On the Computation of the Bivariate Median and the Fermat-Torricelli Problem for Lines

Technical Report SOCS-01.2

Greg Aloupis* Michael Soss* Godfried Toussaint*

February 28, 2001

Abstract

Two definitions for the median of a set of points S in R^2 are the *simplicial median* and the *Oja median*. Both may be used as robust estimators of location. The fastest algorithms to date for computing these medians are described by Rousseeuw and Ruts, and have a complexity of $O(n^5 \log n)$ time and $O(n)$ space. We supply algorithms which reduce the time complexity: we compute the Oja median in $O(n^3 \log^2 n)$ time using $O(n^2)$ space, and we compute the simplicial median in $O(n^4)$ time and space *or* in $O(n^4 \log n)$ time and $O(n^2)$ space. Our results on the Oja median also allow us to compute the Fermat-Torricelli points of n lines in $O(n^2)$ time and space *or* in $O(n^2 \log n)$ time and $O(n)$ space.

1 Introduction

The use of multidimensional medians as robust estimators of location has been studied extensively by computer scientists and statisticians. In general, a median is the location in R^d which maximizes a certain depth definition with respect to a data set. Applications include data description, multivariate

*School of Computer Science, McGill University. {athens,soss,godfried}@cs.mcgill.ca

confidence regions, p-values, quality indices, and control charts [RR96]. An introduction to the topic including definitions and properties of the most common medians is given in [Sma90]. Below, we include descriptions for the simplicial median of Liu [Liu90] and for the Oja median [Oja83].

Let $S = \{s_1, \ldots, s_n\}$ be a set of data points in R^d. The *Oja depth* of a point θ in R^d with respect to S is the cumulative volume of all simplices formed by θ and a subset of d elements from S. To find the Oja depth of θ in R^2, we sum the areas of all triangles formed by θ and two points of S. The Oja median is any point in R^d with maximum Oja depth.

The *simplicial depth* of a point θ in R^d with respect to S is the number of simplices formed by $d+1$ elements of S that contain θ. To find the simplicial depth of θ in R^2, we must find how many triangles formed by triples of points in S contain θ. The simplicial median is any point in R^d with maximum simplicial depth.

Rousseeuw and Ruts [RR96] have presented an algorithm for computing bivariate simplicial depth in $O(n \log n)$ time. A matching lower bound is presented in [ACG$^+$01]. Rousseeuw and Ruts mention that their algorithm leads to a straightforward computation of the simplicial median in $O(n^5 \log n)$ time since only the $O(n^4)$ intersection points of segments between pairs of data points need to be checked. We mention that such intersection points can be reported in $O(n^2 \log n + k)$ time and $O(n^2)$ space, where k is $O(n^4)$ [Bal95]. k is also expected to be $O(n^4)$ for uniform distributions of points. So unless k is known to be small for some distribution of points, we can save the space used by computing all intersection points in a straightforward manner in $O(n^4)$ time and $O(n)$ space.

Rousseeuw and Ruts also briefly describe how their methods can be used to speed up the algorithm of Niinimaa, Oja and Nyblom [NON92] for computing the Oja bivariate median from $O(n^6)$ to $O(n^5 \log n)$ time.

Recently, Roy Barbara [Bar00] showed how to compute the *Fermat-Torricelli points of n lines* in $O(n^3)$ time. Fermat and Torricelli are known for first solving the problem of finding a point which minimizes the sum of distances to three given points, and the generalized problem with n points also bears their name (see [GS98, dF34, LV19]). Barbara refers to the set of points with minimum sum of weighted distances to n given lines as the Fermat-Torricelli points of the n lines. Barbara proves that these points must be found on intersections of the given lines. He then suggests computing the sum for each intersection point, which is straightforward to do in $O(n^3)$ time using $O(n)$ space.

In section 2 we present our algorithms for computing the Oja median. Algorithm *Oja–1* has a time complexity of $O(n^3 \log^2 n)$ and uses $O(n^2)$ space, in the *RAM* model of computation. We refer the reader to [PS85] for an introduction to this model. Our gradient descent algorithm, *Oja–2*, uses $O(n^4)$ time and space *or* $O(n^4 \log n)$ time and $O(n^2)$ space. It may be also be used to find the Fermat-Torricelli points of n lines in $O(n^2)$ time and space *or* in $O(n^2 \log n)$ time and $O(n)$ space. In section 3 we present our algorithm for computing the simplicial median. The complexity is the same as for algorithm *Oja–2*.

2 Oja Median Algorithms

2.1 Properties of the Oja median in R^2

Some properties of the Oja median in R^2 are first stated and proven. Let L be the set of lines formed by each pair of points in a data set S. Note that by "Oja median" we mean one of point for which Oja depth is minimum. If other such points exist it is an easy matter to find them, as should be apparent after the proof of lemma 4.

Lemma 1 *The Oja depth of a point θ with respect to S can be expressed as a weighted sum of the distances from θ to each line in L.*

Proof. By definition, Oja depth is the sum of the areas of all triangles (θ, s_i, s_j), where $1 \leq i < j \leq n$. The area of each triangle may be expressed as $\frac{1}{2}bh$, where b is the distance from s_i to s_j and h is the orthogonal distance from the point θ to the line $\overline{s_i s_j}$.

□

Lemma 2 *The Oja median must be on one of the intersection points of the lines in L.*

Proof. Each cell formed by the arrangement of lines in L can be considered to be a linear program[1] , where the feasible region is determined by the edges of the cell, and the function to be minimized is the weighted sum of lemma 1. Therefore if the median is located within a given cell, it must be on one of its vertices. The median cannot be at infinity (as part of an open cell) since the Oja depth would be infinite, as is apparent by lemma 1.

[1]For an introduction to linear programming, see [Chv83].

□

Lemma 3 *The Oja depth along any line ℓ in the plane varies in a piecewise-linear manner, with inflection points occurring only where ℓ crosses lines in L.*

Proof. The rate of change of Oja depth is constant along ℓ within any cell of the arrangement of lines, since the cell represents a linear program. As ℓ crosses from one cell to the next, it crosses from one linear program to another. The transition occurs at the common edge of two cells, which is on a line in L.

□

Lemma 4 *The Oja depth along a line ℓ has only one local minimum, from which the rate of change of Oja depth increases monotonically in both directions.*

Proof. Suppose that the Oja depth is decreasing as we move along ℓ in some direction, and that along this direction we encounter line m which is a member of L. This encounter does not affect the contribution of any other line in L to Oja depth. Since crossing m can only increase m's contribution to Oja depth, the result is that Oja depth cannot decrease at a faster rate (it can only increase, or decrease at a slower rate). Since Oja depth decreases from infinity as we approach the members of L from either direction on ℓ, the lemma is proven by induction. The local minimum can be a point or a segment between two intersection points.

□

Corollary 5 *The points in R^2 with minimum Oja depth form a convex set with $O(n)$ vertices on its boundary.*

Proof. Lemma 4 implies that if more than two points exist with minimum Oja depth, they must be vertices of one cell. Every data point can contribute at most two lines to the boundary of a cell, so every cell has $O(n)$ segments on its boundary. Thus the number of intersection points with minimum Oja depth is $O(n)$.

□

Corollary 6 *The Oja depth function is convex.*

Proof. A function's surface is convex if every line segment between two points on the surface does not intersect other surface points. Given two points on the surface of the Oja depth function, we can construct a vertical plane containing the two points, which projects to a line in R^2. By lemma 4, Oja depth is convex along this line, so the segment joining the two points does not intersect the surface of the function.

□

Lemma 7 *If the Oja depth decreases along two directions u and v from a point p, then it must also decrease in all directions between u and v (within the acute angle).*

Proof. All points along u or v from p have depth less than p. Between any point along u and any point along v we can form a line, along which the depth cannot have a local maximum, by lemma 4. Therefore the depth along this line and between the two points cannot be greater than both, which implies that the depth cannot be greater than p.

□

Lemma 7 also implies that from any point p, the directions in which depth decreases must span less than 180^o.

Lemma 8 *The minimum Oja depth along any line ℓ can be found in $O(n^2 \log n)$ time.*

Proof. By lemma 1, finding the Oja depth of a point involves computing the weighted distance from the point to the $O(n^2)$ lines in L, which can be done in $O(n^2)$ time. Computing and sorting the intersection points of ℓ with all lines in L can be done in $O(n^2 \log n)$ time. While computing the depth of the first sorted intersection point, we can also calculate the local rate of depth change along ℓ. Thus the depth of its adjacent intersection point in the sorted list may be computed in constant time by multiplying the rate of change by the distance required to reach it. Then the rate of change may be updated in constant time, by reversing the contribution of the local intersecting line. Therefore after computing the depth of one point, $O(n^2)$ time is sufficient to find the depth of all points on the line. The overall complexity is dominated by the sorting step : $O(n^2 \log n)$.

□

2.2 The two algorithms

Algorithm *Oja–1*

For each point s_i in S do the following:

1. Sort radially the lines determined by s_i and every other point of S.
2. Perform a binary search on the sorted lines: each step consists of computing the point d with minimum depth along one line, determining which side of that line contains directions where the depth does not increase from d, and proceeding in that region of space.
3. If the binary search ends with one line remaining, halt and return d.

Theorem 2.1 *Algorithm* Oja–1 *computes the bivariate Oja median of n points in $O(n^3 \log^2 n)$ time and $O(n^2)$ space.*

Proof. Binary search is made possible by lemma 7, which implies that for a point with minimum depth along a line, the depth must increase in all directions of at least one side of the line. The region in which to proceed may be found by using the gradient of the Oja depth function, which can be computed in $O(n^2)$ time by Niinimaa, Oja and Nyblom [NON92] or in $O(n \log n)$ time by Rousseeuw and Ruts [RR96]. Thus the complexity of one step in the binary search is dominated by calculating the minimum depth on a line, which takes $O(n^2 \log n)$ time by lemma 8. The binary search continuously shrinks a wedge in the plane where the Oja median may be found, or in other words expands the region where the median cannot be found. Since by lemma 2 the median must be located on one of the lines in L, there must exist a data point for which this wedge will shrink to a single line. The minimum depth along this line is the Oja median. $O(n^2 \log n) * O(\log n)$ time suffices for one data point. Thus the total complexity is $O(n^3 \log^2 n)$ for all data points. The amount of space used is $O(n^2)$, for computing the minimum depth on a line.

□

Algorithm *Oja–2*

Choose any line in L and label it as ℓ.

1. Find the point p with minimum Oja depth on ℓ.

2. Find the intersecting line at p which has the steepest rate of change of Oja depth (this is done in a similar way to finding which region to proceed in during algorithm *Oja–1*). If the steepest slope is not negative then stop. Otherwise, label the line with steepest slope as ℓ and go to step 1.

Theorem 2.2 *Algorithm* Oja–2 *computes the bivariate Oja median of* n *points in* $O(n^4 \log n)$ *time and* $O(n^2)$ *space or alternatively in* $O(n^4)$ *time and space.*

Proof. Due to lemma 7 we know that once the region on one side of a line is rejected, it will never be re-visited by a gradient descent. There are no local minima other than the global minimum, by lemma 4. Therefore the descent is guaranteed to succeed after visiting $O(n^2)$ lines. By lemma 8, $O(n^2 \log n)$ time suffices for step 1. Since all lines may be visited, the total complexity amounts to $O(n^4 \log n)$. Only one line is processed at a time, so the space used does not exceed $O(n^2)$. As a preprocessing step, we could compute the arrangement of L in $O(n^4)$ time and space [O'R95]. Then sorting could be avoided when computing the minimum depth on a line ℓ, so the minimum depth on ℓ could be computed in $O(n^2)$ time. This results in $O(n^4)$ time and space complexity overall.

□

Note that we could also simply find the minimum depth on each of the $O(n^2)$ lines without using a gradient descent. The complexity would only differ by a constant factor.

2.3 The Fermat-Torricelli points of n lines

In section 2.1, lemmas 1 and 2 imply that the Oja median is a Fermat-Torricelli point of L. However we cannot use algorithm *Oja–1* for any set of lines, since this algorithm relies on the fact that a large number of lines pass through each point in S. Instead algorithm *Oja–2* may be used since it assumes nothing about the position of the lines. Therefore we have the following theorem:

Theorem 2.3 *The Fermat-Torricelli points of n lines may be computed in $O(n^2 \log n)$ time and $O(n)$ space, or in $O(n^2)$ time and space, using algorithm* Oja–2.

3 Simplicial Median Algorithm

Let S be a data set of n points in R^2, and I be the set of line *segments* formed between every pair of data in S. For simplicity, assume that S is in general position, in the sense that no three points are collinear.

Lemma 9 *To find a point with maximum simplicial depth it suffices to consider the intersection points of segments in I.*

Proof. Consider the arrangement of cells whose boundaries are segments in I. All points in the interior of a given cell must have equal simplicial depth. Any point on the boundary of this cell must have depth at least equal to points in the interior. Finally, any vertex of the cell must have depth at least equal to any point on an adjacent boundary. These vertices are the intersection points of segments in I.

□

Algorithm *Simp-Med*

1. Calculate the simplicial depth of every data point in S.

2. Compute the number of points which are strictly to one side of each segment in I (each side separately).

3. For every segment ℓ in I,

   (a) compute and sort the intersection points of all other segments with ℓ, if there are any.

   (b) let MAX be the greatest depth among the endpoints of ℓ.

(c) if there exist other intersection points on ℓ, calculate the simplicial depth d of the intersection point adjacent to MAX. Update MAX (if $MAX < d$, MAX$\leftarrow$ d).

(d) continue through the sorted list: every time an intersecting line is left behind, subtract from d the number of points strictly behind the line. Every time a line is encountered, add to d the number of points strictly ahead of that line. Update MAX after encountering each intersection.

4. Exit with the greatest MAX value found over all segments, and the associated intersection point.

Theorem 3.1 *Algorithm* Simp-Med *computes the bivariate simplicial median of n points in $O(n^4 \log n)$ time and $O(n^2)$ space or alternatively in $O(n^4)$ time and space.*

Proof. All intersection points are checked, so by lemma 9 a point of maximum simplicial depth is found. Step 1 can be done $O(n^2 \log n)$ time in total since the calculation of simplicial depth for a point takes $O(n \log n)$ time with the algorithm of Rousseeuw and Ruts [RR96]. Step 2 is straightforward to do in $O(n^3)$ time by brute force. It may also be done in $O(n^{2.695})$ time using the algorithm of Edelsbrunner and Welzl [EW86] for counting the number of points to one side of a query line[2]. Step 3a takes $O(n^2 \log n)$, 3b is constant, 3c is $O(n \log n)$, and 3d takes $O(n^2)$ time. Thus step 3 takes $O(n^4 \log n)$ once run for every segment, and dominates the time complexity of this algorithm. The space used is $O(n^2)$ for step 3a. As with algorithm *Oja–2*, if we compute the arrangement of I beforehand then step 3a is no longer necessary. Computing the arrangement of I would take $O(n^4)$ time and space and would dominate the complexity of the algorithm.

□

Figure 1 illustrates the logic behind step 3 of algorithm *Simp-Med*. Once we know the depth of an intersection point (step 3c), we can compute all other depths along the given segment, each in constant time. Each intersecting segment forms a triangle with every point strictly to one side. Thus every

[2]For other sub-cubic algorithms and for a recent survey on similar problems of counting and reporting geometric objects, see [AE98].

time we encounter a segment, we enter as many triangles as there are points on the other side of this segment. Every time we leave a segment behind, we exit triangles formed by that segment and each point on the other side. So in fig 1, suppose we are moving from left to right along the current line, and just before encountering segment t, we have some value $d = 10$. Upon encountering t, we can see that we are now inside three new triangles: one for each point to the right of t. As soon as we move to the right of t, we will no longer be inside five triangles. So on t, $d = 13$, and immediately after, $d = 8$.

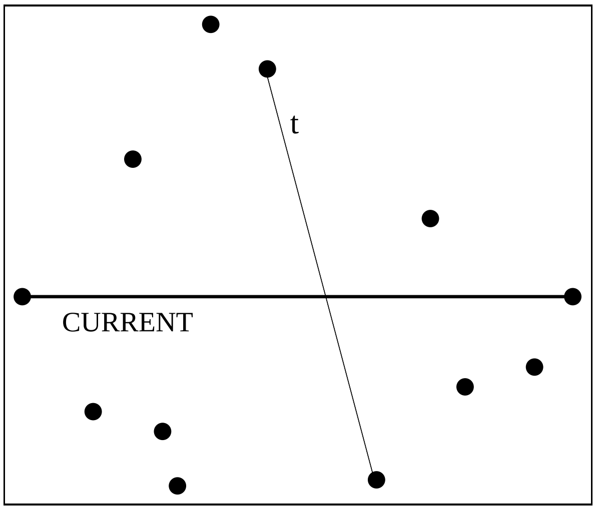

Figure 1: The current line traversed, and an intersecting segment t

4 Conclusion

We have reduced the time complexity of calculating the Oja and simplicial medians. The Oja median may be computed in $O(n^3 \log^2 n)$ time and $O(n^2)$ space. The simplicial median may be computed in $O(n^4 \log n)$ time with $O(n^2)$ space, or alternatively, in $O(n^4)$ time and space. This improves the results of Rousseeuw and Ruts, who computed each median in $O(n^5 \log n)$ time and $O(n)$ space.

Our algorithm *Oja–2* for the Oja median may be used to compute the Fermat-Torricelli points of n lines in $O(n^2 \log n)$ time and $O(n)$ space, or in

$O(n^2)$ time and space, thus improving the algorithm suggested by Barbara which takes $O(n^3)$ time and $O(n)$ space.

Furthermore, our proof that the solution must be on an intersection point is much shorter.

Acknowledgements

This research was started at the Workshop on Applications of Computational Geometry in Statistics, organized by Godfried Toussaint at the Bellairs Research Institute of McGill University in Barbados, February 1999. We thank all the participants for helpful discussions: Prosenjit Bose, Anna Bretcher, Carmen Cortes, Francisco Gomez, Michael Houle, Henk Meijer, Mark Overmars, Suneeta Ramaswami, Peter Rousseeuw, Toni Sellares, Diane Souvaine, Ileana Streinu and Anja Struyf.

References

[ACG+01] G. Aloupis, C. Cortes, F. Gomez, M. Soss, and G. Toussaint. Lower bounds for computing statistical depth. Technical Report SOCS-01.1, School of Computer Science, McGill University, February 2001.

[AE98] P. Agarwal and J. Erickson. Geometric range searching and its relatives. In J. Goodman B. Chazelle and R. Pollack, editors, *Advances in Discrete and Computational Geometry.* AMS Press, Providence, RI, 1998.

[Bal95] Ivan J. Balaban. An optimal algorithm for finding segment intersections. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 211–219, 1995.

[Bar00] R. Barbara. The Fermat-Torricelli points of n lines. *Mathematical Gazette*, 84:24–29, 2000.

[Chv83] V. Chvátal. *Linear Programming.* W. H. Freeman, New York, NY, 1983.

[dF34] P. de Fermat. Abhandlungen über maxima und minima. *Ostwalds Klassiker der exacten Wissenschaften, M. Miller (ed)*, 238, 1934.

[EW86] H. Edelsbrunner and Emo Welzl. Halfplanar range search in linear space and $O(n^{0.695})$ query time. *Inform. Process. Lett.*, 23:289–293, 1986.

[GS98] C. Groß and T. Strempel. On generalizations of conics and on a generalization of the Fermat-Torricelli problem. *American Mathematical Monthly*, 105(8):732–743, 1998.

[Liu90] R. Liu. On a notion of data depth based upon random simplices. *The Annals of Statistics*, 18:405–414, 1990.

[LV19] G. Loria and G. Vassura, editors. *Opere di Evangelista Torricelli*, volume 1. Faenza, 1919.

[NON92] A. Niinimaa, H. Oja, and J. Nyblom. Algorithm AS 277: The Oja bivariate median. *Applied Statistics*, 41:611–617, 1992.

[Oja83] H. Oja. Descriptive statistics for multivariate distributions. *Statistics and Probability Letters*, 1:327–332, 1983.

[O'R95] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1995.

[PS85] F. Preparata and M. Shamos. *Computational Geometry. An Introduction.* Springer-Verlag, 1985.

[RR96] P. Rousseeuw and I. Ruts. Bivariate location depth. *Applied Statistics*, 45:516–526, 1996.

[Sma90] C. Small. A survey of multidimensional medians. *International Statistical Review*, 58:263–277, 1990.