Computing the center of planar point sets

JIŘÍ MATOUŠEK
Department of Applied Mathematics, Charles University
Malostranské nám. 25, 118 00 Praha 1, Czechoslovakia
and
School of Mathematics, Georgia Institute of Technology
Atlanta, GA 30 332 USA

July 6, 2000

Abstract

Given a collection H of n lines in the plane, the *level* of a point x is the number of lines of H lying below x or passing thru x. We show that for a given k, one can compute the convex hull of the set of points of level $\leq k$, in time $O(n \log^4 n)$. This implies that a description of the set of *centerpoints* of a given n-point set in the plane can be found within the same time bound, and a point of greatest *Tukey depth* (*Tukey median*) for a n-point set can be computed in time $O(n \log^5 n)$. We also mention the computation of an approximate centerpoint in higher dimension.

1 Introduction

Given a n-point set P in E^d and a point $x \in \mathrm{E}^d$, one defines the *Tukey depth* (or depth for short) of x as the minimum number of points of P contained in a closed halfspace whose bounding hyperplane passes thru x. The Helly theorem implies that for any P, there always exists a point x of depth at least $\lceil \frac{n}{d+1} \rceil$ (not necessarily belonging to P); such a point is called a *centerpoint* of P, and the set of all centerpoints is called the *center* of P. Since the condition that x is a centerpoint of P can be formulated as the containment of x in all halfspaces of a certain set (those containing at least $\lceil \frac{n}{d+1} \rceil$ points of P), the center is closed and convex.

A point of P of the largest depth is called a *Tukey median* of P. The Tukey median is one of many possible generalizations of the notion of median for a set of real numbers, and it has applications in statistics, see e.g. [GSW90].

An efficient algorithm computing a centerpoint in the plane was given by Cole et al. [CSY87]. This algorithm has complexity $O(n \log^5 n)$, and with the technique of Cole [Col87] this can be improved to $O(n \log^3 n)$ (this improved result, however, probably has not appeared anywhere in a written form). The algorithm has been extended to dimension 3 by Naor and Sharir [NS90], with complexity $O(n^2 \log^6 n)$. It is straightforward to determine the depth of all points of a given set in time $O(n^d)$ (by constructing the dual arrangement of hyperplanes), and thus the Tukey median can be determined within the same time bound.

In this paper we consider an efficient computation of the whole center and of the Tukey median in the plane, and we show the following:

Theorem 1.1 *Given a n-point set P in the plane and a parameter k, the description of the set of points of depth at least k can be computed in time $O(n \log^4 n)$. The Tukey median of P can be found in time $O(n \log^5 n)$.*

We will reduce the problem to finding of the convex hull of all points of level at most k in an arrangement of n lines, whose computation may be a result of independent interest.

2 A reduction: Level convex hull problem

Let H be a collection of n nonvertical lines in the plane. For a point x, we define its *level* as the number of lines of H lying strictly below x or passing thru x. We will denote the closure of the set of all points of level at most k by $L_k(H)$ or simply L_k where H is understood. The boundary of L_k will be denoted by λ_k and called the *level* k (of H). It is not difficult to see that λ_k is a x-monotone piecewise linear curve, whose vertices are pairwise intersections of the lines of H and whose segments are parts of the lines of H. If the lines of H are in general position, then all points of λ_k have level $k+1$ or $k+2$.

We will also use the notation U_k for the closure of the set of points of level at least k.

These notions are related to the concepts in the introduction via a duality transform (this transform sends a point with coordinates (a, b) to a line with equation $y = 2ax - b$ and vice versa, see e.g. [Ede87]). Namely, a point x has depth at most k relative to a point set P, iff all points of the line dual to x have level at least k and at most $n-k$, relative to the collection of lines dual to P. We thus say that a line ℓ has *depth* at least k if it intersects neither the interior of L_k nor the interior of U_{n-k}. A line of depth at least k is a dual notion to a point of depth at least k.

We observe that a line intersects the interior of L_k iff it intersects the interior of its convex hull. Thus the property "having depth $\geq k$" for a line can be decided with a knowledge of $\mathrm{conv}(L_k)$ and $\mathrm{conv}(U_{n-k})$. In the sequel, γ_k (or $\gamma_k(H)$) will denote the boundary of $\mathrm{conv}(L_k)$.

An obvious way to compute γ_k is to compute all vertices of the boundary λ_k of L_k, and use some convex hull algorithm. As shown by Edelsbrunner and Welzl [EW86], a complete description of λ_k can be computed in time $O(n \log n + |\lambda_k| \log^2 n)$, where $|\lambda_k|$ denotes the number of segments forming λ_k (this has been improved to $O(n \log n + |\lambda_k| \log^2 k)$ by Cole et al.[CSY87]). However, the worst-case complexity of λ_k is essentially unknown; the current best estimates are $\Omega(n \log k)$ from below and $O(n\sqrt{k}/\log^* k)$ from above (the upper bound is due to Pach et al. [PSS90]; see [Ede87] for older references) and tightening these bounds is one of major open problems in combinatorial geometry.

On the other hand, we have the following easy observation:

Lemma 2.1 *For any k, γ_k has at most n vertices.*

Proof: Indeed, any vertex of γ_k is a vertex of λ_k, which in turn is an intersection of two lines of H. But every line may pass through at most two vertices of the convex chain γ_k, and two lines are needed at every vertex, thus the number of vertices is at most n. $\square$

The main result of this paper is an algorithm computing γ_k efficiently, without computing a complete description of λ_k:

Theorem 2.2 *For a collection of n lines in the plane and a parameter k, the convex hull $\mathrm{conv}(L_k)$ can be computed in time $O(n \log^4 n)$.*

The computation of $\mathrm{conv}(U_{n-k})$ is completely symmetric. Let us indicate how this implies Theorem 1.1. If we know the convex hulls of L_k and of U_{n-k}, we can determine whether a given line intersects their interior, using an algorithm determining the position of a line relative to a given convex polygon (with running time $O(\log n)$ after a simple preprocessing of the polygon; see e.g. [PS85]). Hence once these convex hulls are computed, we can find all lines of H of depth at least k in additional $O(n \log n)$ time. The deepest line (dual to the Tukey median) is then found by a binary search on k, in $O(\log n)$ stages.

The set of all nonvertical lines of depth $\geq k$ can be described as the lines lying below all vertices of $\mathrm{conv}(U_{n-k})$ and above all vertices of $\mathrm{conv}(L_k)$. This gives the set of points of depth $\geq k$ in the primal plane as an intersection of $O(n)$ halfplanes.

3 Computing tangents to levels

In this section we describe several subroutines used in the main algorithm, which will be given in the next section. The material of this section (in a dual setting) is essentially contained in [CSY87]. For reader's convenience, we briefly explain these parts of our algorithm. The results this section are based on various versions of Megiddo's parametric search technique, and although we outline the necessary material, a familiarity with that technique might be helpful (see e.g. the original paper [Meg83] for a nice exposition).

For the sake of simplicity, we will assume that the given lines are in general position. This assumption could be removed either by so-called *simulation of simplicity* (see e.g. [Ede87]), or by a careful analysis of possible degenerate cases in our algorithm.

Lemma 3.1 *Given a collection H of n lines and a line q, one can find all vertices of λ_k lying on q in time $O(n \log n)$.*

Proof: We sort the intersections of the lines of H with q by increasing x-coordinate, which is also in the order of their appearance along q. The level of the leftmost intersection (or the bottommost one for a vertical q) is determined directly, and we then go along q and update the current level (by ± 1) appropriately at every intersection. $\square$

Lemma 3.2 *Given a collection H of n lines and a point x, one can find the tangent to γ_k passing thru x and touching γ_k to the right of x (if it exists) in time $O(n \log^2 n)$.*

Proof: We will actually compute a line τ^* such that its right semiline from x touches L_k (and does not cut it). In the end, we check whether the left semiline cuts L_k; if not, then τ^* is the answer, otherwise we infer that x lines below γ_k and no tangent exists.

We begin by checking whether the level of x is at least k (otherwise no tangent exists). Then there comes our first application of the parametric search technique. We will search for the slope z^* of the desired tangent $\tau^* = \tau(z^*)$. To this end, we will run an algorithm sorting the intersections of τ^* with the lines of H, in left-to-right order along τ^* (to have a unique order, we formally consider the order of intersections along a line arising by lifting τ^* upwards by a sufficiently small amount). We don't know what τ^* actually is, but nevertheless for two given lines $\ell, \ell' \in H$ we can decide the order of their intersections along τ^* (we will refer to this as *answering the question* (ℓ, ℓ')): it suffices to know whether the intersection $y = \ell \cap \ell'$ lies above or below τ^*. This in turn can be decided by checking whether the line xy intersects L_k to the right of x, and thus a question (ℓ, ℓ') can be answered in time $O(n \log n)$ by the previous lemma.

With Megiddo's technique, one answers batches of questions, taking advantage of the batching; to this end, an efficient parallel algorithm is used for the sorting. In our case, suppose that we are given $m \leq n/2$ questions $(\ell_1, \ell_1'), \ldots, (\ell_m, \ell_m')$ $(\ell_i, \ell_i' \in H)$. For every i, we compute the intersection $y_i = \ell_i \cap \ell_i'$ and the slope z_i of the line $\tau_i = xy_i$. We then find the median z of $z_1, \ldots, z_m$ (i.e. half of the lines of $\tau_1, \ldots, \tau_m$ have slope larger than z and half have slope smaller than z), and we find out whether the line $\tau(z)$ passing thru x and with slope z intersects L_k to the right of x. Based on this, we can answer half of the questions $(\ell_1, \ell_1'), \ldots, (\ell_m, \ell_m')$ in $O(n \log n)$ time. In order to apply an extension to Megiddo's technique due to Cole [Col87] (improving the efficiency by a logarithmic factor), we require the following technical modification: We are given m questions together with some nonnegative real weights $w_1, \ldots, w_m$ (summing up to 1, say), and we want to answer questions of total weight at least 1/2; let us call this a weighted questions batch problem. The above outlined method can be easily adapted to this, simply by taking the slope z to be a weighted median of the slopes $z_1, \ldots, z_m$ with weights $w_1, \ldots, w_m$ respectively. Cole's results show that the whole sorting of intersections along the unknown τ^* can be done by solving $O(\log n)$ weighted questions batch problems, giving a total time $O(n \log^2 n)$ in our case. It turns out that when the sorting is finished, the unknown τ^* must have been considered as one of the lines $\tau(z)$ used in solving the weighted questions batch problems, and thus we know τ^* explicitly. $\Box$

Let us remark that a special case of the previous lemma (with x in the infinity) allows us to compute a tangent to γ_k with a prescribed slope.

Lemma 3.3 *Given a collection H of n lines and a vertical line v, one can compute a tangent τ^* touching γ_k at its intersection with v, in time $O(n \log^3 n)$.*

Proof: Among all tangents τ to γ_k, the desired τ^* minimizes the y-coordinate of the intersection with v. We will find such τ^* in two steps.

First we determine the order relation of the slope of τ^* to the slopes of all the lines of H. We will do a binary search locating the slope of τ^*, and all we need is an algorithm deciding the relation of the slope of τ^* to the slope of a given line q. Using Lemma 3.2, we find a tangent τ' to γ_k parallel to q. If τ' touches γ_k strictly to the left of v, we can push the intersection of τ' with v down by slightly lowering the right end of τ', and hence the slope of τ^* is smaller than the slope of q. We proceed similarly for τ' touching γ_k to the right of v. Thus the whole binary search can be performed in time $O(n \log^3 n)$.

The second step will already find τ^*, by a parametric search. We will again be sorting the intersections along τ^*. Since we know the relative slope of τ^* with respect to all lines of H, a question (ℓ, ℓ') (about the order of intersections of ℓ and ℓ' along τ^*) can be decided once we know whether the intersection $x = \ell \cap \ell'$ lies above or below τ^*. Suppose that x lies e.g. to the left of v. We compute the tangent τ to γ_k passing thru x and touching γ_k to the right of x (if such a tangent does not exist then x is below γ_k and hence also below τ^*). Now it is easy to verify that if the touchpoint of τ is to the right of v, then x lies above τ^*, and otherwise it lies below τ^* (or on τ^* if there are touchpoints on both sides of v or at v).

We are now aiming at solving the weighted questions batch problem (see the proof of Lemma 3.2). Given a weighted collection $(\ell_1, \ell_1'), \ldots, (\ell_m, \ell_m')$ of questions, we compute the intersections $x_i = \ell_i \cap \ell_i'$, whose position relative to τ^* should be decided. We then pass to the dual plane, where we get the following situation: We have a weighted collection Q of lines (those dual to the points $x_1, \ldots, x_m$), and an unknown point t^* (the one dual to τ^*), and for any given line r, we can decide the position of t^* relative to r (in time $O(n \log^2 n)$).

It is known that one can construct a constant number of lines $r_1, \ldots, r_c$ such that by deciding the position of t^* with respect to these lines, the position of t^* with respect to at least half of the lines of Q (in terms of weight) is determined. The lines $r_1, \ldots, r_c$ can be constructed in linear time. A construction of this type was first given by Megiddo (see the book [Ede87]). Other type of such constructions originated in the theory of random sampling, see e.g. [HW87], [Mat90]. Using such a construction, we can solve the weighted questions batch problem in $O(n \log^2 n)$ time, and the sorting of intersections along τ^* can be thus performed in $O(n \log^3 n)$ time. Again one can show that when this algorithm finishes, the desired tangent τ^* must have been considered explicitly in answering the questions. $\square$

4 Main algorithm

In this section we prove Theorem 2.2. We begin by some notation and simple observations.

For a point a in the plane, let $x(a)$ denote its x-coordinate. For a number x_0, let $v(x_0)$ stand for the vertical line $x = x_0$, and we write just $v(a)$ for $v(x(a))$.

We assume that H is a collection of n lines in general position. Let a, b be two points, $x(a) < x(b)$. We denote by $\gamma_k(H, a, b)$ the portion of $\gamma_k(H)$ lying between $v(a)$ and $v(b)$. The number of intersections of the lines of H with x-coordinate in an interval (x_0, x_1) will be denoted by $N(H, x_0, x_1)$. We will make use of the following elementary lemma:

Lemma 4.1 [Mat90] *Suppose that a, b have the same level in the arrangement of H. Then the open segment ab is intersected by at most $2\lfloor\sqrt{N(H, x(a), x(b))}\rfloor$ lines of H.* $\Box$

Let us now consider the situation with H, a, b as above. Put $N = N(H, x(a), x(b))$ and let us moreover suppose that a and b are vertices of $\gamma_k(H)$ (thus $a, b \in \lambda_k$, and a, b have the same level). Let τ_a, τ_b be tangents to $\gamma_k(H)$ touching at a, b respectively (for definiteness, we assume that τ_a has the smallest possible slope among all tangents at a and τ_b the largest possible slope among all tangents at b). We introduce a classification of the lines of H based on their relation to a, b and related objects. First we distinguish the position of a line ℓ with respect to the open segment ab (below, above, crossing). For the lines above ab, we introduce two finer classifications. We say that a line ℓ above ab is *low above ab*, if at least one of the points $\ell \cap v(a)$, $\ell \cap v(b)$ has level at most $k + 1 + \lfloor\sqrt{N}\rfloor$; otherwise ℓ is *high above ab*. We say that a line ℓ above ab is *flat above ab* if the slope of ℓ is between the slopes of τ_b and τ_a, otherwise ℓ is *steep above ab*. Finally we call ℓ *redundant* (with respect to a, b) if it is below ab or steep above ab or high above ab; otherwise ℓ is *nonredundant*. The following lemma says just that redundant lines are really redundant:

Lemma 4.2 *Let H' be the set of the nonredundant lines in H, and let k' be k minus the number of lines below ab. Then $\gamma_k(H, a, b) = \gamma_{k'}(H', a, b)$.*

Proof: We proceed by induction, showing that by removing one redundant line and adjusting k appropriately, the portion of the convex hull in $x(a), x(b)$ does not change. It is enough to show that each redundant line is completely above $\gamma_k(H, a, b)$ or completely below it. For the lines below ab it is clear, as well as for the lines steep above ab. Let ℓ be a line high above ab. Let $a' \in v(a)$, $b' \in v(b)$ be points of level $k + \lfloor\sqrt{N}\rfloor + 2$. Since ℓ is high above ab, the segment $a'b'$ lies below the line ℓ. At the same time, no point of the segment $a'b'$ has level $k + 2$ or smaller (otherwise the segment $a'b'$ would be intersected by more than $2\lfloor\sqrt{N}\rfloor$ lines, contradicting Lemma 4.1), and thus all vertices of $\gamma_k(H, a, b)$ lie below $a'b'$. Therefore the whole convex arc $\gamma_k(H, a, b)$ lies below $a'b'$ and also below ℓ. $\Box$

We need one more algorithmic lemma.

Lemma 4.3 *Let c be a given constant, H be a collection of n lines and a, b given points, $x(a) < x(b)$. One can find x-coordinates $x_0 = x(a) < x_1 < \cdots < x_{c-1} < x_c = x(b)$ such that $N(H, x_{i-1}, x_i) \le n^2/c$, $i = 1, 2, \ldots, c$, in time $O(n)$.*

This result is implied by [Mat90], and with the time bound $O(n \log n)$ instead of $O(n)$ it immediately follows from the results of Cole et al. [CS$^+$89]. $\Box$

Let us now describe the algorithm which will be used for computing $\gamma_k(H)$. It will be a recursive divide-and-conquer type algorithm. Its input will be H, k

and vertices a, b of γ_k, $x(a) < x(b)$. It will output (a suitable representation of) $\gamma_k(H, a, b)$. The main algorithm computing the whole γ_k is then obtained as follows: We compute the left and right unbounded rays of λ_k, we find tangents τ_l and τ_r to γ_k parallel to these rays and we use the vertices a, b where τ_l, τ_r touch γ_k in the initial call to the recursive algorithm.

In the recursive algorithm, we proceed as follows:

1. If the size of H is smaller than a suitable constant, we compute the desired convex arc $\gamma_k(H, a, b)$ by some straightforward method. Otherwise we pass to the next step.

2. We use Lemma 4.3 with $c = 64$, and we find numbers $x_0 = x(a) < x_1 < \cdots < x_{c-1} < x_c = x(b)$ such that $N(H, x_{i-1}, x_i) \leq n^2/64$.

3. For every $i = 1, 2, \ldots, c-1$, we apply Lemma 3.3 to compute a tangent τ_i touching γ_k at its intersection with $v(x_i)$. We let b_{i-1} be the leftmost vertex in which τ_i touches γ_k and a_i the rightmost such vertex. We also set $a_0 = a$ and $b_c = b$.

4. For every $i = 1, 2, \ldots, c$, we let H_i be the set of the lines of H which are nonredundant with respect to a_i, b_i (we, however, eliminate the pairs a_i, b_i with $x(a_i) \geq x(b_i)$). We let k_i be k minus the number of lines which are below $a_i b_i$.

5. We call the algorithm recursively on H_i, k_i, a_i, b_i for every i, obtaining $\gamma_{k_i}(H_i, a_i, b_i)$, which by Lemma 4.2 is equal to $\gamma_k(H, a_i, b_i)$.

6. We merge the above obtained convex arcs $\gamma_k(H, a_i, b_i)$ together with the segments $b_{i-1} a_i$, obtaining the desired convex arc $\gamma_k(H, a, b)$.

Let us analyze the running time of this algorithm. Excluding the time spent on the recursive calls, the running time of the algorithm with $|H| = n$ is $O(n \log^3 n)$, by our previous results. Let us estimate the size of the H_i's. Each nonredundant line is either crossing $a_i b_i$ or low above $a_i b_i$. The number of lines crossing $a_i b_i$ does not exceed $2\lfloor \sqrt{N(H, a_i, b_i)} \rfloor \leq 2\lfloor \sqrt{N(H, x_i, x_{i+1})} \rfloor \leq n/4$, by Lemma 4.1. The number of lines low above $a_i b_i$ is at most $2\lfloor \sqrt{N(H, a_i, b_i)} \rfloor \leq n/4$, so $|H_i| \leq n/2$. Hence the depth of recursive calls in our algorithm is $O(\log n)$.

Let us now view the computation as a rooted tree, with nodes corresponding to the recursive calls of the algorithm. If a node corresponds to a call with some a, b, then all lines entering that computation are nonredundant with respect to a, b, in particular they are all crossing ab or flat above ab. If we look at the points occurring as a, b at one level of the recursion tree, we see that they form a convex chain. Hence any given line occurs in at most two nodes at any given level of the recursion tree (either it is flat above in one node, or crossing in two nodes). Therefore the total computation time for every level of the recursion tree is $O(n \log^3 n)$, and the total running time is $O(n \log^4 n)$ as claimed. This finishes the proof of Theorem 2.2. $\square$

Let us conclude this section by a remark about the implementability of our algorithm. As described, the algorithm includes quite heavy algorithmic tools (e.g., the AKS sorting network, hidden in the application of Megiddo's parametric search technique for sorting with Cole's improvement, and also in Lemma 4.3). However, if we permit randomization, this can be simplified considerably, essentially by replacing the parametric search subroutines by "randomized interval halving" (this idea is explained in [Mat91b]). The expected running time of the resulting randomized algorithm can be made no worse than the running time of the deterministic algorithm.

5 Approximate centerpoints

An obvious problem is the computation of a centerpoints, of the center and of the Tukey median for point sets in higher dimensions. Even if the methods used in the planar case could be extended to higher dimensions (in the spirit of [NS90]), one would obtain algorithm of complexity close to n^{d-1} at best. This does not look very satisfactory if all we want is a single point in the center, but achieving a more substantial improvement for this problem in higher dimensions probably requires quite a new insight.

Here we show that if we do not insist on exact determining of centerpoints or of Tukey median, but rather we admit some tolerance in the depth, we can get an efficient solution in any fixed dimension. Let us say that a point $x \in \mathrm{E}^d$ is an ε-approximate centerpoint for P, if it has depth at least $(1-\varepsilon)\frac{n}{d+1}$. Then for any fixed $\varepsilon > 0$, and ε-approximate centerpoint can be found in $O(n)$ time.

To this end, we use so-called ε-approximations. For our problem, we need the following definition: A point set $A \subseteq P$ is called an *ε-approximation for P (with respect to halfspaces)*, if for every halfspace h it is

$$\left| \frac{|A \cap h|}{|A|} - \frac{|P \cap h|}{|P|} \right| < \varepsilon.$$

Obviously if A is an ε-approximation for P and x is a centerpoint of A, then x is an ε-approximate centerpoint for P as well. Now for any $P \subseteq \mathrm{E}^d$ (where the dimension d is fixed) and for every $\varepsilon > 0$, there exists an ε-approximation of size $O(\varepsilon^{-2}\log(1/\varepsilon))$ (see [VC71], [HW87]) and for any *fixed* $\varepsilon > 0$, such an ε-approximation can be computed in time $O(n)$ (with the constant strongly depending on d and ε) – see [Mat91a]. For the constant-sized ε-approximation, we can then use a "brute-force" algorithm for finding a centerpoint; an $O(n^d)$ algorithm is straightforward. A problem with this approach is a large constant of proportionality hidden in the big-Oh notation.

References

[Col87] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34:200–208, 1987.

[CS+89] R. Cole, J. Salowe, , W. Steiger, and E. Szemerédi. An optimal-time algorithm for slope selection. *SIAM Journal on Computing*, 18(4):792–810, 1989.

[CSY87] R. Cole, M. Sharir, and C. Yap. On k-hulls and related problems. *SIAM Journal on Computing*, 16(1):61–67, 1987.

[Ede87] H. Edelsbrunner. *Algorithms in combinatorial geometry.* Springer, 1987.

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

[GSW90] J. Gil, W. Steiger, and A. Widgerson. Geometric medians. Manuscript (1990); to appear in a volume dedicated to Z. Frolík (J. Nešetřil ed.).

[HW87] D. Haussler and E. Welzl. ε-nets and simplex range queries. *Discrete & Computational Geometry*, 2:127–151, 1987.

[Mat90] J. Matoušek. Construction of ε-nets. *Discrete & Computational Geometry*, 5:427–448, 1990.

[Mat91a] J. Matoušek. Approximations and optimal geometric divide-and-conquer. In *Proc. 23. ACM Symposium on Theory of Computing*, pages 506–511, 1991.

[Mat91b] J. Matoušek. Randomized optimal algorithm for slope selection. *Information Processing Letters*, 1991, to appear.

[Meg83] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30:852–865, 1983.

[NS90] N. Naor and M. Sharir. Computing a point in the center of a point set in three dimensions. In *Proc. 2. Canadian Conference on Computational Geometry*, 1990.

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

[PSS90] J. Pach, W. Steiger, and E. Szemerédi. An upper bound on the number of planar k-sets. In *Proc. 30. IEEE Symposium on Theory of Computer Science*, pages 72–79, 1990.

[VC71] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.