Algorithms for Bivariate Medians and a Fermat-Torricelli Problem for Lines

Greg Aloupis [a,*] , Stefan Langerman [a] , Michael Soss [b]
Godfried Toussaint [a]

[a] *School of Computer Science, McGill University, 3480 University Street, rm. 318, Montreal, QC, H3A 2A7, Canada*

[b] *Chemical Computing Group, Inc.*

Abstract

Given a set S of n points in $\mathbb{R}^2$, the *Oja depth* of a point θ is the sum of the areas of all triangles formed by θ and two elements of S. A point in $\mathbb{R}^2$ with minimum depth is an *Oja median*. We show how an Oja median may be computed in $O(n \log^3 n)$ time. In addition, we present an algorithm for computing the *Fermat-Torricelli points of n lines* in $O(n)$ time. These points minimize the sum of weighted distances to the lines. Finally, we propose an algorithm which computes the *simplicial median* of S in $O(n^4)$ time. This median is a point in $\mathbb{R}^2$ which is contained in the most triangles formed by elements of S.

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 $\mathbb{R}^d$ which maximizes a certain depth definition with respect to a data set. Desirable properties for a median include invariance to affine transformations, monotonicity, and a high breakdown point, which measures how much a data set must be altered to move a median towards infinity. Applications include data description, multivariate confidence regions, p-values, quality indices, and control charts [1]. An introduction to the topic including definitions and properties of the most common medians is given

* Corresponding author.
Email addresses: athens@uni.cs.mcgill.ca (Greg Aloupis), sl@cgm.cs.mcgill.ca (Stefan Langerman), soss@chemcomp.com (Michael Soss), godfried@cs.mcgill.ca (Godfried Toussaint).

in [2]. For a recent account on the computational complexity of computing some of these medians, see [3]. For an extensive survey of depth measures used in nonparametric statistics, see [4]. Below, we include descriptions for the Oja median [5] and for the simplicial median of Liu [6].

Let $S = \{s_1, \ldots, s_n\}$ be a set of data points in $\mathbb{R}^d$. The *Oja depth* of a point θ in $\mathbb{R}^d$ 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 $\mathbb{R}^2$, we sum the areas of all triangles formed by θ and two points of S. The Oja median is any point in $\mathbb{R}^d$ with minimum Oja depth.

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

Both medians are invariant to affine transformations. It has been shown [7] that the Oja median may have a small breakdown point for certain data sets, although previously it was suspected that its breakdown point was near optimal. We are not aware of any published results concerning monotonicity or the breakdown point of the simplicial median.

Rousseeuw and Ruts [1] presented an algorithm for computing bivariate simplicial depth in $O(n \log n)$ time. The same technique was discovered independently by Khuller and Mitchell [8] and by Gil, Steiger and Wigderson [9]. A matching lower bound was presented in [10]. Rousseeuw and Ruts mentioned 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. They also briefly described how their methods can be used to speed up the algorithm of Niinimaa, Oja and Nyblom [11] for computing the Oja bivariate median from $O(n^6)$ to $O(n^5 \log n)$ time. This time complexity was improved by Aloupis, Soss and Toussaint [12,3]. They proved that the Oja depth function is convex and that the minimum is on an intersection of lines formed by data points. The minimum depth on a line could be computed in $O(n^2)$ time, and in this time the side containing the median could be determined. This allowed the median to be found with $O(n)$ binary searches, in time $O(n^3 \log n)$.

Roy Barbara [13] referred to the set of points with minimum sum of weighted distances to n given lines as the Fermat-Torricelli points of the n lines. Fermat and Torricelli are known for first solving the problem of finding a point which minimizes the sum of distances to three given points [14–16]. The generalized problem with n points also bears their name, though it is also known as the problem of computing the geometric 1-median. For other generalizations of the Fermat-Torricelli problem, see [17,18].

Barbara proved that a Fermat-Torricelli point of n lines may always be found on one of the intersections of the lines. He then suggested computing the weighted sum for each intersection point, which is straightforward to do in $O(n^3)$ time using $O(n)$ space.

In section 2 we propose an optimal $O(n)$ time algorithm to find a Fermat-Torricelli point of n lines. It is not difficult to see that this also leads to an $O(n^2)$ time algorithm for computing the Oja median of n points. We further reduce this complexity to $O(n \log^3 n)$ in section 3, with certain techniques developed recently by Langerman and Steiger [19]. Finally, in section 4 we present an algorithm for computing the simplicial median in $O(n^4)$ time and $O(n^2)$ space.

2 The Fermat-Torricelli Points of n Lines

As mentioned above, Barbara [13] proved that it suffices to consider the intersections of n lines in order to find their Fermat-Torricelli point. Here, we provide a shorter proof:

Lemma 1 *One of the intersection points of a set L of n lines must be a Fermat-Torricelli point of L.*

PROOF. Each cell formed by the arrangement of the n lines can be considered to be a linear program, where the feasible region is determined by the edges of the cell, and the function to be minimized is the weighted sum of distances to the n lines. Because this is a linear function, if a Fermat-Torricelli point is located within a given cell, one of the vertices of the cell must also be a Fermat-Torricelli point. $\square$

Fact 1 *The sum of two or more linear functions is linear.*

Fact 2 *The sum of two or more piecewise-linear functions is piecewise-linear.*

Fact 3 *The sum of two or more convex functions is convex.*

Let $D_L(p)$ denote the function whose value at a point p in the plane is the sum of weighted distances from p to a set L of n lines. In other words we have

$$D_L(p) = \sum_{\ell \in L} D_\ell(p)$$

where $D_\ell(p)$ is the distance from p to a line ℓ. The minimum value D^* of the function D_L occurs at a Fermat-Torricelli point of L, by definition.

Lemma 2 *The function D_L is piecewise-linear and convex, and the linear pieces correspond to the cells of the arrangement of L.*

PROOF. $D_L(p)$ is a sum of piecewise-linear convex functions, so the proof follows from facts 1–3. □

Lemma 3 *The minimum value of D_L along any line ℓ may be found in $O(n)$ time. In the same time bound, we can determine which side of ℓ contains a Fermat-Torricelli point.*

PROOF. Compute the intersection points of ℓ with all lines of L in $O(n)$ time. Let the weight of any given line in L be $sin(\alpha)$, where α is the acute angle formed between the line and ℓ. Thus for any point on ℓ, D_L becomes a weighted sum of distances to all intersection points on ℓ. In other words, the point p with minimum value of D_L along ℓ is equal to the weighted univariate median of n points, which can be computed in $O(n)$ time. p is a point where ℓ intersects one or more lines of L.

Since D_L is convex, by computing a gradient vector at p in $O(n)$ time, we can determine which side of ℓ contains D^*. This is done as follows: for each line m in L which does not contain p, form a unit vector orthogonal to m and directed away from p. This represents the optimal direction to move from p in order to minimize the distance to m. Let g be the sum of these vectors. If only one line in L contains p, then we project g onto this line. We know that D_L decreases from p in the direction of this projection, and thus we know which side of ℓ contains D^* (if the projection has zero magnitude then p is a Fermat-Torricelli point, by convexity). If, instead, p is on the intersection of two or more lines in L, we then repeat the entire procedure on two new parallel lines which are located symbolically above and below ℓ. The minimum on each of these lines will intersect only one line in L. Therefore one of them will indicate that D^* is located strictly to one side of ℓ, or they will indicate that D^* is on ℓ which means p is a Fermat-Torricelli point. □

Our algorithm for computing a Fermat-Torricelli point of n lines continuously shrinks a region, inside which we know that a Fermat-Torricelli point is located. This is done by carefully selecting certain lines and determining which side of these lines contains the point. While doing so, we implicitly obtain the same information for larger sets of lines in L. Explicit knowledge of the boundaries of this region is not necessary. At the end of the algorithm, we know, for all but a constant number of the lines in L, on which side a Fermat-Torricelli point lies. Thus by checking the intersection points of a constant number of lines, we can find a Fermat-Torricelli point.

Let K be the set of lines in L for which we know on which side a Fermat-Torricelli point lies. Initially, K is empty. Let U be the complement of K. We can compute a partial gradient belonging to the lines in K. This partial gradient may be updated in constant time for each new line introduced to K. Thus we can use only $O(|U|)$ time to determine which side of a line contains a Fermat-Torricelli point, by slightly modifying the procedure of lemma 3. The modification merely involves taking the partial gradient into account while computing the point p on a line. Essentially the lines in K are substituted by a single line, and contribute a single weight.

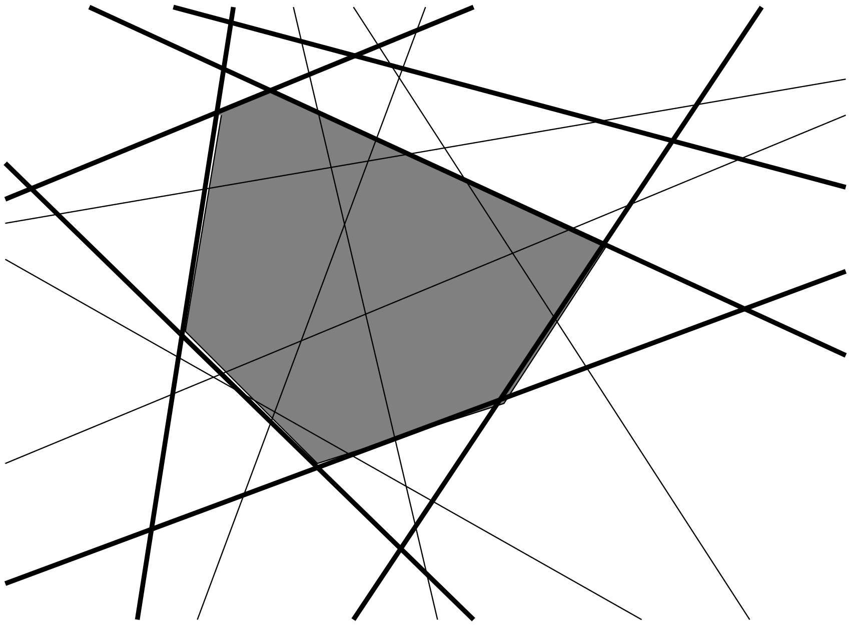

Fig. 1. Lines passing through the shaded region are in U.

In figure 1 the solid lines are in K, so we know that a Fermat-Torricelli point is located within the shaded region. This means that we may express the weighted distance to the solid lines as a gradient vector. This gradient is valid within the entire shaded region. Regardless of where the Fermat-Torricelli point lies, the gradient tells us which is the optimal direction to move in order to minimize the sum of distances to the solid lines. Once a line is placed in K we only need to update the gradient in constant time, never having to worry about this line for the remainder of our algorithm. Note that even though we are continuously shrinking the shaded region, there is no explicit knowledge of its boundaries, and some intersection points of lines in U may be outside the shaded region but still be processed. However, the total gradient vector will only be computed at intersection points within the shaded region, where the partial gradient is valid.

We say that two lines $\{\ell_A, \ell_B\}$ form an *equitable partition* of a set L of n lines in $\mathbb{R}^2$ if each quadrant determined by ℓ_A and ℓ_B is guaranteed not to intersect roughly a quarter of the lines in L. Langerman and Steiger [19] have shown that an equitable partition always exists and may be computed in $O(n)$ time. They do this by considering the dual plane, where the n lines are mapped to n points. After constructing a vertical line v which splits the dual points evenly, they compute a *ham-sandwich cut* h. Thus the lines v and h form quadrants in the dual plane, each containing at least $\frac{n}{4}$ points. If A is the intersection of v and h, and B is a point at infinity on h, then the lines ℓ_A and ℓ_B in the regular plane, which have A and B as duals, form an equitable partition.

Algorithm FT

(1) Compute an equitable partition $\{\ell_A, \ell_B\}$ of U.
(2) Find which quadrant determined by the lines ℓ_A and ℓ_B contains a Fermat-Torricelli point.
(3) Determine which lines in U do not intersect the quadrant.
(4) Add to K the lines found in step 3, and update the gradient vector representing the distance to all lines in K.
(5) Go to step 1 if $|U|$ is still larger than some constant. Otherwise evaluate the function D_L on each intersection point of the lines in U.

Steps 1, 2 and 3 take $O(|U|)$ time in each iteration, and step 4 takes $O(n)$ time in total. Since a constant fraction of the lines is pruned by each iteration, we have the following theorem:

Theorem 4 *Algorithm FT computes a Fermat-Torricelli point of n lines in $O(n)$ time.*

3 The Oja Median

In this section we provide an algorithm for finding the Oja median of n points in $O(n \log^3 n)$ time and $O(n)$ space. Before doing so, we point out that the technique of algorithm *FT* may also be used to find the Oja median in $O(n^2)$ time.

Suppose $S = \{s_1, s_2, \ldots, s_n\}$ is a set of data points in $\mathbb{R}^2$, L is the set of lines formed by all pairs of points in S, and A is the arrangement of L.

Lemma 5 *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 $\overline{s_i s_j}$. □

Corollary 6 *To compute the Oja median of a set of points S it suffices to consider only the vertices of corresponding arrangement A.*

PROOF. The proof follows from lemma 1. □

Corollary 7 *The Oja depth function is convex.*

PROOF. The proof follows from lemma 2. □

Since the Oja depth function can be described as a weighted distance function to $O(n^2)$ lines, we can use algorithm FT to obtain the Oja median in $O(n^2)$ time.

Let $f_S(p) : \mathbb{R}^2 \to \mathbb{R}$ be a function for which:

- The minimum of f_S is on a vertex of A.
- Given a point a, in time $T(n)$ we can find a halfplane H determined by a line through a where $f_S(q) \geq f_S(a)$ for all $q \in H$.

Langerman and Steiger [19] have proved the following theorem:

Theorem 8 *The minimum $f^\star$ of f_S and a vertex x of A with $f_S(x) = f^\star$ may be computed in $O(T(n) \log^2 n + n \log^3 n)$ time.*

We prove that the bivariate Oja depth function, for which the minimum is the Oja median, satisfies the properties of f_S, with $T(n) = O(n \log n)$.

The following lemma concerns computing the gradient of the Oja depth function at a point in $O(n \log n)$ time. Specifically, we describe a simplified version of a technique proposed by Rousseeuw and Ruts [1].

Lemma 9 *The Oja gradient at a point θ may be computed in $O(n \log n)$ time.*

PROOF. Sort the data points $\{s_1, \ldots, s_n\}$ radially in an anti-clockwise order about θ. Let v_i be the vector from θ to s_i. Every pair of data points s_i, s_j contributes a vector to the gradient. The magnitude of this vector is equal to the distance from s_i to s_j and it is directed orthogonally from $\overrightarrow{s_i s_j}$ away from θ. Thus we wish to sum all vectors $\overrightarrow{s_i s_j}$, where s_j is in the open halfspace to the left of v_i. The resulting vector must then be rotated clockwise by an angle of $\frac{\pi}{2}$. In figure 2 the vectors summed for $i = 1$ are shown dashed.

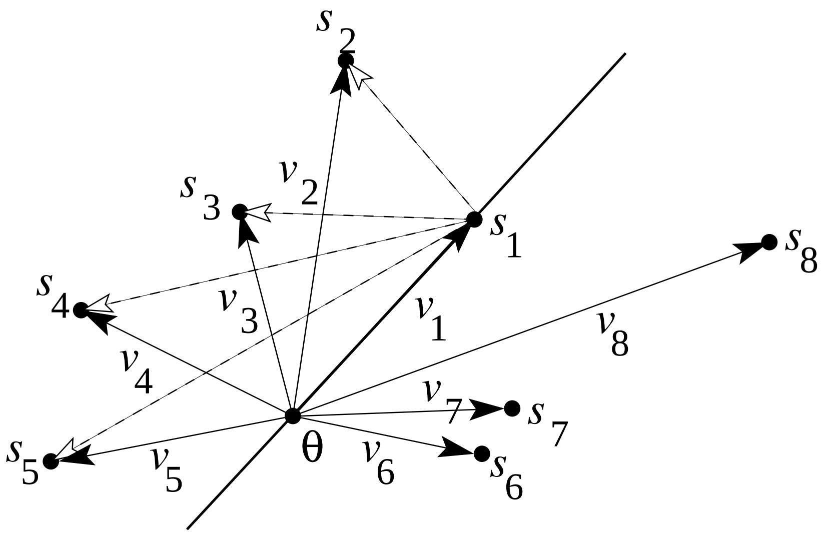

Fig. 2. Oja gradient calculation: the vectors summed for s_1.

The sum of these vectors may be expressed as

$$(v_2 - v_1) + (v_3 - v_1) + (v_4 - v_1) + (v_5 - v_1).$$

In general, if k points are to the left of v_i, the sum of vectors A_i is

$$A_i = -kv_i + \sum_{j=i+1}^{i+k} v_j \tag{1}$$

where j is taken modulo n. A_1 may be computed in $O(n)$ time. Let ℓ be a line through θ and s_1. Rotate ℓ counterclockwise through the sorted list of points. Every time ℓ encounters a point, we add or subtract its vector from the vector sum in equation 1, depending on whether the point is entering or exiting the open halfspace to the left of ℓ. We also update k and increment i. Since each vector will be added and subtracted once, this process takes $O(n)$ time to compute all remaining A_i. Therefore the time complexity of this procedure is dominated by the initial sorting step. $\square$

We can use figure 2 to give an example of the above procedure. Initially we have $A_1 = -4v_1 + V$, where V is the vector sum of v_2,v_3,v_4,v_5. When we sweep a line through the points in a counterclockwise direction, we first encounter s_2. Thus we subtract v_2 from V and compute $A_2 = -3v_2 + V$. When we reach s_4, V will be equal to v_5. After s_4 the rotating line will cross s_6 and s_7, so v_6 and v_7 are added to V. Therefore $A_5 = -2v_5 + (v_6 + v_7)$.

At any point p in the plane, we can compute the Oja gradient in $O(n \log n)$ time. The line orthogonal to the gradient determines a halfplane inside which the Oja depth of any point cannot be less than the depth of p. Thus the Oja depth function satisfies both conditions required by theorem 8, and we obtain the following theorem:

Theorem 10 *The bivariate Oja median may be computed in $O(n \log^3 n)$ time.*

Lemma 11 *The points in $\mathbb{R}^2$ with minimum Oja depth form a convex polygon with $O(n)$ vertices on its boundary.*

PROOF. Since the graph of the Oja depth function is a convex polyhedron, the minimum value must be at a vertex, a line segment or a face of the polyhedron. If we project these objects back onto the plane, we obtain an intersection point, a boundary segment on a cell, or a single cell in the arrangement of L. 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 $O(n)$. □

4 The Simplicial Median

Let S be a data set of n points in $\mathbb{R}^2$, and I be the set of line *segments* formed between every pair of points in S. In this section we provide algorithms for computing the simplicial median of n points. Algorithm *Simp-Med* uses $O(n^4 \log n)$ time and $O(n^2)$ space, or alternatively $O(n^4)$ time and space. This algorithm essentially processes each segment in I, scanning its sorted intersection points with other segments in I. If we perform a topological sweep instead of processing each segment separately, we can reduce the complexity of computing the simplicial median to $O(n^4)$ time and $O(n^2)$ space.

Lemma 12 *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 that of points in the interior. Finally, any vertex of the cell must have depth at least equal to that of any point on an adjacent boundary. These vertices are the intersection points of segments in I. □

To simplify the description of algorithm *Simp-Med* for computing the simplicial median, assume that S is in general position, in the sense that no three points are collinear. Later we explain how this assumption may be removed, if desired, without influencing the time or space complexity.

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 one of the endpoints. 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.

Figure 3 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 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 figure 3, suppose we are moving from left to right along the current line segment, 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, $d = 8$.

In case we have multiple segments intersecting the current line at the same point, we may form a temporary list of these segments. When the last one is encountered, we process the list: First we add the points to the right of each line, then we subtract all the points to the left of each line.

Theorem 13 *Algorithm* Simp-Med *computes the bivariate simplicial median*

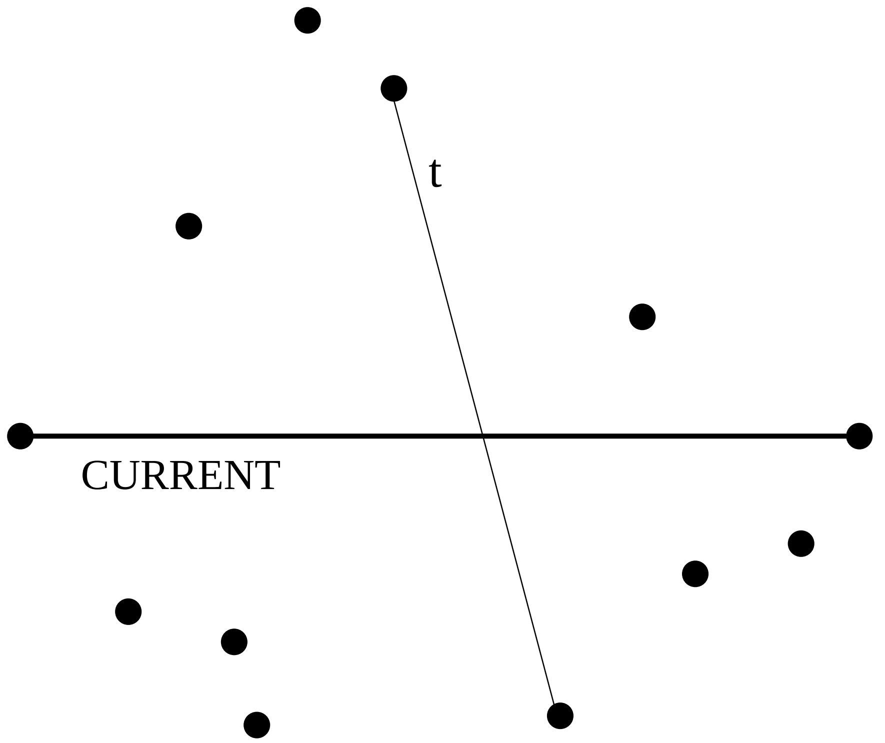

Fig. 3. Algorithm *Simp–Med*: the current line segment traversed, and an intersecting segment t.

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 12 a point of maximum simplicial depth is found. Step 1 takes $O(n^2 \log n)$ time in total since the calculation of simplicial depth for a point takes $O(n \log n)$ time [1,8,9]. Step 2 is straightforward to do in $O(n^3)$ time by brute force. Step 3a takes $O(n^2 \log n)$, 3b is constant, 3c is $O(n \log n)$, and 3d takes $O(n^2)$ time ($O(1)$ per intersection point). 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 steps 2 and 3a. If we compute the arrangement of I in $O(n^4)$ time and space as preprocessing, then step 3a is no longer necessary. In this case, the preprocessing step would dominate the complexity of the algorithm. □

Instead of performing step 3 in algorithm *Simp-Med*, we can perform a topological sweep, which takes $O(n^2)$ time and $O(n)$ space for n lines [20]. Thus instead of processing each segment sequentially, we process all $O(n^2)$ segments in parallel. Every time the sweep curve encounters an intersection point on some segment, we process the point in the same way as step 3d. Thus we have the following theorem:

Theorem 14 *The simplicial median of n points may be computed in $O(n^4)$ time and $O(n^2)$ space.*

By taking advantage of the structure of the lines in this problem, we can reduce the time complexity of certain steps in the above algorithm, although the overall complexity is only improved by a constant factor. For example, in step 2 we can compute the *halfspace depth* of every data point, but store the points counted for every halfplane considered. We remind the reader that the halfspace depth of a point p with respect to a set of n points is the minimum number of points contained in any open halfplane determined by a line through p. Halfspace depth may be computed in $O(n \log n)$ time with a simple algorithm by Rousseeuw and Ruts [1] (for a matching lower bound, see [21,10]). The algorithm takes $O(n)$ time after sorting the points radially about p. It is possible to sort all points about each point in $O(n^2)$ time [22,23], so the halfspace depth procedure then takes only $O(n)$ time per data point. This means that the time complexity of step 2 reduces to $O(n^2)$. The same applies for computing the simplicial depth of all data points in step 1.

Finally we explain how the general position assumption may be removed. Steps 1 and 2 are not affected. While performing step $3a$ for some line ℓ, if we encounter an intersection point which happens to be a data point, we can simply stop processing ℓ and go to the next segment. We may do this because the intersecting data point will form segments with both endpoints of ℓ, so all points on ℓ will be processed anyway. This leaves one more case to consider. Suppose that if we extend an intersecting segment t we will find c collinear points. We can determine this easily since we know the number of points strictly to the side of t. It follows that we must also have more intersecting segments collinear with t. If there are m such segments, then we claim that on this intersection point we are inside $\frac{m(c-2)}{2}$ triangles, apart from those considered by the regular algorithm. We arrive at this number from the following argument: With the endpoints of an intersecting segment and one of the other c points, we can form a triangle containing the intersection point. Therefore using the two endpoints we can form $c-2$ such triangles. Suppose that x is the third point for one of these triangles. Then x and one of the endpoints will form another intersecting segment. So the same triangle will be counted exactly one more time when the other segment is processed. Therefore we have $c-2$ triangles counted for each of the m segments, and we multiply by a factor of $\frac{1}{2}$ to avoid counting each triangle twice.

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. We also thank three anonymous referees for their valuable comments.

References

[1] P. Rousseeuw, I. Ruts, Bivariate location depth, Applied Statistics 45 (1996) 516–526.

[2] C. Small, A survey of multidimensional medians, International Statistical Review 58 (1990) 263–277.

[3] G. Aloupis, On computing geometric estimators of location, M.Sc. thesis, McGill University (2001).

[4] R. Liu, J. Parelius, K. Singh, Multivariate analysis of data depth: descriptive statistics, graphics and inference, Annals of Statistics 27 (3) (1999) 783–858.

[5] H. Oja, Descriptive statistics for multivariate distributions, Statistics and Probability Letters 1 (1983) 327–332.

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

[7] A. Niinimaa, H. Oja, M. Tableman, The finite-sample breakdown point of the Oja bivariate median and of the corresponding half-samples version, Statistics and Probability Letters 10 (1990) 325–328.

[8] S. Khuller, J. Mitchell, On a triangle counting problem, Information Processing Letters 33 (1989) 319–321.

[9] J. Gill, W. Steiger, A. Wigderson, Geometric medians, Discrete Mathematics 108 (1992) 37–51.

[10] G. Aloupis, C. Cortes, F. Gomez, M. Soss, G. Toussaint, Lower bounds for computing statistical depth, Technical Report SOCS-01.1, School of Computer Science, McGill University, (to appear in: Computational Statistics and Data Analysis) (February 2001).

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

[12] G. Aloupis, M. Soss, G. Toussaint, On the computation of the bivariate median and a Fermat-Torricelli problem, Technical Report SOCS-01.2, School of Computer Science, McGill University (February 2001).

[13] R. Barbara, The Fermat-Torricelli points of n lines, Mathematical Gazette 84 (2000) 24–29.

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

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

[16] G. Loria, G. Vassura (Eds.), Opere di Evangelista Torricelli, Vol. 1, Faenza, 1919.

[17] L. Dalla, A note on the Fermat-Torricelli point of a d-simplex, J. Geom. 70 (2001) 38–43.

[18] Y. Kupitz, H. Martini, Geometric aspects of the generalized Fermat-Torricelli problem, Intuitive Geometry, Bolyai Society Math Studies 6 (1997) 55–127.

[19] S. Langerman, W. Steiger, Optimization in arrangements, Technical Report SOCS-02.7, School of Computer Science, McGill University (2002).

[20] H. Edelsbrunner, L. Guibas, Topologically sweeping an arrangement, Journal of Computer and System Sciences 38 (1989) 165–194.

[21] S. Langerman, W. Steiger, Computing a maximal depth point in the plane, in: Japan Conference on Discrete and Computational Geometry, November 2000.

[22] D. Lee, Y. Ching, The power of geometric duality revisited, Information Processing Letters 21 (1985) 117–122.

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