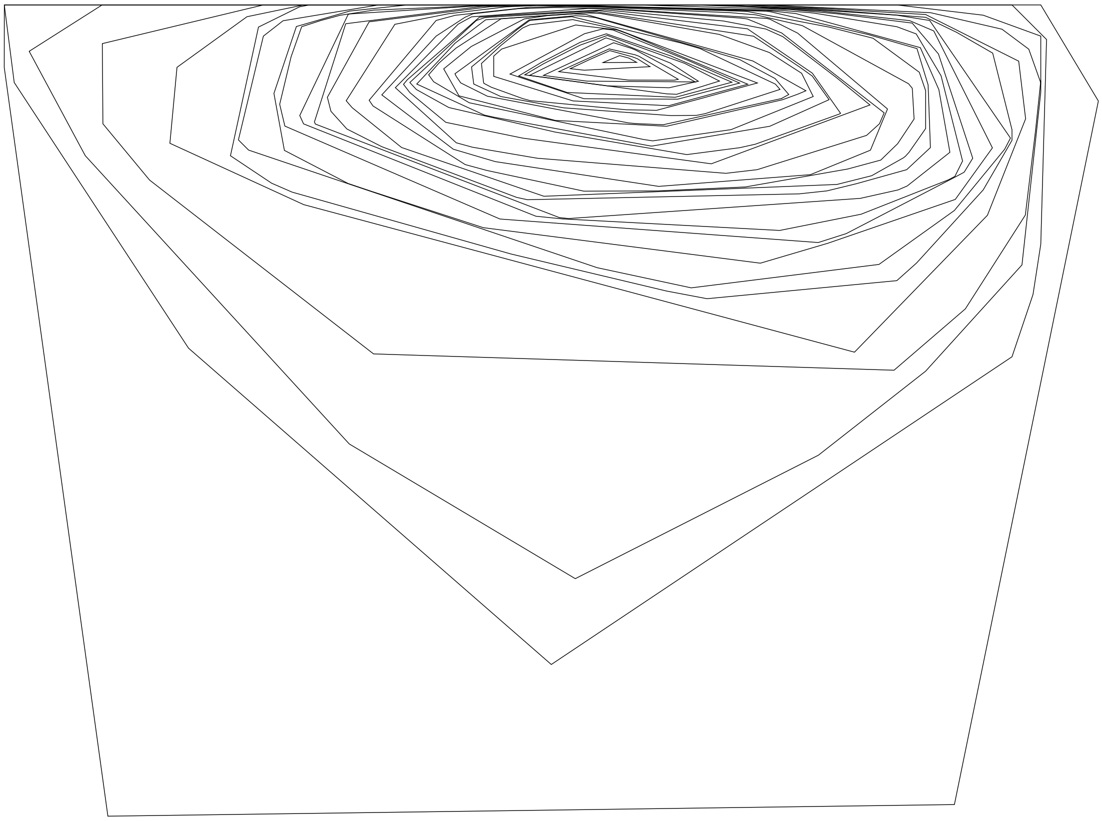

Figure 12: Depth contours produced by our program for a set of 79 points.

4 Concluding Remarks

We have constructed fast algorithms for computing all depth contours, the Tukey median and the bag and for performing subsequent depth queries. Our results provide a significant speed-up over existing algorithms. The code, which requires LEDA 3.8 or higher, is available at

`http://www.eecs.tufts.edu/r/geometry/locdepth.`

Our implementation currently performs depth queries in $O(\log^2 n)$ time and assumes that no two points in the input set have the same x-coordinate. This assumption can easily be satisified, for example by an appropriate linear transformation of the data set. Recently Rafalin, Souvaine and Streinu (2002) developed a method for removing this non-degeneracy assumption and preliminary code for computing the depth contours of an arbitrary point set is available at

`http://www.eecs.tufts.edu/r/geometry/half_space.`

n	Average CPU time over 10 runs (seconds)	
	HALFMED	Our algorithm
10	.03	2.1
20	.09	2.1
40	.61	2.1
80	5.6	2.2
100	11.8	2.3
160	37.2	2.8
200	84.4	3.1
320	490	4.5
500	760	6.0
640	1143	8.3
750	4954	14.2
1000	9002	22.1

Table 1: Average runtimes.

Mitchell (2002) has also implemented an algorithm for $O(\log n)$ depth query.

A natural open question concerns faster algorithms for computing only the Tukey median. Theoretically, faster algorithms exist (Matoušek 1991). However, a more implementable algorithm is required in practice. The problem of an efficient implementation for computing the Tukey median in $o(n^2)$ time is the outstanding open problem in two dimensions from a practical point of view. Analysis and implementation of a randomized version of Matoušek's algorithm is underway (Lal and Souvaine 2002).

The next goal is to compute all depth contours, the Tukey median, and the bag in three dimensions. Anagnostou, Guibas, and Polimenis (1990) presented an algorithm for topological sweep in three dimensions from a theoretical perspective that assumes general position. Building off of this algorithm but creating and incorporating a generalized version of the Rafalin, Souvaine, and Streinu (2002) strategy for handling degeneracy and incorporating our strategies for building two dimensional contours should lead to the desired three dimensional code. Work is underway.

References

Anagnostou E. G., Guibas L. J., and Polimenis V. G. 1990. Topological sweeping in three dimensions. Proceedings of 1st Annual SIGAL International Symposium on Algorithms, Springer-Verlag, Lecture Notes in Computer Science 450:310–317.

Bai Z.-D. and He X. 1999. Asymptotic distributions of the maximal depth estimators for regression and multivariate location. The Annals of Statistics 27:1616–1637.

de Berg M., van Kreveld M., Overmars M., and Schwarzkopf O. 1997. Computational Geometry: Algorithms and Applications, Springer-Verlag, Berlin.

Cleveland W.S. 1993. Visualizing Data, Hobart Press, Summit, New Jersey.

Cole R., Sharir M., and Yap C.K. 1987. On k-hulls and related problems. SIAM Journal on Computing 15:61–77.

Dey T. 1998. Improved bounds on planar k-sets and related problems. Discrete and Computational Geometry 19:373–382.

Dobkin D.P. and Souvaine D.L. 1987. Computational Geometry – A User's Guide. Chapter 2 in: Schwartz J.T and Yap C.K. (Eds.), Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics, Lawrence Erlbaum Associates, pp. 43–93.

Donoho D.L. and Gasko M. 1992. Breakdown properties of location estimates based on halfspace depth and projected outlyingness. The Annals of Statistics 20:1803–1827.

Edelsbrunner H. 1987. Algorithms in Combinatorial Geometry, Springer-Verlag, Berlin.

Edelsbrunner H. and Guibas L.G. 1989. Topologically sweeping an arrangement. Journal of Computer and System Sciences 38:165–194.

Edelsbrunner H., Guibas L. J., and Stolfi J. 1986. Optimal point location in a monotone subdivision. SIAM Journal of Computing 15:2:317–340.

Edelsbrunner H. and Souvaine D. 1990. Computing median-of-squares regression lines and guided topological sweep. Journal of the American Statistical Association 85:115–119.

Hodges J.L. 1955. A bivariate sign test. The Annals of Mathematical Statistics 26:523–527.

Kirkpatrick D. 1983. Optimal search in planar subdivisions. SIAM Journal on Computing 12:28–35.

Lal A. and Souvaine D. 2002. Finding the Tukey median of a planar point set using randomization. In preparation.

Langerman S. and Steiger W. 2000. Computing a Maximal Depth Point in the Plane. Proceedings 4th Japan Conference on Discrete and Computational Geometry, Lecture Notes in Computer Science, to appear.

Liu R.Y. 1990. On a notion of data depth based on random simplices. The Annals of Statistics 18:405–414.

Liu R.Y., Parelius J., and Singh K. 1999. Multivariate analysis by data depth: descriptive statistics, graphics and inference. The Annals of Statistics 27:783–840.

Liu R.Y. and Singh K. 1997. Notions of limiting P values based on data depth and bootstrap. Journal of the American Statistical Association 92:266–277.

Matoušek J. 1991. Computing the center of planar point sets. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 6:221–230.

Mitchell T. 2002. Point location in a nested set of convex hulls. Senior Design Project, EECS Department, Tufts University. `http://www.eecs.tufts.edu/r/geometry/depthquery`

Preparata F. and Shamos M. I. 1985. Computational Geometry: An Introduction, Springer-Verlag, New York.

Rafalin E., Souvaine D., and Streinu I. 2002. Topological Sweep in Degenerate cases. Algorithms Engineering and Experiments (ALENEX 2002). Springer-Verlag Lecture Notes in Computer Science 2409:155–165.

Rosenberger H. 1990. Topological plane sweep implemented in `C`, University of Illinois at Urbana-Champaign.

Rousseeuw P.J. 1984. Least median of squares regression. Journal of the American Statistical Association 79:871–880.

Rousseeuw P.J. and Ruts I. 1996. Algorithm AS 307: Bivariate location depth. Applied Statistics (JRSS-C) 45:516–526.

Rousseeuw P.J. and Ruts I. 1998. Constructing the bivariate Tukey median. Statistica Sinica 8:827–839.

Rousseeuw P.J., Ruts I., and Tukey J.W. 1999. The bagplot: A bivariate boxplot. The American Statistician 53:382–387.

Rousseeuw P.J. and Struyf A. 1998. Computing location depth and regression depth in higher dimensions. Statistics and Computing 8:193–203.

Rousseeuw P.J. and Struyf A. 2000. Characterizing angular symmetry and regression symmetry. Technical report, University of Antwerp, submitted.

Ruts I. and Rousseeuw P.J. 1996. Computing depth contours of bivariate point clouds. Computational Statistics and Data Analysis 23:153–168.

Sarnak N. and Tarjan R.E. 1986. Planar point location using persistent search trees. Communications of the ACM 29:669–679.

Souvaine D. and Steele J.M. 1987. Efficient time and space algorithms for least median of squares regression. Journal of the American Statistical Association 82:794–801.

Tukey J.W. 1975. Mathematics and the picturing of data. Proceedings of the International Congress of Mathematicians, Vancouver, 2:523–531.

Tukey J.W. 1977. Exploratory Data Analysis, Addison-Wesley, Reading, MA.

Wenger R. 1997. Helly-Type theorems and geometric transversals. In J. E. Goodman and J. O'Rourke, editors, Handbook of Discrete and Computational Geometry, pp. 63–82, CRC Press.