# Staying in the Middle: Exact and Approximate Medians in $\mathbb{R}^1$ and $\mathbb{R}^2$ for Moving Points *

Pankaj K. Agarwal† Mark de Berg‡ Jie Gao§ Leonidas J. Guibas¶
Sariel Har-Peled‖

June 2, 2005

**Abstract**

Many divide-and-conquer based geometric algorithms and order-statistics problems ask for a point that lies "in the middle" of a given point set. We study several fundamental problems of this type for moving points in one and two dimensions. In particular, we show how to kinetically maintain the median of a set of $n$ points moving on the real line, and a center point of a set of $n$ points moving in the plane, that is, a point such that any line through it has at most $2n/3$ on either side of it. Since the maintenance of exact medians and center points can be quite expensive, we also show how to maintain $\varepsilon$-approximate medians and center points and argue that the latter can be made to be much more stable under motion. These results are based on a new algorithm to maintain an $\varepsilon$-approximation of a range space under insertions and deletions, which is of independent interest. All our approximation algorithms run in near-linear time.

## 1 Introduction

In this paper we study the problem of "staying in the middle": we have a set of points moving in a geometric space and wish to maintain another point (possibly one of the given points, but not necessarily) that stays continuously "in the middle" of the moving set. Our motivation for studying this problem stems from a number of different applications. First, many data structures for efficiently querying a point set are based on balanced partitions of the set using medians or other equi-partitioning constructs — for example, many well-known range-searching data structures and graph-separator based algorithms utilize such balanced partitioning [2, 26]. When a point set is in motion, however, these partitions need to be updated, and it becomes important to avoid the cost of recomputing them from scratch at each time step.

*P.A. was partially supported by NSF under grants CCR-00-86013 EIA-98-70724, EIA-99-72879, EIA-01-31905, and CCR-02-04118, and by a grant from the U.S.-Israeli Binational Science Foundation. Work by L.G. and J.G. was supported by NSF grant CCR-9910633 and grants from the Stanford Networking Research Center, the Okawa Foundation, and the Honda Corporation. S. H.-P. was supported by NSF CAREER award CCR-0132901.

†Department of Computer Science, Duke University, Durham, NC 27708, USA. E-mail: pankaj@cs.duke.edu

‡Institute of Information and Computing Sciences, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands. markdb@cs.uu.nl

§Center for the Mathematics of Information, California Institute of Technology, Pasadena, CA 91125, USA. Work was done when the author was with Department of Computer Science, Stanford University. E-mail: jgao@ist.caltech.edu

¶Department of Computer Science, Stanford University, Stanford, CA 94305, USA. E-mail: guibas@cs.stanford.edu

‖Department of Computer Science, University of Illinois, Urbana, IL 61801, USA. Email: sariel@uiuc.edu

We wish to maintain the median, or more generally a point of rank $k$, in a set of points moving in $\mathbb{R}^1$. It is well-known that the notion of median and rank order can be extended to higher dimensions, as follows. Let $P$ be a point set in $\mathbb{R}^d$, and define the *depth* $\Delta(p)$ of a point $p \in \mathbb{R}^d$ as the minimum number of points of $P$ on either side of any hyperplane passing through $p$. A point with depth at least $\delta n$ is called a *$\delta$-center point*. A $1/(d+1)$-center point is called just a *center point*, and it generalizes the concept of median. It has been proven using Helly's Theorem that any point set has a center point [16]. For $\mathbb{R}^2$, given some $k \leq n/2$, the set of points with depth at least $k$ is called $k$-th *depth contour*, denoted by $D_k$. Center points and depth contours are widely used in the analysis of rank and order statistics in point sets and bivariate distributions [20], computing separators for planar graphs [26], and other areas, thus providing a second motivation for studying the kinetic or parametric versions of these problems.

This notion of kinetic medians and center points is related to, but distinct from more classic kinetic facility-location problems [8], where the goal is to maintain certain mobile servers near groups of mobile clients. An important special instance of that is the kinetic maintenance of clusters of mobile nodes so that the clustering is both nearly optimal and stable under motion [18]. In these client-server problems a metric notion of "being in the middle" is more appropriate (as in the 1-center problem in which we minimize the maximum distance from server to the most distant client), while our emphasis is on combinatorial medians — since our primary goal is to use them for divide-and-conquer data structures. However, we are also interested in applications where the medians must have a physical embodiment and not just be data structure helpers, and thus we require that their motion be continuous (so as to be physically realizable).

Like in the kinetic facility-location problems, we provide a trade-off between the quality of our medians and their continuity and stability under motion. We study both exact and approximate algorithms for kinetic medians (in $\mathbb{R}^1$) and kinetic center points and depth contours (in $\mathbb{R}^2$). As we will see, in both cases the approximate algorithms offer far greater stability and maintenance efficiency, for a very modest loss in the quality of the partitions they can generate. Another advantage of approximation algorithms is that we can not only maintain the (median or) center point of current configuration of the points but we can also maintain a compact representation of the trajectory of an approximate (median or) center point over the entire motion of points, so we can answer various queries on the center point of any future configuration of points. Such queries are central to spatial-temporal database systems. A main tool we develop for these approximate algorithms is the efficient maintenance of an $\varepsilon$-approximation of a range space under point insertion and deletion — which we believe to be of independent interest.

Most of our algorithms are based on the *kinetic data structure* (KDS) framework, originally proposed by Basch *et al.* [6], which provides a toolbox for designing and analyzing data structures under motion. The basic idea in the kinetic framework is that even though the objects move continuously, the relevant combinatorial structure changes only at certain discrete times — therefore one need not update the data structure continuously. See the review article by Guibas [19] for details.

**Related work.** Finding the median (or more generally the point of rank $k$, for some given $k$) of a set of points in $\mathbb{R}^1$ can be done in linear time [13]. If the points move on the real line, then maintaining the point of rank $k$ is closely related to the concept of *levels* in the arrangement [1] defined by the trajectories in the $xt$-plane, as defined next. The *level* of a point $p \in \mathbb{R}^2$ in an arrangement $\mathcal{A}(\Gamma)$ defined by a set $\Gamma$ of $x$-monotone curves is the number of curves in $\Gamma$ lying below $p$. Let $\Lambda_k(\Gamma)$ denote the closure of the edges of $\mathcal{A}(\Gamma)$ whose level is $k$; $\Lambda_k(\Gamma)$ is an $x$-monotone curve and it is called the $k$-level of the arrangement. Hence,

[1] The arrangement $\mathcal{A}(\Gamma)$ is the planar decomposition induced by $\Gamma$. Its vertices are the intersection points of curves in $\Gamma$, its edges are maximal (open) connected portions of curves that do not contain a vertex, and its faces are the connected components of $\mathbb{R}^2 \setminus \bigcup \Gamma$.

the point of rank $k$ at time $t_0$ is given by the trajectory that is on the $k$-level for $t = t_0$. Set $\lambda_k(\Gamma) = |\Lambda_k(\Gamma)|$, and define $\lambda_k(n) = \max\{\lambda_k(\Gamma)\}$, where the maximum is taken over all sets $\Gamma$ of $n$ curves of a given type (lines, for example, or algebraic curves of a certain maximum degree). Maintaining the median thus reduces to computing the $n/2$-level, and $\lambda_{n/2}(n)$ bounds the number of changes of the median if the trajectories in the $xt$-plane are of the given type. If $\Gamma$ is a set of $n$ lines, then $\lambda_k(\Gamma) = O(n^{4/3})$ [15]. For general curves, a recent result of Chan [9] implies that $\lambda_k(\Gamma) = O(n^{2-1/2s})$, where $s$ is the maximum number of intersection points between two curves.

Maintaining an $\varepsilon$-approximate median reduces to computing an $\varepsilon$-approximate $n/2$-level, that is, an $x$-monotone curve lying between the $(1-\varepsilon)n/2$-level and the $(1+\varepsilon)n/2$-level. There has been a lot of work on computing approximate levels. Edelsbrunner and Welzl [17] showed that an $\varepsilon$-approximate median level with at most $\lceil \lambda_{n/2}(n)/(\varepsilon n) \rceil$ edges can be computed for line arrangements. Later Matoušek [22] proposed an algorithm for obtaining an $\varepsilon$-approximate median level with constant complexity (depending on $\varepsilon$ only) for an arrangement of $n$ lines. This idea was further explored by Agarwal *et al.* [3] to obtain efficient off-line and on-line maintenance of an approximate median level of an arrangement of lines or line segments. They compute an $\varepsilon$-approximate median with size $O(1/\varepsilon^2)$ in time $O(n \log n/\varepsilon^2)$.

In 2-D, a center point can be computed in linear time [21]. Clarkson *et al.* [12] proposed a randomized algorithm to compute an $\varepsilon$-approximate center point. The computational cost is polynomial in only the dimension and $1/\varepsilon$. Miller *et al.* used $O(n^2)$ time and space to find all the depth contours, by computing the arrangement of the $n$ lines in the dual plane [25]. A single $k$-th depth contour can be computed in $O(n \log^2 n)$ time [10]. To our knowledge, there are no prior results on maintaining the exact/approximate depth contours or center points under motion.

**Our results.** We start by briefly looking at the (rather easy) problem of maintaining medians and points of a given rank $k$ for points moving in $\mathbb{R}^1$ (Section 2.1). Then we present two KDS's for maintaining a center point of a set of $n$ points moving in the plane (Sections 2.2 and 2.3). The first KDS actually maintains the $k$-th depth contour for any given $0 \le k < n$. As a byproduct, it also gives a KDS for maintaining the entire center region, i.e., the set of all center points. It requires $O(n^2)$ certificates and processes $O(n^{4+\delta})$ events under pseudo-algebraic motion of points, each event requiring $O(\log^2 n)$ time. The second KDS maintains a subset of the center points. It requires only $O(n \log n)$ certificates but processes $O(n^{7+\delta})$ events in the worst case. As stated above, these are the first results known for this problem.

Since these exact algorithms are quite expensive, we then study approximation algorithms. We first describe an algorithm for maintaining an *$\varepsilon$-approximation* $A \subseteq S$ of a finite range space $(S, \mathcal{R})$ under insertions and deletions of points (see Section 3). Because $\varepsilon$-approximations are a widely-used tool in computational geometry, this result is of interest on its own. We show that a center point of $A$ for a suitably defined range space is an $\varepsilon$-approximate center point of $S$. Whenever we compute $A$, we also compute in $O(1/\varepsilon^{O(1)})$ time a $t$-monotone polygonal curve $\gamma$ so that $\gamma(t)$ is a center point of $A(t)$. Therefore for any $t$, we can compute in $O(\log(1/\varepsilon))$ time an $\varepsilon$-approximate center point of the points in $S(t)$ based on their current trajectories (Section 4.1). The set $A$ does not change as long as the trajectories of the points in $S$ don't change. Whenever a point $p$ changes its trajectory, we delete $p$ from $S$ and re-insert it with the new trajectory. Our algorithm can update $A$ in $(\log(n)/\varepsilon)^{O(1)}$ time. The same idea can also be used to maintain an $\varepsilon$-approximate median of a set of points moving in $\mathbb{R}^1$.

The $\varepsilon$-approximation based algorithm maintains an $\varepsilon$-approximate median that changes $O(1/\varepsilon^4 \log^2(1/\varepsilon))$ times, and the total time spent is $O(n/\varepsilon^{O(1)})$. If the trajectories of the points are known in advance, then one can improve the running time and maintain a more stable median (Section 4.2). More precisely, if the

trajectories of points are fixed-degree algebraic curves, then we can compute in $O(n + 1/\varepsilon^{O(1)})$ time a $t$-monotone curve $\gamma$ with $O(1/\varepsilon^2)$ breakpoints so that $\gamma(t)$ is an $\varepsilon$-approximate median of $S(t)$. If furthermore the points in $S$ move with fixed velocities, we can give a Las Vegas algorithm that reduces the number of breakpoints on $\gamma$ to $O(1/\varepsilon^{4/3}\log^2(1/\varepsilon))$ by spending $O(n/\varepsilon^{1/3}\log(1/\varepsilon))$ expected time, , thus improving the result of Agarwal et al. [3].

# 2 Exact Algorithms

We begin by describing a KDS for maintaining the median of a set of points moving in $\mathbb{R}^1$. Next, we describe a KDS to maintain the depth contour of a set $S$ of points moving in the plane, which as a byproduct also gives a KDS for maintaining the *center region*, which is the set of all center points of $S$. Finally, we describe an alternate more space-efficient KDS for maintaining a center point of $S$.

## 2.1 Maintaining the median

Maintaining the exact median of a set $S$ of $n$ points moving in $\mathbb{R}^1$, or more generally, maintaining the point of rank $k$ for a fixed $k$, can be done by adapting the HeapSweep algorithm [7]. We maintain (i) the point of rank $k$, (ii) the maximum of $S_{<k}(t)$, the points of rank less then $k$, in a kinetic tournament, and (iii) the minimum of $S_{>k}$, the points with rank greater than $k$, in a kinetic tournament. This is easy to maintain. E.g., when the maximum of $S_{<k}(t)$ swaps with the point of rank $k$, it is deleted from $S_{<k}(t)$, and it becomes the new point of rank $k$; the old point of rank $k$ is inserted into $S_{<k}(t)$. Following the analysis of [7], we prove the following.

**Theorem 2.1** *Let $S$ be a set of $n$ points moving in $\mathbb{R}^1$, and let $k$ be an integer. We can maintain the point of $S$ of rank $k$ using a KDS that uses $O(n)$ certificates. Assuming pseudo-algebraic motion, the number of events processed by the KDS is $O(\lambda_k(n)\log^2 n)$, and each event can be handled in $O(\log n)$ time.*

## 2.2 Maintaining the depth contour and center region

Given a set $S$ of $n$ points moving in $\mathbb{R}^2$ and an integer $0 \le k < \lceil n/2 \rceil$, we describe a KDS for maintaining the $k$-th depth contour $D_k$ of $S$ as the points in $S$ move. The center region of $S$ is simply $D_{\lfloor n/3 \rfloor}$. Note that the $k$-th depth contour of a set $P$ of $n$ stationary points in the plane is also closely related to the $k$-th and $(n-k)$-th levels in arrangements: if we dualize [14] $P$, we obtain a set $P^*$ of $n$ lines, and a point at depth $k$ dualizes to a line lying between $\Lambda_k(P^*)$ and $\Lambda_{n-k}(P^*)$. Hence, the dual of points whose depth is at least $k$ is the region lying between the lower hull of $\Lambda_{n-k}(S^*)$ and the upper hull of $\Lambda_k(S^*)$. Thus the problem at hand reduces to the following. Let $L = \{\ell_1, \ldots, \ell_n\}$ be a set of $n$ lines in the plane, each moving independently, i.e., $\ell_i : y = a_i(t) + b_i(t)x$, where $a_i(\cdot)$ and $b_i(\cdot)$ are polynomials in $t$. We wish to maintain the upper (or lower) hull of $\Lambda_k(L)$, the $k$-th level of $\mathcal{A}(L)$, as the lines in $L$ move.

For each line $\ell \in L$, let $v^-(\ell)$ (resp. $v^+(\ell)$) be the leftmost (resp. rightmost) point of $\Lambda_k(L) \cap \ell$. Let $V = \{v^-(\ell), v^+(\ell) \mid \ell \in L\}$. The upper hull of $\Lambda_k(L)$ is the same as that of $V$. We maintain the upper hull of $V$ using the kinetic data structure described in [6]. In more detail, we proceed as follows.

For a line $\ell \in L$, let $\Sigma(\ell)$ be the sequence of vertices of $\mathcal{A}(L) \cap \ell$, sorted from left to right. $\Sigma(\ell)$ partitions $\ell$ into the edges of $\mathcal{A}(L)$. For each edge $e$, let $\lambda(e)$ be the level of the points in $e$. Note that if $e'$ lies immediately to the right of $e$, then $\lambda(e') = \lambda(e) \pm 1$ depending on the local geometric situation. We maintain the set $E(\ell)$ of all the edges on $\ell$ whose level is $k$. We add the left endpoint of the leftmost edge

and the right endpoint of the rightmost edge of $E(\ell)$ (i.e., $v^-(\ell), v^+(\ell)$) to $V$. For each $\ell$, we maintain: (i) the sequence of vertices and edges of the arrangement on $\ell$, (ii) $E(\ell)$, and (iii) $v^-(\ell)$, $v^+(\ell)$. In addition, we also use a KDS to maintain the upper hull of $V$, as the lines in $L$ move [6]. We will also need it to handle insertion and deletion of points.

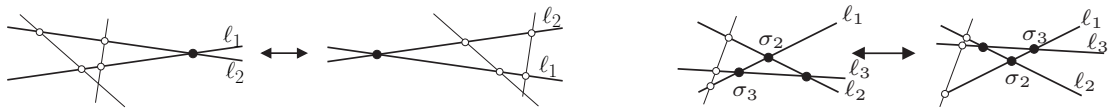



Figure 1: (i) An event of type (E1); $\ell_1$ and $\ell_2$ become parallel. (ii) An event of type (E2); $\ell_1, \ell_2, \ell_3$ become concurrent.

There are three types of events that the algorithm has to handle to maintain the above structures:

(E1) *Two lines $\ell_1, \ell_2$ become parallel.* Suppose $\ell_1 \cap \ell_2$ was the rightmost intersection point on $\ell_1$ (and $\ell_2$) before the event. Then $\ell_1 \cap \ell_2$ becomes the leftmost intersection point on $\ell_1$ (and $\ell_2$) after the event; see Fig. 1 (i). We update the ordering of $\Sigma(\ell_1)$ and $\Sigma(\ell_2)$. The two rightmost edges on $\ell_1$ and $\ell_2$ are merged and the leftmost edge on each of $\ell_1$ and $\ell_2$ is split into two edges. The level of the new edge on $\ell_1$ (resp. $\ell_2$) can be computed in $O(1)$ time from the level of its neighboring edge on $\ell_1$ (resp. $\ell_2$). If necessary, we also update $E(\ell_1)$, $E(\ell_2)$, $V$, and the KDS for maintaining $\mathcal{UH}(V)$. There are $O(n^2)$ such events, each requiring $O(\log n)$ time to update $\mathcal{A}(L)$ and $E(\ell)$. In addition, at each such event, we may also performs $O(1)$ insert/deletion operations on the KDS, which in turn can generate additional internal events for the KDS.

(E2) *Three lines $\ell_1, \ell_2, \ell_3$ become concurrent.* In this case $\sigma_2 = \ell_1 \cap \ell_2$ and $\sigma_3 = \ell_1 \cap \ell_3$ are adjacent along $\ell_1$. After the event their ordering along $\ell_1$ flips; see Fig. 1 (ii). We therefore swap $\sigma_2$ and $\sigma_3$ in $\Sigma(\ell_1)$, and update the level $\lambda(e)$ of the edge $e \subseteq \ell_1$ induced by $\sigma_2$ and $\sigma_3$ in $O(1)$ time. If necessary, we also update $E(\ell_1)$, $V$, and the KDS for maintaining $\mathcal{UH}(V)$. We repeat the same procedure for $\ell_2$ and $\ell_3$. There are $O(n^3)$ such events, each requires $O(\log n)$ time and $O(1)$ insert/delete operations in the KDS.

(E3) *An event of KDS for maintaining $\mathcal{UH}(V)$.* Roughly speaking, the KDS by Basch *et al.* [6] to maintain convex hull of a set of moving points is a tree data structure, which partitions $V$ into two roughly equal subsets $V_1$ and $V_2$ and recursively maintains $\mathcal{UH}(V_1)$ and $\mathcal{UH}(V_2)$. The root of the tree maintains the upper hull of $\mathcal{UH}(V_1) \cup \mathcal{UH}(V_2)$ and a linear number of "certificates" that keep track of the combinatorial structure of $\mathcal{UH}(V)$ and some additional book-keeping information. The KDS triggers an *event* when one of the certificates is no longer true, indicating that the structure needs to be updated. In addition to the events caused by the motion of points, insertion/deletion of a point also triggers events in KDS. Each of these events can be processed in $O(\log^2 n)$ time, as described in [6]. The following lemma bounds the number of these events.

**Lemma 2.1** *There are $O(n^{4+\delta})$ events of type (E3).*

*Proof.* We only count the number of events at the root of the KDS. The events at other nodes can be counted by a recursive argument. Let $q$ be a vertex of $\mathcal{A}(L)$ that was in $V$ during the time interval $[t_1, t_2]$. If $q$ appears in $V$ more than once, we regard these occurrences of $q$ as different points. Let $q^*(t)$ be the line

dual to $q(t)$. Define $\gamma_q$ to be the ruled surface in the $txy$-space swept by $q^*(t)$ during the interval $[t_1, t_2]$, i.e., $\gamma_q = \bigcup_{t_1 \le t \le t_2} t \times q^*(t)$. Regard each $\gamma_q$ as the graph in the $txy$-space of a partial bivariate function $\gamma_q(t, x)$. Let $G_1$ (resp. $G_2$) be the set of these surfaces corresponding to the points that ever appeared in $V_1$ (resp. $V_2$). Following the same argument as in [6], we can show that the number of events of type (E3) at the root, including the ones triggered by insertion/deletion of points, is proportional to the number of vertices in the overlay of the upper envelopes of surfaces in $G_1$ and $G_2$. It thus suffices to bound the number of such vertices.

We partition the $t$-axis into $O(n^2)$ slices so that there are at most $2n$ events of type (E1) and (E2) in each slice. For each slice $\Delta$, let $G_1^\Delta$ (resp. $G_2^\Delta$) be the set of surfaces in $G_1$ (resp. $G_2$) that intersect the slice $\Delta$. Since the number of vertices in $V$ at any given time is at most $2n$, we have $|G_1^\Delta| + |G_2^\Delta| \le 4n$. By a result of Agarwal *et al.* [5], the number of vertices in the overlay of the upper envelopes of $G_1^\Delta$ and $G_2^\Delta$ is $O(n^{2+\delta})$. Summing this bound over all $O(n^2)$ slices and using a recursive argument to bound the number of events at other nodes of the tree, we obtain the desired bound. □

The KDS for maintaining $\mathcal{UH}(V)$ maintains $O(n \log n)$ certificates. We need additional $O(n^2)$ certificates to detect events of type (E1) and (E2): (i) leftmost and right vertices along each line of $L$, and (ii) adjacent pairs of vertices of $\mathcal{A}(L)$ along each line of $L$. Hence, we conclude the following.

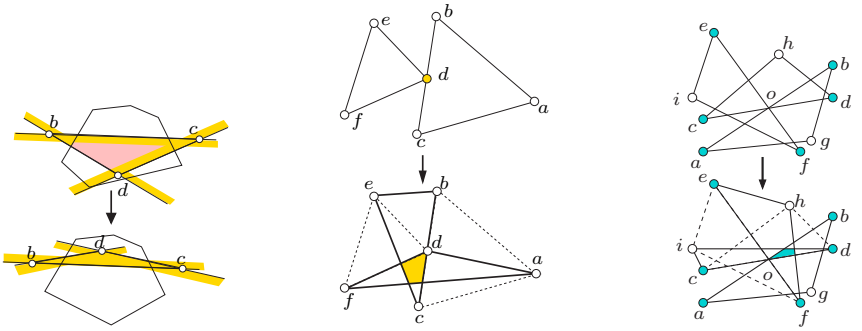
**Theorem 2.2** *Let $L$ be a set of $n$ lines in the plane, each moving independently, and let $0 \le k < n$ be an integer. We can maintain the upper and lower hulls of $\Lambda_k(L)$ using $O(n^2)$ certificates. The number of events processed by the algorithm under the algebraic motion of $L$ is $O(n^{4+\delta})$, and each event can be processed in $O(\log^2 n)$ time.*

**Corollary 2.1** *Let $S$ be a set of $n$ points moving in the plane. We can maintain the center region of $S$ or, more generally, the $k$-th depth contour of $S$, using $O(n^2)$ certificates. The algorithm processes $O(n^{4+\delta})$ events under algebraic motion of $S$, and each event can be processed in $O(\log^2 n)$ time.*

### 2.3 A space-efficient algorithm for maintaining a center point

We now describe another KDS for maintaining a center point of a set $S$ of $n = 3m$ points moving in the plane. Unlike the previous algorithm, the new KDS uses only $O(n \log n)$ certificates. First, let us assume that the points in $S$ are stationary. A result by Tverberg [27] (see also [24]) implies that $S$ can be partitioned into $m$ triples $S_1, \ldots, S_m$ so that the intersection of triangles $\Delta_i = \mathrm{conv}(S_i)$, $1 \le i \le m$, is nonempty. Such a partition is called a *Tverberg partition*, and any point in $\Delta = \bigcap_{i=1}^m \Delta_i$ is called a *Tverberg point*. Any point in $\Delta$ is also a center point. Moreover, a Tverberg partition in the plane can be computed in $O(n \log n)$ time [26]. Since each Tverberg triangle $\Delta_i$ can be regarded as the intersection of three halfplanes, each bounded by a line passing through a pair of points in $\Delta_i$, the region $\Delta$ is the intersection of a set $\mathcal{H}$ of $3m = n$ halfplanes. In the dual plane, a halfplane $h \in \mathcal{H}$ is mapped to a point $h^*$ and the intersection of the halfplanes maps to the convex hull of the set $\mathcal{H}^*$ of points. Therefore we use a KDS to maintain the convex hull $\mathcal{CH}(\mathcal{H}^*)$ [6]; it uses $O(n \log n)$ certificates.

Now as the points in $S$ move, we wish to maintain $\Delta$. More precisely, we compute an initial Tverberg partition of $S$ in $O(n \log n)$ time and maintain $\Delta$ using the kinetic convex hull structure described by Basch *et al.* [6], with the following twist. Whenever $\Delta$ shrinks to a point, we update the underlying Tverberg partition so that the interior of $\Delta$ after the modification becomes nonempty. The idea is similar to the one used by Tverberg [28]. There are two types of events that cause the interior of $\Delta$ to become empty.




(i) (ii) (iii)

Figure 2: (i) Vertex $d$ of $\triangle bcd$ crosses the edge $bc$. (ii) Vertex $d$ of $\triangle def$ crosses the edge $bc$ of $\triangle abc$. (iii) An edge-edge event.

(i) *Edge-vertex event.* Suppose a point $d \in \Delta_i$ crosses a line passing through a pair of points $b, c \in \Delta_j$. If $i = j$ (see Fig. 2 (i)), then basically the halfplanes defining $\Delta_i$ change. We update $\mathcal{H}$ and modify the KDS accordingly. The Tverberg partition does not change.

If $i \neq j$, then we update the Tverberg partition by redistributing the points of $S_i \cup S_j$ into two triples. Let $\Delta_i = \triangle def$ and $\Delta_j = abc$. Then we set $S_i = \{a, d, f\}$ and $S_j = \{b, c, e\}$. It can be checked that the new $\Delta_i \cap \Delta_j \neq \emptyset$ and therefore $\Delta$ has nonempty intersection; see Fig. 2 (ii). We update the set $\mathcal{H}$ and the KDS.

(ii) *Edge-edge event.* Suppose three edges, coming from distinct triangles, become concurrent, see Fig. 2 (iii) for an example. We can again ensure that $\Delta$ remains non-empty by regrouping the vertices of the three triangles involved. We omit the details.

Both events involve $O(1)$ updates in $\mathcal{H}$ and KDS. There are $O(n^3)$ vertex-edge events and $O(n^6)$ edge-edge events, so the total number of updates on $\mathcal{H}$ is $O(n^6)$. By partitioning the time-axis into $O(n^5)$ slices with $n$ events each, and following the analysis of Lemma 2.1, we get:

**Theorem 2.3** *Let $S$ be a set of $n$ points moving in the plane. We can maintain a center point of $S$ using $O(n \log n)$ certificates. The algorithm processes $O(n^{7+\delta})$ events under algebraic motion of $S$, and each event can be processed in $O(\log^2 n)$ time.*

# 3 Maintaining $\varepsilon$-approximations

A range space is a pair $X = (S, \mathcal{R})$, where $S$ is a set and $\mathcal{R} \subset 2^S$. $S$ is called the set of *points*, and $\mathcal{R}$ is called the set of *ranges*. In the sequel we deal with finite range spaces, where $S$ is finite. A subset $A \subseteq S$ is called an *$\varepsilon$-approximation* for $X$ if, for every range $R \in \mathcal{R}$, $||A \cap R|/|A| - |R|/|S|| < \varepsilon$. The concept of $\varepsilon$-approximation plays an important role in computational geometry, and forms the basis of many efficient randomized and deterministic algorithms and data structures [11]. In this section we describe an algorithm for maintaining an $\varepsilon$-approximation of a range space as points are inserted or deleted. Maintaining an $\varepsilon$-approximation under insertions and deletions is trivial if we allow randomization, because we can just resample the $\varepsilon$-approximation after each update, but we cannot efficiently verify whether the random sample

is an $\varepsilon$-approximation. In the kinetic setting, where the number of updates of trajectories could be quite large, this Monte Carlo approach is not viable because the size of the sample needed to guarantee success with high probability would have to be large. Thus, a deterministic solution in this case is desirable, which guarantees that we are always working with an $\varepsilon$-approximation. We first briefly review an algorithm for constructing an $\varepsilon$-approximation [11] and then describe how we dynamize it. Before proceeding we state some well-known properties of $\varepsilon$-approximations.

**Lemma 3.1** *(i) If $A'$, $B'$ are $\varepsilon$-approximations for the range spaces $(A, \mathcal{R})$ and $(B, \mathcal{R})$ respectively, and $|A| = |B|$, $|A'| = |B'|$, then $A' \cup B'$ is an $\varepsilon$-approximation of $(A \cup B, \mathcal{R})$.*

*(ii) If $A''$ is an $\varepsilon$-approximation for $(A', \mathcal{R})$ and $A'$ is an $\varepsilon'$-approximation for $(A, \mathcal{R})$, then $A''$ is an $(\varepsilon + \varepsilon')$-approximation of $(A, \mathcal{R})$.*

*(iii) Halving: Given $(A, \mathcal{R})$, a range space of VC-dimension [2] $d$, one can compute a bichromatic coloring of $A$ such that the set $A'$ of black points is an $O((\log(n))/\sqrt{n})$-approximation for $(A, \mathcal{R})$, and $|A'| = |A|/2$. This takes $O(n^{d+1})$ time, where $n = |A|$.*

**Construction of an $\varepsilon$-approximation.** Assume $|S| = 2^i$. The algorithm can easily be modified to handle general values of $n$, but we omit the details here. Lemma 3.1 offers a rather natural way of computing an $\varepsilon$-approximation of a point set $S$: build a balanced binary tree over the points of $S$, and compute the $\varepsilon$-approximation in a bottom-up fashion. At every node $v$, take the two approximations computed for the children, and merge them together to form an approximation for all the points stored in the subtree of $v$. If this set is too large, halve it using Lemma 3.1 (iii).

It is easy to verify that by fine-tuning the above construction we can get an $\varepsilon$-approximation to $S$ in the root of this tree. Indeed, we just perform merging in nodes that have less than $\mu = \frac{c}{\varepsilon^2}(\log n \log(\log(n)/\varepsilon))^2$ points in their subtree. In every other node, we perform merging and halving. The merging itself does not introduce any error. The only source for error is the halving, and the total error created is bounded by the sum of the errors introduced in each level. We have that the approximation quality is

$$\sum_{i=1}^{\lceil \log_2 n \rceil} O\left(\frac{\log \mu}{\sqrt{\mu}}\right) \leq \frac{\varepsilon}{2},$$

for $c$ large enough. However, the set stored in the root of the tree is too big. We further shrink it by repeated halving till it is of size $O(1/\varepsilon^2 \log(1/\varepsilon))$. This adds an extra error of $\varepsilon/2$. Overall, the resulting set is an $\varepsilon$-approximation of size $O(1/\varepsilon^2 \log 1/\varepsilon)$.

**Lemma 3.2** *Given a range space $X = (S, \mathcal{R})$ of VC-dimension $d$ and a parameter $\varepsilon$, we can compute an $\varepsilon$-approximation of $X$ of size $O(1/\varepsilon^2 \log(1/\varepsilon))$ in time $O(n\mu^d)$, where $|S| = n$ and $\mu = \frac{c}{\varepsilon^2}(\log n \log(\log(n)/\varepsilon))^2$.*

The above result can be extended to a range space $X = (S, \mathcal{R})$ in which each point $p$ of $S$ is assigned a nonnegative weight $w(p)$ and a subset $A \subseteq S$ is called an $\varepsilon$-approximation of $S$ if $|w(A \cap R)/w(A) - w(R)/w(S)| < \varepsilon$, where $w(N) = \sum_{p \in N} w(p)$ of $S$.

**Theorem 3.1** *Given a weighted range space $X = (S, \mathcal{R})$ of $n$ points (the weights are non-negative real numbers) of VC-dimension $d$ and a parameter $\varepsilon > 0$, we can compute an $\varepsilon$-approximation for $X$ in $O\big(n\mu^d \log(1/\varepsilon)\big)$ time, where $\mu = \frac{c}{\varepsilon^2}(\log n \log(\log(n)/\varepsilon))^2$.*

[2] The *VC-dimension* of a range space $X = (S, \mathcal{R})$ is the size of the largest subset $N \subseteq S$ such that $|\{N \cap r \mid r \in \mathcal{R}\}| = 2^{|N|}$.

**Handling insertions and deletions.** We are going to maintain the current set $S$ as a set of $O(\log n)$ perfectly balanced trees, where a tree of rank $i$ has $2^i$ points stored in it. Every node of those trees maintains in it an approximation to all the points in the subtree rooted at this node. The size of the approximation is $m = c(\log^2 n)/\varepsilon^2$, where $c$ is a large enough constant.

When inserting a point, we create a tree that has only this point in it. When deleting a point from a tree, we break it into $O(\log n)$ trees. After every such operation, we go over the collection of trees sorted in increasing order by their rank, and merge every two trees $T_1, T_2$ that have the same rank $r$. This is done by creating a new tree with both trees as subtrees, and computing the approximation associated with the root by merging the two approximations associated with its two children, and halving them. The two old trees are removed from the collection, and the new tree is added with its rank being $r + 1$. Clearly, after $O(\log n)$ such operations, the collection of trees becomes legal again.

This yields $M = O(\log n)$ sets $S_1, \ldots, S_M$, each one $\varepsilon/4$-approximating the points stored in the tree it was extracted from. We can not just merge those sets to form an $\varepsilon$-approximation for the whole point set, as the original point sets have different cardinalities. Instead, we associate with all the points of $S_i$ a weight which is $2^i/|S_i|$ (the set $S_i$ associated with a tree of rank $i$ containing $2^i$ points). $S_1 \cup \ldots \cup S_M$ with those associated weights is an $\varepsilon/4$-approximation for the current point set. We can finally normalize this into an unweighted $\varepsilon$-approximation, by computing an $\varepsilon/4$-approximation of $(S_1 \cup \ldots \cup S_m, \mathcal{R})$ using Theorem 3.1. The resulting approximation is clearly the required $\varepsilon$-approximation. Omitting the details of the analysis, we conclude the following.

**Theorem 3.2** *Given a range space $X = (S, \mathcal{R})$ of VC-dimension $d$ and a parameter $\varepsilon$, one can (deterministically) maintain an $\varepsilon$-approximation of $X$ of size $O(1/\varepsilon^2 \log(1/\varepsilon))$, in $O\left(\frac{\log^{2d+3} n}{\varepsilon^{2d+2}} (\log(\log(n)/\varepsilon))^{2d+2}\right)$ time per insertion or deletion.*

# 4 Approximation Algorithms

In this section we describe approximation algorithms for maintaining $\varepsilon$-approximate median and center points. We first show that we can get fast approximation algorithms using our results on $\varepsilon$-approximations. As mentioned in the introduction, they not only maintain an $\varepsilon$-approximate median or center point of the current configuration of points sets, but also compute a $t$-monotone curve $\gamma$ such that $\gamma(t)$ is an $\varepsilon$-approximate median or center point of $S(t)$. Next we show that we can develop even faster algorithms in the off-line setting, where the trajectories of input points are given in advance, that maintain more stable $\varepsilon$-approximate medians.

## 4.1 The $\varepsilon$-approximation based algorithms

We show that the algorithm described in Section 3 can be used to maintain an $\varepsilon$-approximate median and center point. Let $S$ be a set of $n$ points moving in $\mathbb{R}^1$. Let $R$ be the set of vertically downward rays in the $tx$-plane, i.e., $R = \{(-\infty, x] \times t \mid x, t \in \mathbb{R}\}$. For an element $\rho = (-\infty, x] \times t \in R$, let $S_\rho = \{p \mid$ there is a $t$ such that $p(t) \in \rho\}$. We define the range space $\mathcal{M} = (S, \{S_\rho \mid \rho \in R\})$. The following lemma is straightforward.

**Lemma 4.1** *Let $A \subseteq S$ be an $\varepsilon$-approximation of $S$ for the range space $\mathcal{M}$. Then for any $t$, the median of $A(t)$ is an $\varepsilon$-approximate median of $S(t)$.*

It can be proved that if the trajectories of $S$ are algebraic then the VC-dimension of $\mathcal{M}$ is finite. Therefore we can use the algorithms of the previous section to maintain an $\varepsilon$-approximation of $S$. We can compute the median level of the trajectories of $A$. Since we compute the entire median level, there are no events unless the trajectory of a point in $S$ changes. When the trajectory of a point changes, $A$ is recomputed in $(\log(n)/\varepsilon)^{O(1)}$ time using Theorem 3.2, and we recompute the median level of $A$ in $(1/\varepsilon)^{O(1)}$ additional time. We thus obtain the following.

**Theorem 4.1** *Let $S$ be a set of points moving in $\mathbb{R}^1$, and let $\varepsilon > 0$ be a parameter. Assuming that the motion of $S$ is algebraic, we can construct a data structure that can be updated in $(\log(n)/\varepsilon)^{O(1)}$ time whenever the trajectory of a point in $S$ changes, and that, for any time $t$, can return an $\varepsilon$-approximate median of $S(t)$ in $O(\log(1/\varepsilon))$ time based on the current trajectories of $S$.*

Using a similar argument, we can maintain an $\varepsilon$-approximate center point of $S$.

**Theorem 4.2** *Let $S$ be a set of points moving in $\mathbb{R}^2$, and let $\varepsilon > 0$ be a parameter. Assuming that the motion of $S$ is algebraic, we can construct a data structure that can be updated in $(\log(n)/\varepsilon)^{O(1)}$ time whenever the trajectory of a point in $S$ changes, and that, for any time $t$, can return an $\varepsilon$-approximate center point of $S(t)$ in $O(\log(1/\varepsilon))$ time based on the current trajectories of $S$.*

## 4.2 Faster off-line algorithms for approximate median

We show that an $\varepsilon$-approximate median of a set $S$ of $n$ points moving in $\mathbb{R}^1$ can be computed more efficiently in the off-line setting. However, unlike the algorithm described in Section 4, the $\varepsilon$-approximate median we maintain is not necessarily one of the input points, and we cannot change the trajectory of points on-line. Let $L$ denote the set of trajectories of points in $S$. We first describe an algorithm for the algebraic motion that computes, in $O(n + 1/\varepsilon^{O(1)})$ time, a $t$-monotone curve $\gamma$ (in the $tx$-plane) with $O(1/\varepsilon^2)$ breakpoints so that $\gamma(t)$ is an $\varepsilon$-approximate median of $S(t)$. If the points move with fixed velocity, then the number of breakpoints can be reduced to $O(1/\varepsilon^{4/3}\log^2(1/\varepsilon))$ by spending $O(n/\varepsilon^{1/3}\log(1/\varepsilon))$ expected time. Because of lack of space we omit many details.

**The case of algebraic motion.** Set $r = c/\varepsilon$ for some constant $c > 0$, and let $\mu$ be the maximum degree of the polynomials defining the trajectories of input points. Using the linearization technique, we compute in $O(n \log r + r^{\mu+1})$ time, a $(1/r)$-cutting $\Xi$ of $L$ [23, 4] of size $O(r^2) = O(1/\varepsilon^2)$.[3] Let $V$ be the set of vertices of $\Xi$; $|V| = O(r^{\mu+1})$. By preprocessing $V$ into a data structure, we can prove the following lemma.

**Lemma 4.2** *We compute in time $O(n \log r + r^{2(\mu+1)^2})$ time the level of every point of $V$ in $\mathcal{A}(L)$.*

Next, let $E$ be the set of edges of $\Xi$ whose both endpoints have level at least $n/2$. The level of all points on an edge of $\Xi$ with $u$ as one of its endpoints lies in the range $\lambda(u) \pm \varepsilon n/c$. We take the lower envelope $\gamma$ of $E$. Using the structural properties of $\Xi$, we can prove that $\gamma$ is a continuous curve with at most $O(1/\varepsilon^2)$ vertices and the level of every breakpoint in $\gamma$ is at most $n/2 + \varepsilon n/c$. The level of any point in $\gamma$ thus lies in the range $[n/2 - \varepsilon n/c, n/2 + 2\varepsilon n/c]$. If we choose $c > 4$, then $\gamma$ is an $\varepsilon$-approximate median level of $L$ and thus $\gamma(t)$ is an $\varepsilon$-approximate median of $S(t)$. Hence, we obtain the following.

[3] A $1/r$-cutting of $L$ is a partition of the plane into pairwise-disjoint "pseudo-trapezoids" so that at most $|L|/r$ arcs of $L$ cross each pseudo-trapezoid of $\Xi$.

**Theorem 4.3** *Let $S$ be a set of points moving in $\mathbb{R}^1$, and $\varepsilon > 0$ a constant. Assuming that their trajectories are algebraic with degree of motion $\mu$, we can compute in $O(n \log(1/\varepsilon) + 1/\varepsilon^{O(\mu^2)})$ time a $t$-monotone curve $\gamma$ of size $O(1/\varepsilon^2)$ so that the rank of $\gamma(t)$, for any $t \in \mathbb{R}$, lies between $n/2(1 \pm \varepsilon)$.*

**The case of linear motion.** We can compute an $\varepsilon$-approximate median with fewer breakpoints if the input points are moving with fixed velocities, as follows. Let $L$ be a set of lines in the plane. We use the randomized incremental algorithm by Agarwal *et al.* [1] for computing the level in the arrangement of lines. It chooses a random permutation $\ell_1, \ldots, \ell_n$ of $L$ and adds the lines one by one in this order. Let $L_i = \{\ell_1, \ldots, \ell_i\}$ and $\delta_i = c(n/i) \log i$ for some constant $c \geq 1$. After having inserted the first $i$ lines, the algorithm maintains a family $\mathcal{C}_i$ of cells of $\mathcal{A}(L_i)$ each of which has at least one vertex whose level with respect to $L$ lies in the range $[n/2 - \delta_i, n/2 + \delta_i]$. Actually, the algorithm maintains a "canonical" triangulation of the cells in $\mathcal{C}_i$, and for each triangle $\Delta$ in the triangulation it maintains the set $L_\Delta \subseteq L$ of lines that intersect the interior of $\Delta$. $L_\Delta$ helps constructing $\mathcal{C}_{i+1}$ from $\mathcal{C}_i$ efficiently. See the original paper for details.

In our case, instead of adding all the lines of $L$ we add only the first $r = (c/\varepsilon) \log(1/\varepsilon)$ lines of the permutation. As in the previous algorithm, we choose the edges of cells in $\mathcal{C}_r$ whose both endpoints have level at least $n/2$ and compute their lower envelope $\gamma$. Following the analysis in [1], we can argue that $\gamma$ is a continuous curve and its expected size is $O(n^{4/3}\delta_r^{2/3}) \times (r^2/n^2) = O((1/\varepsilon)^{4/3} \log^2(1/\varepsilon))$, and that the expected running time of the algorithm is $O((n/\varepsilon^{1/3}) \log(1/\varepsilon))$. If $\gamma$ has more than $O((1/\varepsilon)^{4/3} \log^2(1/\varepsilon))$ breakpoints or an edge of $\gamma$ intersects more than $\varepsilon n/4$ lines of $L$, we restart the algorithm. The expected number of iterations is $O(1)$. Hence, we conclude the following.

**Theorem 4.4** *Let $S$ be a set of points moving in $\mathbb{R}^1$ moving with fixed velocity, and $\varepsilon > 0$ a parameter. We can compute in expected $O((n/\varepsilon^{1/3}) \log(1/\varepsilon))$ time a $t$-monotone curve $\gamma$ of size $O(1/\varepsilon^{4/3} \log^2(1/\varepsilon))$ so that the rank of $\gamma(t)$, for any $t \in \mathbb{R}$, lies between $n/2(1 \pm \varepsilon)$.*

It would be interesting to also obtain faster off-line algorithms for approximate center points of moving points in $\mathbb{R}^2$. The above techniques do not seem to be directly applicable to this case.

# References


[1] P. K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf. Constructing levels in arrangements and higher order Voronoi diagrams. *SIAM J. Comput.*, 27:654–667, 1998.

[2] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.

[3] P. K. Agarwal, J. Gao, and L. J. Guibas. Kinetic medians and $kd$-trees. In *Proc. 10th Annu. European Sympos. Algorithms*, pages 5–16, 2002.

[4] P. K. Agarwal and J. Matoušek. On range searching with semialgebraic sets. *Discrete Comput. Geom.*, 11:393–418, 1994.

[5] P. K. Agarwal, O. Schwarzkopf, and M. Sharir. The overlay of lower envelopes and its applications. *Discrete Comput. Geom.*, 15:1–13, 1996.

[6] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. *J. Algorithms*, 31(1):1–28, 1999.

[7] J. Basch, L. J. Guibas, and G. D. Ramkumar. Reporting red-blue intersections between two sets of connected line segments. *Algorithmica*, 35:1–20, 2003.

[8] S. Bespamyatnikh, B. Bhattacharya, D. Kirkpatrick, and M. Segal. Mobile facility location. In *4th International Workshop on Discrete Algorithms and Methods for Mobile Computing & Communications*, pages 46–53, 2000.

[9] T. M. Chan. On levels in arrangements of curves, II: a simple inequality and its consequence. In *Proc. 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 544–550, 2003.

[10] T. M. Chan. An optimal randomized algorithm for maximum tukey depth. In *Proc. 15th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 423–429, 2004.

[11] B. Chazelle. *The Discrepancy Method*. Cambridge University Press, 2000.

[12] K. L. Clarkson, D. Eppstein, G. L. Miller, C. Sturtivant, and S.-H. Teng. Approximating center points with iterative Radon points. *Internat. J. Comput. Geom. Appl.*, 6:357–377, 1996.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press / McGraw-Hill, Cambridge, Mass., 2001.

[14] M. de Berg, M. van Kreveld, M. H. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.

[15] T. K. Dey. Improved bounds for planar $k$-sets and related problems. *Discrete Comput. Geom.*, 19(3):373–382, 1998.

[16] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Heidelberg, 1987.

[17] H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrangements with applications. *SIAM J. Comput.*, 15:271–284, 1986.

[18] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. *Discrete and Computational Geometry*, 30(1):45–63, 2003.

[19] L. J. Guibas. Kinetic data structures — a state of the art report. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Proc. 5th Workshop Algorithmic Found. Robot.*, pages 191–209. A. K. Peters, Wellesley, MA, 1998.

[20] J. Hodges. A bivariate sign test. *Annals of Math. Stat.*, 26:523–527, 1955.

[21] S. Jadhav and A. Mukhopadhyay. Computing a centerpoint of a finite planar set of points in linear time. *Discrete Comput. Geom.*, 12:291–312, 1994.

[22] J. Matoušek. Construction of $\epsilon$-nets. *Discrete Comput. Geom.*, 5:427–448, 1990.

[23] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[24] J. Matoušek. *Lectures on Discrete Geometry*. Springer, 2002.

[25] K. Miller, S. Ramaswami, P. Rousseeuw, T. Sellares, D. L. Souvaine, I. Streinu, and A. Struyf. Fast implementation of depth contours using topological sweep. In *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, pages 690–699, 2001.

[26] S. H. Teng. *Points, Spheres, and Separators: a unified geometric approach to graph partitioning*. PhD thesis, School Comp. Sci., Carnegie-Mellon Univ., 1990. Report CMU-CS-91-184.

[27] H. Tverberg. A generalization of Radon's theorem. *J. London Math. Soc.*, 41:123–128, 1966.

[28] H. Tverberg. A generalization of Radon's theorem II. *Bull. Australian Math. Soc.*, 24:321–325, 1981.