Fast Implementation of Depth Contours using Topological Sweep[*]

Kim Miller[†] Suneeta Ramaswami[‡] Peter Rousseeuw[§] Toni Sellarès[¶]
Diane Souvaine[†] Ileana Streinu[‖] Anja Struyf[§]

Abstract

The concept of location depth was introduced in statistics 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. These require the computation of depth regions, which form a collection of nested polygons. The center of the deepest region 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 depth contours in $O(n^2)$ time and space, using topological sweep of the dual arrangement of lines. Once the contours are known, the location depth of any point is computed in $O(\log^2 n)$ time. We provide fast implementations of these algorithms to allow their use in everyday statistical practice.

1 Introduction

The location depth of a given point a relative to a bivariate data set first occurred (without being given a name) as a test statistic of Hodges [8] for the hypothesis that a is the center of the probability distribution from which the data was drawn. More recently, Liu and Singh [11] constructed other statistical tests based on location depth.

Definition. Let $P = \{p_1, \ldots, p_n\}$ be a finite set of points in R^2 and a be an arbitrary point, not necessarily in P. The *location depth* of a relative to P is the minimum number of points of P lying in any closed halfplane determined by a line through a.

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

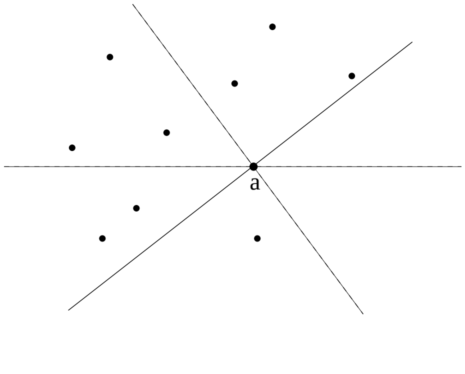

Figure 1: Point a has location depth 1.

[*] This research started at the Workshop on Applications of Computational Geometry in Statistics, organized by Godfried Toussaint at the Bellairs Research Institute in Barbados, Feb. 1999.

[†] Department of Electrical Engineering and Computer Science, Tufts University, Medford, MA 02155. <kmillr,dls>@eecs.tufts.edu. Partially supported by NSF grant #EIA-99-96237 and by *Forward in SEM*.

[‡] Contact author. 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,Anja.Struyf>@uia.ua.ac.be

[¶] Institut d'Informàtica i Aplicacions, Universitat de Girona, Spain. sellares@ima.udg.es. Partially supported by grants MEC-DGES-SEUID PB98-0933 of the Spanish Government and DURSI 1999SGR00162 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

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

Tukey [18] proposed to use depth contours in a graphical display of the data. The k-th contour is the "inner" intersection of all halfplanes of a set yielding the same location depth (see Figure 2). Note that depth contours are different from convex layers; they have as vertices both points from the original set and new points from the intersection of halfplanes of the same contour. The maximum depth (over all a) of P is denoted by k^*, and from Helly's theorem it follows that $k^* \geq \lfloor n/3 \rfloor$ is always true (this fact is also known as the existence of at least one centerpoint of P).

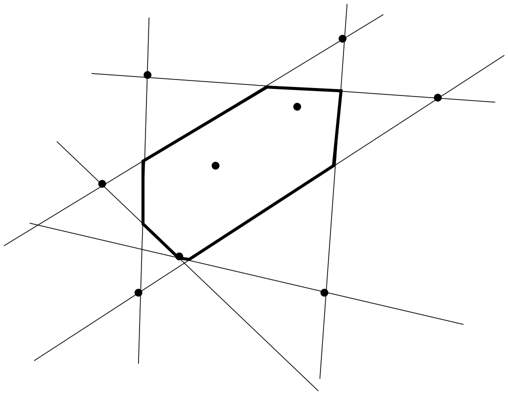

Figure 2: Depth contour 2 is drawn in bold.

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

Donoho and Gasko [5] showed that T^* is a location estimator with several desirable properties. It is robust in the statistical sense, meaning that it does not change much when there are some outlying data points. (Such outliers can be the result of measurement errors, exceptional phenomena, heterogeneity of the population, or other deviations of the standard assumptions.) The Tukey median has a good breakdown value (making it more robust than the median based on simplicial depth), and it is equivariant under affine transformations (unlike the spatial median, which is based on distances).

We also need to compute the *bag*, first proposed by Rousseeuw, Ruts, and Tukey [15], which is part of a larger construct called the *bagplot*. The bagplot is a two-dimensional generalization of Tukey's univariate boxplot [19], and provides a visual representation of the location, spread, correlation, skewness, and tails of the data. The bag is a convex polygon containing at most 50% of the original data points. It is determined by an interpolation between the two adjacent depth contours that enclose more than half, and less than half, respectively, of the original points.

1.1 Previous implementations and our results

Theoretical complexity results on depth contours, or k-hulls, have been known for some time [2]. The best known theoretical result for computing the Tukey median is an $O(n \log^5 n)$ algorithm by Matoušek [12], but its complex structure makes it an unlikely candidate for practical applications. Fast implementations of depth contours are critical for several statistical applications. Not many implemented algorithms were available. The program ISODEPTH [17] 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 [14] and BAGPLOT [15] 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 implementation 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 bounds. All our implementations are robust in the algorithmic sense, i.e. numerically stable. Our code allows collinearities in the point set, but currently assumes that no two points have the same x-coordinate. We also give empirical results on the dramatic speed-up (of more than 300 for data sets of size 700 and higher) provided by our implementation over those of ISODEPTH and HALFMED.

2 The Algorithms

2.1 Determining the Depth Contours

Constructing the depth contours involves three major steps.

1. First we use a standard dual mapping to define an arrangement of lines.
2. Then we use topological sweep to efficiently find all intersections of lines within the arrangement. Each intersection in the dual corresponds to a line between two points in the primal. As we find each intersection we can determine the depth contour to which the corresponding halfplane potentially belongs.
3. Finally, for each contour, we find the intersection of its halfplanes to give us the convex depth contour.

This method was first suggested by Cole, Sharir, and Yap [2], but without using topological sweep. We also make explicit several important details that are necessary for the efficient *implementation* of depth contours, the focus of our work.

Creating the Arrangement. We use a standard dual mapping of points to lines (e.g. [6, 4]) $p(a, b) \rightarrow l : y = -ax + b$ to create the arrangement. The mapping preserves the order of points along the x-axis (as the slope of the lines). A line through two points in the primal maps to the intersection point of the corresponding lines in the arrangement. The mapping preserves the above/below relationship: if a point is above a line between two points in the primal, then the corresponding line is above the corresponding point in the dual.

For every intersection point in the arrangement, we determine the contour to which the corresponding halfplane in the primal contributes. 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 I (resp. below I) exactly equals the number of points lying above L (resp. below L) in the primal. Let m be the number of crossings above I. Let r be the number of crossings below I. The contour to which L contributes is therefore the minimum of m and r (this is also said to be the *level* of I). After the intersections have been examined we are left with a list of halfplanes/intersections for each potential contour. See Figure 3.

Each depth contour is the "interior" intersection of the halfplanes that potentially belong to this contour. During the sweep, the halfplanes for each contour are split into upper and lower sets. If m was the minimum, more points lie "below" the halfplane in the primal and therefore it belongs to the upper set. Conversely, if r was the minimum, it belongs to the lower set.

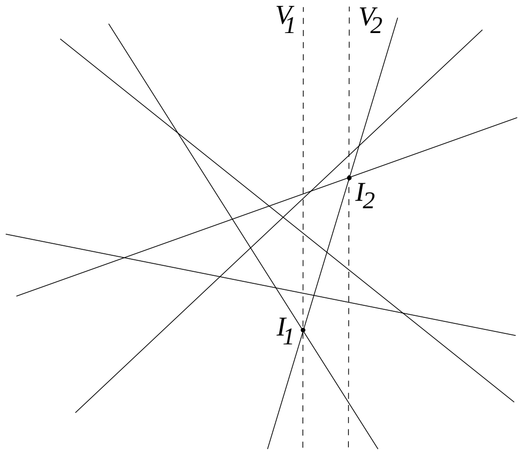

Figure 3: Intersection point I_1 has 4 lines above and 0 line below; hence it belongs to the lower set of the first contour. Intersection point I_2 has 1 line above and 3 lines below; hence it belongs to the upper set of the second contour.

The lines containing the edges of the lower convex hull of the upper set of dual intersections correspond to the vertices of the upper boundary of the contour in the primal. The lines containing the edges of the upper convex hull of the lower set of dual intersection correspond to the vertices of the lower boundary of the contour. The intersection of the lower hull of the upper set and the upper hull of the lower set is the complete contour. If the intersection is empty, the depth contour does not exist.

Topological Sweep. Topological sweep allows the traversal of all vertices of an arrangement of lines in $O(n^2)$ time and linear space [7]. The typical vertical sweepline method maintains a priority queue (heap) of potential next points in the vertical sweep, at a cost of $O(\log n)$ per update and a total time complexity of $O(n^2 \log n)$ and linear space. Explicit construction of the arrangement as a planar graph uses $O(n^2)$ time and space. Topological sweep improves the time complexity for finding all intersections by maintaining a data structure called the horizon tree (efficiently implemented using arrays). The horizon tree stores one line segment per level of the arrangement and uses a stack that contains all points of intersection of current line segments on adjacent levels, represented as array indices. The algorithm ostensibly sweeps a curved line across the arrangement, over the intersection points of currently incident line segments. Each line segment has at most two neighbors, so the stack remains linear in size. If we are computing all depth contours, however, all $\Theta(n^2)$ intersection points of the arrangement are saved to generate a halfplane for the computation of contours, producing quadratic space complexity.

Intersection of Halfplanes. The upper and lower hulls of a set of points can be computed in linear time, if the points are given in sorted order. After the topological sweep, we have the sorted list of intersection points that form the upper and lower sets for each contour. Hence, we find the lower convex hull of the upper set, and the upper convex hull of the lower set in time linear in the size of the sets. Note that while the size of the k-th depth contour is always $O(n)$ [12], the best known upper bound on the size of the k-level is $O(nk^{1/3})$ [3]. The hulls represent the upper envelope and lower envelope of the corresponding halfplanes in the primal. See Figure 4. The two hulls are then intersected, which gives us the k-contour (if it exists) in time linear in the size of the k-level. The total run-time of this step is $O(n^2)$.

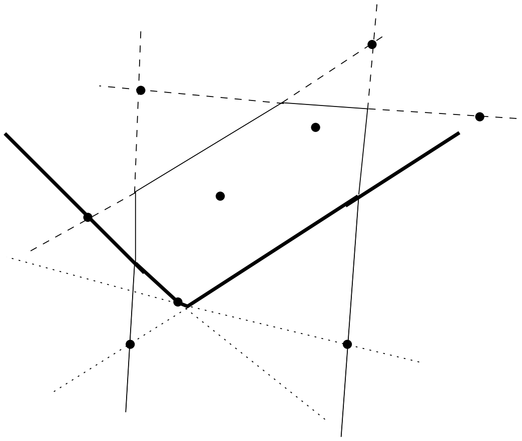

Figure 4: The dashed (dotted) lines are the halfplanes associated with the upper (lower) set of intersection points in the arrangement.

2.2 Location Depth of a Single Point

After Contour Calculations After calculating all the depth contours, finding the depth of a single point is reduced to $O(\log^2 n)$ complexity. The initial contour calculations provide a set of concentric convex polygons representing each depth contour. Determining if a point is inside or outside of a convex polygon can be done in $O(\log n)$ time. To determine the depth of a single point we can thus perform a binary search on the depth contours to achieve the result in $O(\log^2 n)$ time.

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 as follows: As before, consider a point set P and a point

a. The minimum number of points found on either side of the lines $\overline{pa}, p \in P$, is the depth of a in P. The first step in the algorithm is to sort the points p in P radially around a and store the slope of line $\overline{pa}$ in an array. Let's say the slopes are stored counterclockwise in the array (around a). The array can then be sectioned off into "quadrants". The first quadrant represents all points to the left and above a, the second all points left and below, the third all points below and right, the fourth all points above and right. Then, for each point(slope) p in the array, we index the slope in only the opposite quadrant of the array using binary search. Let's call this index i_2. The number of points to either side of the line $\overline{pa}$ can then be calculated by subtracting the index of p from i_2 and i_2 from the index of p modulo the array size.

Recently it was shown [9] 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.

2.3 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 within, 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.

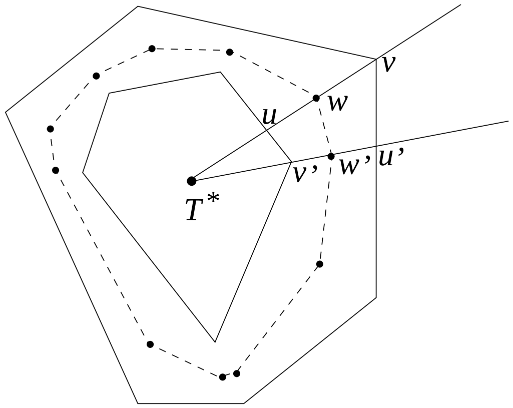

Figure 5: Determining the vertices of the bag

3. Finally, the bag is constructed by interpolating between the two contours by using the λ value. For each vertex on D_k or D_{k-1} let l be the line extending 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 will lie 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 5.

Figure 6 shows the bagplot of the engine displacement versus the weight of 60 cars (the data can be found in [1]). The Tukey median T^* is depicted by a cross, and the dark grey area around the median is the bag, which gives us an idea of the shape of the data cloud. Here we see that there is a positive correlation between the two variables. The light grey area is the convex hull of the bag and the non-outliers. A data point is considered a non-outlier if it lies inside of the fence (not displayed in the bagplot), which is obtained by inflating the bag by a factor 3 relative to T^*. The computation of the fence and the light grey area are immediate if the Tukey median and the bag are known. Here we have four outlying observations, depicted by stars, in the upper right corner of the plot. These four cars have a high engine displacement relative to their weight.

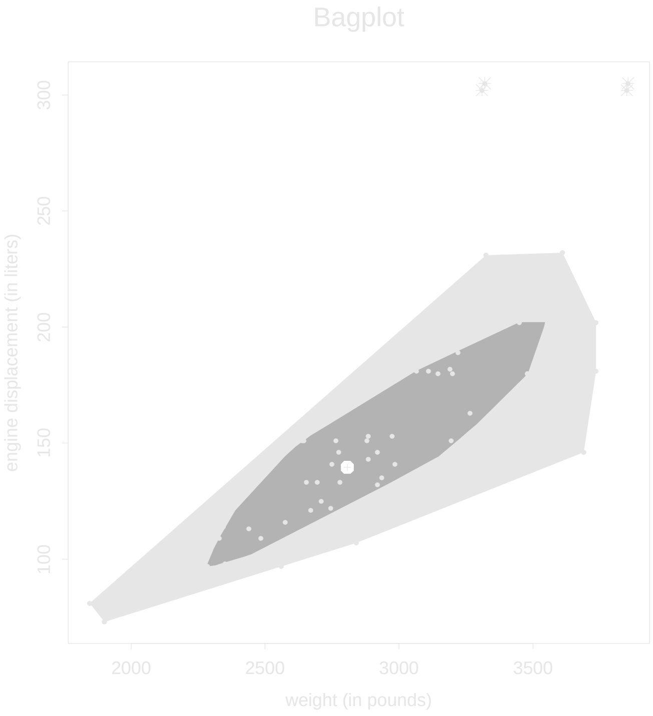

Figure 6: Bagplot of car weight versus engine displacement of 60 cars.

3 The Implementations

Our implementation of the algorithms for computing depth contours and the bagplot was carried out largely in response to requests made by statisticians who wish to use the code for practical purposes. 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 of a set of n planar 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 [10]. 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 7 and 8 for examples of the graphical output produced by our program.

Once the contours are available, computing the

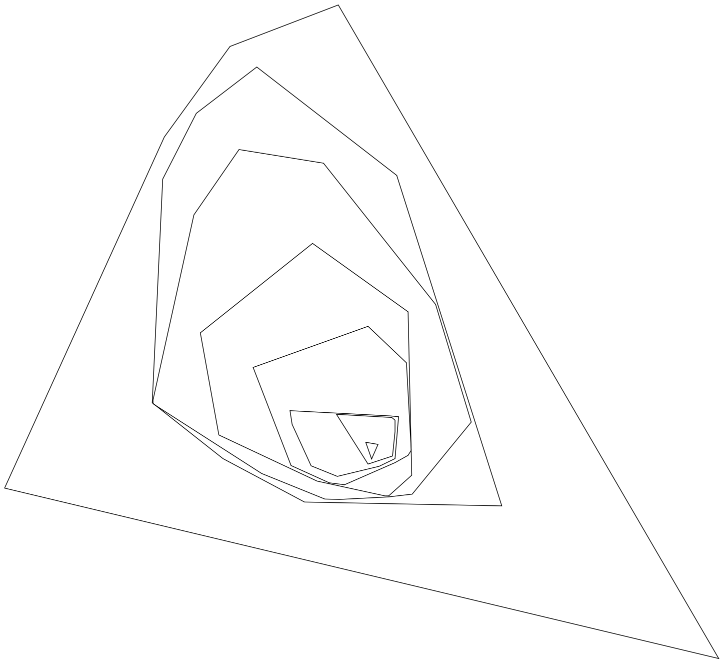

Figure 7: Depth contours produced by our program for a set of 20 points.

bag requires only linear time. Our implementation referred to the C code for topological sweep given in [13], but we re-wrote 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 input set of points, but currently assumes that no two points have the same x-coordinate. All primitive geometric computations rely on LEDA and inherit their robust implementation.

3.3 Significance

The testing phase consists of run-time comparisons for computing the Tukey median. To check the correctness of our code, we first ran it on sets of data for which the Tukey median was known previously. As mentioned earlier, the previously best-known implementation for computing the Tukey median is HALFMED [14], 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 implementation 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 points sets of each size were generated, and the total run-time for each size was recorded using perl benchmark scripts. The average run-times for HALFMED and our code are shown in Table 1.

As Table 1 shows, our implementation provides a dramatic speed-up over that of HALFMED for input sets of size greater than 40. For smaller data sets, the overhead of computing the arrangement and using more sophisticated data structures causes the run-time to be slower than for

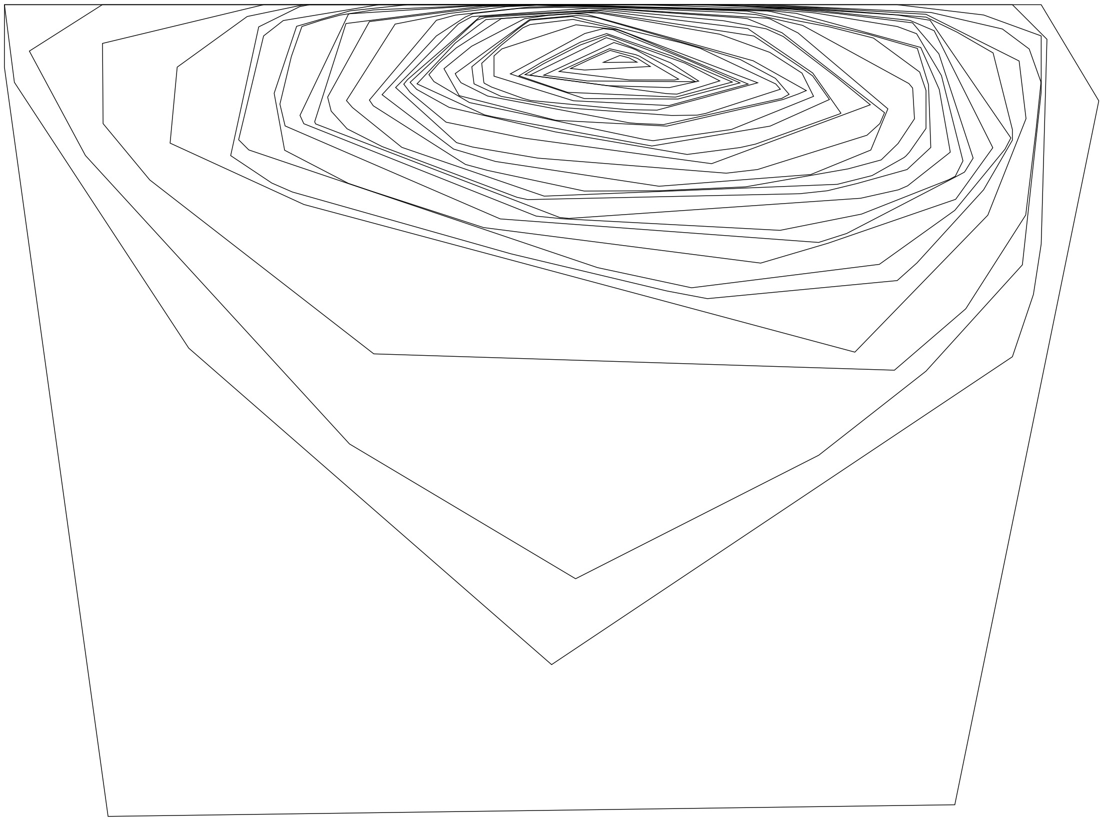

Figure 8: Depth contours produced by our program for a set of 79 points.

HALFMED. However, the improvement in runtime for larger data sets is notable, and of practical significance to statisticians. The graph in Figure 9 illustrates the speed-up of our implementation over that of HALFMED.

4 Concluding Remarks

We have given fast implementations for computing all depth contours and consequently, the Tukey median and the bag. Our results provide a dramatic speed-up over existing implementations. The code, which requires LEDA 3.8 or higher, is available at `http://www.eecs.tufts.edu/~dls/locdepth`. Our code currently assumes that no two points in the input set have the same x-coordinate. This assumption can be easily removed, for example, by an appropriate linear transformation of the input set. In the next version of our program, we intend to remove this non-degeneracy assumption.

A natural open question, and one of particular interest to statisticians, is on faster implementations for computing only the Tukey median. Theoretically, faster algorithms are known [12]. 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 from a practical point of view. We are currently working on an implementation of a randomized version of Matoušek's algorithm [12] for computing the Tukey median.

n	Average CPU time over 10 runs (seconds)	
	HALFMED	Our code
10	0.034	2.098
20	0.094	2.060
40	.607	2.085
80	5.599	2.202
100	11.822	2.335
160	37.222	2.822
200	84.449	3.075
320	490.231	4.481
500	759.655	6.011
640	1142.619	8.317
750	4953.736	14.231
1000	9002.314	22.071

Table 1: Average run-times.

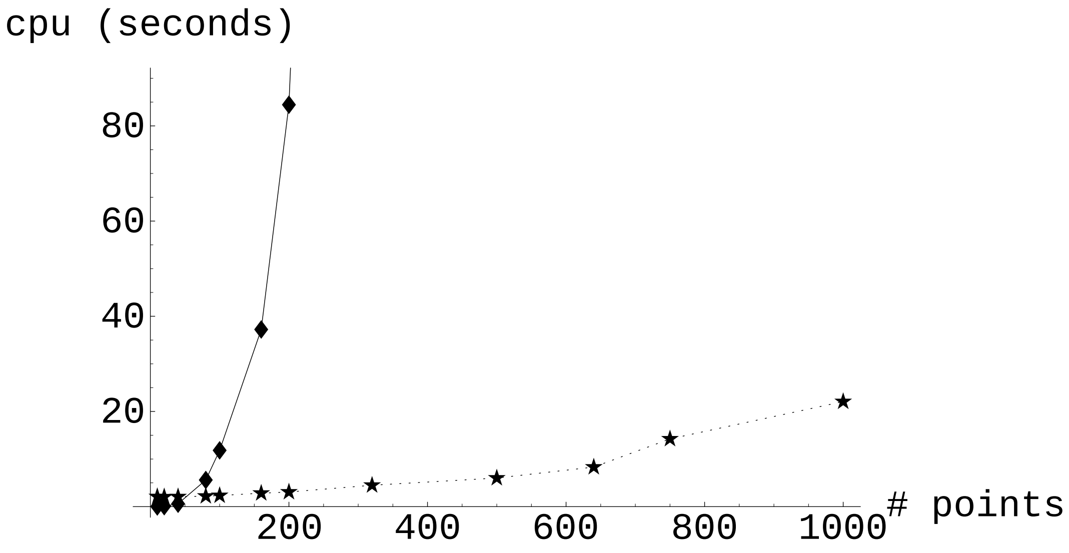

Figure 9: Comparison of run-times: HALFMED (solid) vs. our implementation (dotted)

References

[1] J. M. Chambers and T. J. Hastie. *Statistical Models in S.* Chapman and Hall, London, 1993.

[2] R. Cole, M. Sharir, and C. K. Yap. On k-hulls and related problems. *SIAM Journal on Computing*, 15(1):61–77, 1987.

[3] T. Dey. Improved bounds on planar k-sets and related problems. *Discrete Comput. Geom.*, 19:373–382, 1998.

[4] D. P. Dobkin and D. L. Souvaine.. Computational Geometry – A User's Guide. **Chapter 2** of *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics*, J. T. Schwartz and C. K. Yap, eds., Lawrence Erlbaum Associates, 1987, 43-93.

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

[6] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin, 1987.

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

[8] J. L. Hodges. A bivariate sign test. *Annals of Mathematical Statistics*, 26:523–527, 1955.

[9] S. Langerman and W. Steiger. An optimal algorithm for hyperplane depth in the plane. *Proceedings Eleventh Symposium on Discrete Algorithms*, 54–59, 2000.

[10] LEDA. Library of Efficient Data structures and Algorithms. `www.ag2.mpi-sb.mpg.de/LEDA`

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

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

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

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

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

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

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

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

[19] J. W. Tukey. *Exploratory Data Analysis.* Addison-Wesley, Reading, MA, 1977.