A Nearly Quadratic Bound for the Decision Tree Complexity of k-SUM*

Esther Ezra[1] **and Micha Sharir**[2]

1 **Georgia Institute of Technology, Atlanta, GA, USA**
eezra3@math.gatech.edu
2 **Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, Israel**
michas@post.tau.ac.il

Abstract

We show that the k-SUM problem can be solved by a linear decision tree of depth $O(n^2 \log^2 n)$, improving the recent bound $O(n^3 \log^3 n)$ of Cardinal *et al.* [7]. Our bound depends linearly on k, and allows us to conclude that the number of linear queries required to decide the n-dimensional KNAPSACK or SUBSETSUM problems is only $O(n^3 \log n)$, improving the currently best known bounds by a factor of n [28, 29]. Our algorithm extends to the RAM model, showing that the k-SUM problem can be solved in expected polynomial time, for any fixed k, with the above bound on the number of linear queries. Our approach relies on a new point-location mechanism, exploiting "ε-cuttings" that are based on *vertical decompositions* in hyperplane arrangements in high dimensions. A major side result of the analysis in this paper is a sharper bound on the complexity of the vertical decomposition of such an arrangement (in terms of its dependence on the dimension). We hope that this study will reveal further structural properties of vertical decompositions in hyperplane arrangements.

1998 ACM Subject Classification F.2.2 [Nonnumerical Algorithms and Problems] Computations on Discrete Structures, Geometrical Problems and Computations

Keywords and phrases k-SUM and k-LDT, linear decision tree, hyperplane arrangements, point-location, vertical decompositions, ε-cuttings

Digital Object Identifier 10.4230/LIPIcs.SoCG.2017.41

1 Introduction

Problem definition and the model. In this paper we study the k-SUM problem, and the more general *k-linear degeneracy testing* (k-LDT) problem. We define them formally:

▶ **Definition 1** (k-SUM)**.** Given a point $\mathbf{x} := (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$, decide whether there exist k indices $i_1, i_2, \ldots, i_k$ such that $x_{i_1} + x_{i_2} + \ldots + x_{i_k} = 0$.

In what follows, we assume that we are looking for a k-tuple of *distinct* coordinates. The case where some coordinates can be repeated more than once is also easy to handle, by a straightforward extension of the technique presented here, which we omit, for the sake of simplicity of presentation.

* Work on this paper by Esther Ezra has been supported by NSF CAREER under grant CCF:AF 1553354. Work on this paper by Micha Sharir was supported by Grant 892/13 from the Israel Science Foundation, by Grant 2012/229 from the U.S.–Israel Binational Science Foundation, by the Blavatnik Research Fund in Computer Science at Tel Aviv University, by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), and by the Hermann Minkowski-MINERVA Center for Geometry at Tel Aviv University.

▶ **Definition 2** (k-LDT)**.** Given a fixed k-variate linear function $f(y_1, \ldots, y_k) = a_0 + \sum_{i=1}^{k} a_i y_i$, where $a_0, a_1, \ldots, a_k$ are real coefficients, and a point $\mathbf{x} := (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$, decide whether there exist k indices $i_1, i_2, \ldots, i_k$ such that $f(x_{i_1}, x_{i_2}, \ldots, x_{i_k}) = 0$.[1]

By definition, k-SUM is a special case of k-LDT when we set $f(y_1, \ldots, y_k) = \sum_{i=1}^{k} y_i$. We note that the special case $k = 3$, the so-called 3-SUM problem, received considerable attention in the past two decades, due to its implications to conditional lower bounds on the complexity of many fundamental geometric problems; see [18] and below for a list of such problems. From now on we focus only on the k-SUM problem; the algorithm that we present applies more or less verbatim to k-LDT too.

Following the approach in Cardinal *et al.* [7] for k-SUM (see also [2, 15]), let H be the collection of the $\binom{n}{k}$ hyperplanes h in $\mathbb{R}^n$ of the form $x_{i_1} + x_{i_2} + \cdots + x_{i_k} = 0$, over all k-tuples $1 \le i_1 < i_2 < \cdots < i_k \le n$. Then the k-SUM problem can be reformulated as asking, for a query point $\mathbf{x}$, whether $\mathbf{x}$ lies on any hyperplane of H. This can be determined by locating $\mathbf{x}$ in the *arrangement* $\mathcal{A}(H)$ formed by those hyperplanes.

The model in which we consider this problem is the *s-linear decision tree*: Solving an instance of the problem with input $\mathbf{x} = (x_1, \ldots, x_n)$ is implemented as a search with $\mathbf{x}$ in some tree T. Each internal node v of T is assigned a linear function in the n variables $x_1, \ldots, x_n$, with at most s non-zero coefficients. The outgoing edges from v are labeled $<$, $>$, or $=$, indicating the branch to follow depending on the sign of the expression at v evaluated at $\mathbf{x}$. Leaves are labeled "YES" or "NO", where "YES" means that we have managed to locate $\mathbf{x}$ on a hyperplane of H, and "NO" means that $\mathbf{x}$ does not lie on any hyperplane. Each "YES" leaf has an edge labeled "=" leading to it (but not necessarily vice versa, because some of the tests may involve auxiliary hyperplanes that are not part of the input). To solve an instance of the problem, we begin at the root of T. At each node v that we visit, we test the sign at $\mathbf{x}$ of the linear function at v, and proceed along the outgoing edge labeled by the result of the test. We conduct this search until we reach a leaf, and output its label "YES" or "NO". At each internal node, the test (which we also refer to as a *linear query*) is assumed to cost one unit. All other operations are assumed (or rather required) not to depend on the specific coordinates of $\mathbf{x}$ (although they might depend on discrete data that has been obtained from the preceding queries with $\mathbf{x}$), and incur no cost in this model. Thus the length of the search path from the root to a leaf is the overall number of linear queries performed by the algorithm on the given input, and is thus our measure for its cost. In other words, the worst-case complexity of the algorithm, in this model, is the maximum depth of its corresponding tree. As in [7], when $s = n$ (the maximum possible value for s), we refer to the model just as a "linear decision tree". The study in this paper will only consider this unconstrained case.

We also note that, although we could in principle construct the whole tree T in a preprocessing stage, the algorithm that we present only constructs, on the fly, the search path that $\mathbf{x}$ traces in T.

To recap, solving an instance of k-SUM, with input $\mathbf{x}$, in this model amounts to processing a sequence of *linear queries* of the form "Does $\mathbf{x}$ lie on some hyperplane h, or else on which side of h does it lie?". Each such query is a sign test, asking for the sign of $h(\mathbf{x})$, where $h(\cdot)$ is the linear expression defining h. Some of the hyperplanes h that participate in these queries will be original hyperplanes of H, but others will be auxiliary hyperplanes that the algorithm constructs. The algorithm succeeds if at least one of the linear queries that involves

[1] We emphasize that in the k-LDT problem, the coefficients $a_0, a_1, \ldots, a_k$ are fixed and the input is the point $\mathbf{x}$.

an original hyperplane (or, during the recursion, a lower-dimensional hyperplane that is contained in an original hyperplane) results in an equality, determining that $\mathbf{x}$ does lie on a hyperplane of H.

Previous work. The k-SUM problem is a variant of the SUBSETSUM problem,[2] and is therefore NP-complete (when k is part of the input); however, its behavior as a function of k (say, in the standard RAM model) has not yet been fully resolved. Specifically, Erickson [15] showed that the k-SUM problem can be solved (in the RAM model) in time $O((2n/k)^{\lceil k/2\rceil})$ for k odd, and $O((2n/k)^{k/2}\log(n/k))$ for k even. Moreover, he presented a nearly-tight lower bound of $\Omega((n/k^k)^{\lceil k/2\rceil})$ for k-SUM in the k-linear decision tree model (see [2] for a more comprehensive overview of Erickson's result). Ailon and Chazelle [2] slightly improved Erickson's lower bound, and extended it to the s-linear decision tree model, where $s > k$, showing a lower bound of $\Omega\left((nk^{-3})^{\frac{2k-s}{2\lceil(s-k+1)/2\rceil}(1-\varepsilon_k)}\right)$, where $\varepsilon_k > 0$ tends to 0 as k goes to ∞. As stated in [2], in spite of the strength of this latter lower bound, it is not very informative for $s \geq 2k$. In particular, when s is arbitrarily large (the case studied in this paper), one can no longer derive a lower bound of the form $n^{\Omega(k)}$. Indeed, Meyer auf der Heide [29] showed an upper bound of $O(n^4 \log n)$ on the number of linear queries for the n-dimensional KNAPSACK problem[3] (and thus, in particular, for k-SUM). Meiser [28] presented an efficient point-location mechanism for high-dimensional hyperplane arrangements, in the standard real RAM model. When interpreted in the linear decision tree model, and applied to the instances at hand, it yields a linear decision tree for k-SUM (as well as for the more general problem k-LDT) of depth that is only polynomial[4] in k and n. Cardinal *et al.* [7] improved this bound[5] to $O(n^3 \log^3 n)$. Concerning lower bounds in this model of computation, Dobkin and Lipton [14] showed a lower bound of $\Omega(n \log n)$ on the depth of the linear decision tree for k-LDT, and, in another paper [13], a lower bound of $\Omega(n^2)$ for the n-dimensional KNAPSACK problem. See also [5, 31] for more general non-linear decision tree models of computation.

The case $k = 3$, i.e., the 3SUM-problem, is related to various geometric problems to which it can be reduced. These problems are known as 3SUM-*hard*. These include problems such as testing whether there exist three collinear points in a given planar set of n points, testing whether the union of n given triangles in the plane covers the unit square, or just testing whether the union has any holes (i.e., is not simply connected), checking for polygon containment under translation, visibility among triangles in 3-space, planar motion planning (under translations and rotations), translational motion planning in 3-space, maximum depth in an arrangement of disks, and more. The study of 3SUM-hard problems was pioneered in the seminal work of Gajentaan and Overmars [18], who showed subquadratic reductions from 3-SUM to many of these problems; see also Barequet and Har-Peled [4] and Aronov and Har-Peled [3] for several additional reductions. During the last two decades, the prevailing conjecture was that any algorithm for 3-SUM requires $\Omega(n^2)$ time.[6] In a recent dramatic

[2] In this problem, given n real numbers, we want to determine whether there is a subset of them that sums to 0.

[3] This problem is an extension of SUBSETSUM, and asks, given n real numbers, whether there is a subset of them that sums to 1.

[4] The original analysis of Meiser was sketchy and relied on a suboptimal choice of parameters. Meiser's analysis has been somewhat tightened by Liu [26]. A careful and meticulous study of Meiser's algorithm, with improved performance bounds, is given in the full version (written with Har-Peled and Kaplan) [16].

[5] We note however that the bound in [7] only applies to instances of k-LDT where all the coefficients are *rational*.

[6] In fact, before the term "3SUM-hard" was coined, these problems were referred to as "n^2-hard" problems [18].

development, this has been refuted by Grønlund and Pettie [20], who presented a (slightly) subquadratic algorithm for 3-SUM (see also the more recent work of Chan and Lewenstein [8], as well as those of Gold and Sharir [19] and of Freund [17]). Furthermore, Grønlund and Pettie showed that in the $(2k-2)$-linear decision tree model, only $O(n^{k/2}\sqrt{\log n})$ queries are required for k odd. In particular, for 3SUM, this bound is $O(n^{3/2}\sqrt{\log n})$. More recently, this bound has further been improved by Gold and Sharir [19] to $O(n^{3/2})$, or, more generally, to $O(n^{k/2})$ for arbitrary k, under a *randomized* $(2k-2)$-linear decision tree model. Note that in all these cases, the best known lower bound is just the standard $\Omega(n \log n)$ bound, and closing the gap between this bound and the aforementioned upper bounds still remains elusive.

Our result. Our main result is an improvement by (more than) a factor of n over the recent bound of Cardinal *et al.* [7] on the complexity of a linear decision tree for k-SUM and k-LDT. Specifically, we show:

▶ **Theorem 3.** *For any fixed k, the complexity of k-SUM and k-LDT in the linear decision-tree model is $O(n^2 \log^2 n)$, where the constant of proportionality is linear in k.*

In fact, the actual bound in Theorem 3 is $O(n^2 \log n \log |H|)$. We can apply this bound to the n-dimensional KNAPSACK problem, in which the relevant hyperplanes are of the form $x_{i_1} + \cdots + x_{i_k} = 1$, for all the $2^n - 1$ possible nonempty subsets $\{i_1, \ldots, i_k\}$ of $\{1, \ldots, n\}$. Taking then $|H| = 2^n - 1$, we obtain the following corollary of Theorem 3, which improves the previous bounds in [28, 29].

▶ **Corollary 4.** *The complexity of the n-dimensional* SUBSETSUM *and* KNAPSACK *problems in the linear decision-tree model is $O(n^3 \log n)$.*

We note that the two bounds that we obtain in Theorem 3 and Corollary 4 are larger by only a factor of $O(n \log n)$ from the respective lower bounds of Dobkin and Lipton [13, 14].

On the "real" algorithmic front, we show that in the RAM model the k-SUM and k-LDT problems can be solved in expected polynomial time, with the same number, $O(n^2 \log^2 n)$, of linear queries (and with all other operations independent of the actual coordinates of $\mathbf{x}$), as in Theorem 3 (the description of the algorithm is deferred to the full version of this paper).

Our analysis uses a variant of the approach in [7], inspired by the point-location mechanism of Meiser [28], where we locate the input point $\mathbf{x}$ in $\mathcal{A}(H)$ using a recursive algorithm that exploits and locally simulates the construction of (a specific kind of) an *ε-cutting* of $\mathcal{A}(H)$. While this framework is not new, a major difference between the construction of [7] and ours is that the former construction applies *bottom-vertex triangulation* to the cells in arrangements of suitable subsets of H, which partitions each cell into simplices. Since the ambient dimension is n, each simplex is defined (in general) by $\Theta(n^2)$ hyperplanes of H; see, e.g., [1, 12] and below. In contrast, in our construction we partition the cells of such arrangements using the *vertical decomposition* technique [1, 9], where each cell of the arrangement is partitioned into a special kind of *vertical prisms*, each of which is defined by only at most $2n$ hyperplanes of H. In both studies, ours and that of Cardinal *et al.* [7], the local construction of the cell containing $\mathbf{x}$ (in an arrangement of some subsample of H) is carried out through n recursive steps (reducing the dimension by 1 at each step). The difference is that the algorithm in [7] needs to perform roughly quadratically many queries at each such recursive step, whereas our algorithm performs only nearly linearly many queries. With a few additional observations about the structure of vertical decompositions (see below for a detailed discussion), this will eventually decrease the overall depth of the linear decision tree by (slightly more than)

a factor of n, with respect to the bound in [7]. We note that, although the combinatorial bound on the overall complexity of bottom vertex triangulations in hyperplane arrangements is in general smaller than the currently best known bound on the complexity of vertical decompositions (in dimensions $d \geq 5$), this is not an issue in the decision tree model. In other words, for the purpose of locating the cell containing $\mathbf{x}$, in the (linear) decision tree model, using vertical decompositions is the decisive winning strategy.

A note on vertical decomposition. As a by-product of this study, our analysis leads to some new insights concerning the structure and complexity of vertical decompositions of arrangements of hyperplanes, including a sharper bound on the complexity of such a decomposition in high dimensions. Specifically, it follows from the study of Chazelle *et al.* [9], or rather from its extension by Koltun [25], that this bound is $O(n^{2d-4})$, where the coefficient of proportionality is $2^{O(d^2)}$. We improve this coefficient to $2^{O(d)}$ (Theorem 5), using a simple but crucial observation about the structure of vertical decompositions, given in Lemma 8. This property is also used by our algorithm to efficiently construct the (prism-like) cell containing the query point $\mathbf{x}$. This improvement is significant when d is not assumed to be a constant (as in the cases, studied here, of the k-SUM and k-LDT problems, and in the cases of the SubsetSum and Knapsack problems). Moreover, this improvement (from $2^{O(d^2)}$ to $2^{O(d)}$) is crucial for obtaining ε-cuttings with samples of size that is only (nearly) *linear* in d. More details are given later in the paper. We believe these results to be of independent interest, and we hope that these insights will lead to further improved bounds on the complexity of vertical decompositions and for additional useful structural properties and further applications of this construct. A more thorough and detailed analysis of these issues is given in the full version [16].

2 Preliminaries: Arrangements and Vertical Decomposition

Let H be a collection of n hyperplanes in $\mathbb{R}^d$ (observe that the notation in this section is different, as it caters to any collection of hyperplanes in any dimension). We emphasize that H is not necessarily in general position, and that it may contain vertical hyperplanes (as it does in the case of k-SUM). The *vertical decomposition* $\mathcal{V}(H)$ of the arrangement $\mathcal{A}(H)$ is defined in the following recursive manner (see [1, 9] for the general setup, and [21, 25] for the case of hyperplanes in four dimensions). Let the coordinate system be $x_1, x_2, \ldots, x_d$, and let C be a cell in $\mathcal{A}(H)$. For each $(d-2)$-face g on ∂C, we erect a $(d-1)$-dimensional *vertical wall* passing through g and confined to C; this is the union of all the maximal x_d-vertical line-segments that have one endpoint on g and are contained in C. The walls extend downwards (resp., upwards) from faces g on the top boundary (resp., bottom boundary) of C (faces on the "equator" of C, i.e., faces that have a vertical supporting hyperplane, have no wall (within C) erected from them). Note that if g lies on a vertical hyperplane $h \in H$, the vertical wall is contained in h. This collection of walls subdivides C into convex vertical prisms, each of which is bounded by (potentially many) vertical walls, and by two hyperplanes of H, one appearing on the bottom portion and one on the top portion of ∂C, referred to as the *floor* and the *ceiling* of the prism, respectively; in case C is unbounded, a prism may be bounded by just a single (floor or ceiling) hyperplane of H. In rare situations, where all the hyperplanes of H are vertical, prisms have neither a floor nor a ceiling. (Note that, by construction, a floor (resp., a ceiling) of a prism cannot be contained in a vertical hyperplane of H.) More formally, this step is accomplished by projecting the bottom and the top portions of ∂C onto the hyperplane $x_d = 0$, and by constructing the *overlay* of these

two convex subdivisions. Each full-dimensional (i.e., $(d-1)$-dimensional) cell in the overlay, when lifted vertically back to $\mathbb{R}^d$ and intersected with C, becomes one of the above prisms.

Note that after this step, the two bases (or the single base, in case the prism is unbounded) of a prism may have arbitrarily large complexity, or, more precisely, be bounded by arbitrarily many hyperplanes. Each base, say the floor base, is a convex polyhedron in $\mathbb{R}^{d-1}$, namely in the hyperplane h^- containing it, bounded by at most $2n-1$ hyperplanes (of dimension $d-2$), where each such hyperplane is either an intersection of h^- with another original hyperplane h, or the vertical projection onto h^- of an intersection of the corresponding ceiling hyperplane h^+ with some other h (for h vertical, the two cases coincide; that is, they yield the same $(d-2)$-hyperplane within h^-); this collection might also include $h^- \cap h^+$. In what follows we refer to these prisms as *undecomposed prisms*, or *first-stage prisms*. Our goal is to decimate the dependence of the complexity of the prisms on n, and to construct a decomposition of this kind so that each of its prisms is bounded by no more than $2d$ hyperplanes. To do so, we recurse with the construction at each base of each prism, or rather, for simplicity, within the common projection of the bases onto $x_d = 0$. Each recursive subproblem is now $(d-1)$-dimensional.

Specifically, after the first decomposition step described above, we project each of the first-stage prisms just obtained onto the hyperplane $x_d = 0$, obtaining a $(d-1)$-dimensional convex polyhedron C', which we vertically decompose using the same procedure described above, only in one lower dimension. That is, we now erect vertical walls within C' from each $(d-3)$-face of $\partial C'$, in the x_{d-1}-direction. These walls subdivide C' into x_{d-1}-vertical (undecomposed) prisms, each of which is bounded by (at most) two facets of C', which form its floor and ceiling (in the x_{d-1}-direction), and by some of the vertical walls. We keep projecting these prisms onto hyperplanes of lower dimensions, and produce the appropriate vertical walls. We stop the recursion as soon as we reach a one-dimensional instance, in which case all prisms projected from previous steps become line-segments, requiring no further decomposition.[7] We now backtrack, and lift the vertical walls (constructed in lower dimensions, over all iterations), one dimension at a time, ending up with $(d-1)$-dimensional walls within the original cell C; that is, a $(d-i)$-dimensional wall is "stretched" in directions $x_{d-i+2}, \ldots, x_d$ (applied in that order), for every $i = d, \ldots, 2$.

Each of the final cells is a "box-like" prism, bounded by at most $2d$ hyperplanes. Of these, two are original hyperplanes, two are hyperplanes supporting two x_d-vertical walls erected from some $(d-2)$-faces, two are hyperplanes supporting two $x_{d-1}x_d$-vertical walls erected from some $(d-3)$-faces (within the appropriate lower-dimensional subspaces), and so on. Note that since we do not assume general position, some of these vertical walls may be original hyperplanes of H (this issue is discussed in more detail later on).

We note that each final prism is *defined* in terms of at most $2d$ original hyperplanes of H, in a sense made precise in the ensuing description. We establish this property using backward induction on the dimension of the recursive instance. Initially, we have two original hyperplanes h^-, h^+, which contain the floor and ceiling of the prism, respectively. We intersect each of them with the remaining hyperplanes of H (including the intersection $h^- \cap h^+$), and project all these intersections onto the $(d-1)$-hyperplane $x_d = 0$. Suppose inductively that, when we are at dimension j, we already have a set D_j of (at most) $2(d-j)$ original defining hyperplanes (namely, original hyperplanes defining the walls erected so

[7] If we care about the complexity of the resulting decomposition, in terms of its dependence on n, which is not a crucial issue in our approach, it is better to stop the recursion earlier. The terminal dimension is $d = 2$ or $d = 3$ in [9], and $d = 4$ in [25].

far), and that each (lower-dimensional) hyperplane in the current collection H_j of $(j-1)$-hyperplanes is obtained by an interleaved sequence of intersections and projections, which are expressed in terms of some subset of the $\leq 2(d-j)$ defining hyperplanes and (at most) one additional original hyperplane. Clearly, all this holds trivially in the initial step $j = d$. We now choose a new floor and a new ceiling from among the hyperplanes in H_j, gaining two new defining hyperplanes (the unique ones that define the new floor and ceiling and are the ones not in D_j). We add them to D_j to form D_{j-1}, intersect each of them with the other hyperplanes in H_j, and project all the resulting $(j-2)$-intersections onto the $(j-1)$-hyperplane $x_j = 0$, to obtain a new collection H_{j-1} of $(j-2)$-hyperplanes. Clearly, the inductive properties that we assume carry over to the new sets D_{j-1} and H_{j-1}, so this holds for the final setup in $d = 1$ dimensions. Since each step adds at most two new defining hyperplanes, the claim follows.

We apply this recursive decomposition for each cell C of $\mathcal{A}(H)$, and thereby obtain the entire vertical decomposition $\mathcal{V}(H)$. We remark though that our algorithm does not explicitly construct $\mathcal{V}(H)$. In fact, it does not even construct the (full discrete representation of the) prism of $\mathcal{V}(H)$ containing the query point $\mathbf{x}$. It will be clear shortly from the presentation what the algorithm actually constructs. The description given above, while being constructive, is made only to define the relevant notions, and to set the infrastructure within which our algorithm will operate.

3 ε-Cuttings from Vertical Decompositions

Given a finite collection H of hyperplanes in $\mathbb{R}^d$, by an *ε-cutting* for H we mean a subdivision of space into prism-like cells, of the form just defined, that we simply refer to as prisms[8], such that every cell is *crossed* by (i.e., the interior of the cell is intersected by) at most $\varepsilon|H|$ hyperplanes of H, where $0 < \varepsilon < 1$ is the parameter of the cutting. ε-cuttings are a major tool for a variety of applications, including our own; they have been established and developed in several fundamental studies [10, 11, 27].

Roughly speaking, when d is a (small) constant, the random sampling theory of Clarkson [11] (see also Clarkson and Shor [12]) produces an ε-cutting as follows. We draw[9] a random sample R of $\frac{c_d}{\varepsilon}\log\frac{d}{\varepsilon}$ hyperplanes from H, where c_d is a parameter that depends only on d; its actual dependence on d becomes a major issue in the analysis when d is large. We then construct the arrangement $\mathcal{A}(R)$ of R and its vertical decomposition $\mathcal{V}(R)$. With a suitable choice of c_d, the random sampling technique of Clarkson [11] then guarantees, with constant (high) probability, that each prism of $\mathcal{V}(R)$ is crossed by at most $\varepsilon|H|$ hyperplanes of H. (This also follows from the ε-net theory of Haussler and Welzl [23], but, as it turns out, the coefficient c_d has to be much larger when d is large; see below and [16] for more details.)

3.1 The Clarkson Framework

We keep denoting by H a set of n hyperplanes in $\mathbb{R}^d$. Following the definitions and notations in [22, Chapter 8], put $\mathcal{T} = \mathcal{T}(H) := \bigcup_{S\subseteq H} \mathcal{V}(S)$; that is, $\mathcal{T}$ is the set of all possible prisms

[8] In the original studies (see, e.g., [10]), these subcells were taken to be simplices, although both forms have been used in the literature by now.

[9] We use an alternative drawing mode, in which, to get a sample of size r, we sample each element of H independently with probability $p = r/|H|$. The size of the sample is r only in expectation, but this does not affect (in fact, it simplifies) the overall analysis; see, e.g., [30].

defined by the subsets of H. For each prism $\tau \in \mathcal{T}$, we associate with τ its *defining set* $D(\tau)$ and its *conflict set* $K(\tau)$. The former is the smallest subset $D \subseteq H$ such that τ is a prism in $\mathcal{V}(D)$, and the latter is the set of all hyperplanes $h \in H$ for which τ does not appear in $\mathcal{V}(D \cup \{h\})$; they are precisely the hyperplanes in $H \setminus D$ that cross τ. By our discussion in Section 2 we have $|D(\tau)| \leq 2d$, for each $\tau \in \mathcal{T}$.

We have the following two axioms, which hold for any subset $S \subseteq H$:

(i) For any $\tau \in \mathcal{V}(S)$, we have $D(\tau) \subseteq S$ and $K(\tau) \cap S = \emptyset$.

(ii) If $D(\tau) \subseteq S$ and $K(\tau) \cap S = \emptyset$, then $\tau \in \mathcal{V}(S)$.

A key novel property of vertical decompositions, which we establish in this paper, is the following result (some highlights of whose proof are given at the end of this section):

▶ **Theorem 5.** *Let H be a set of n hyperplanes in $\mathbb{R}^d$. Then the cardinality of $\mathcal{T}(H)$ is at most $O\left(\frac{2^{2d}}{d^{7/2}} n^{2d}\right)$.*

In particular, we get a sharper bound on the complexity of vertical decompositions:

▶ **Corollary 6.** *Let H be a set of n hyperplanes in $\mathbb{R}^d$. Then the number of prisms in $\mathcal{V}(H)$ is at most $O\left(\frac{2^{2d}}{d^{7/2}} n^{2d}\right)$.*

▶ Remark. As already mentioned in the introduction, the bound in Corollary 6 significantly improves the previous upper bound of [9] in terms of its dependence on d, in that its "constant" of proportionality drops from $2^{O(d^2)}$ to less than 4^d. We pay a small price (it is small unless n is huge relative to d) in terms of the dependence on n, which is n^{2d} in the new bound, instead of n^{2d-4} in [25] (and only $O(n^d)$ if one uses instead bottom-vertex triangulation). See below for an additional discussion of this issue.

Equipped with Theorem 5, we obtain (we omit the standard proof, which follows the analysis in [11, 12] [10]):

▶ **Theorem 7.** *Given a set H of n hyperplanes in d-space, and a parameter $\varepsilon \in (0, 1)$, a random sample R of $O\left(\frac{d}{\varepsilon} \log \frac{d}{\varepsilon}\right)$ hyperplanes of H (with an appropriate absolute constant of proportionality) satisfies, with constant probability, the property that each prism in the vertical decomposition $\mathcal{V}(R)$ of $\mathcal{A}(R)$ is crossed by at most $\varepsilon|H|$ hyperplanes of H.*

▶ Remark.

1. To turn this random sampling into a procedure that generates an ε-cutting almost surely, we draw a random sample R of the aforementioned (expected) size, and test whether it satisfies the property asserted in Theorem 7. If not, we simply discard this sample and repeat the construction with a new sample. Clearly, since we fail with constant probability (which we can make rather small by increasing the constant of proportionality), the expected number of trials till a successful sample is drawn is constant (close to 1).
2. Our construction uses vertical decomposition. Expanding upon an earlier made comment, we note that an alternative construction, for arrangements of hyperplanes, is the *bottom-vertex triangulation* (see [1]). It has the advantage, over vertical decomposition, that the (bound on the) number of cells (simplices) that it produces is significantly smaller (at most $|R|^d$), but its major disadvantage for the analysis in this paper is that the typical size of a defining set of a simplex in this decomposition is $d_0 = d(d+3)/2$, as opposed to the much smaller value $d_0 = 2d$ for vertical decomposition; see above, [1], and the

[10] It is important to notice that we can follow this analysis because the coefficient is only singly exponential in d.

full version [16], for more details. The fact that prisms in the vertical decomposition have such a smaller bound on the size of their defining sets, combined with our improved bound on the complexity of vertical decomposition, is what makes vertical decomposition a superior technique for the (decision-tree complexity of the) k-SUM problem.

3. We also remark that the method that we use here is not optimal, from a general perspective, in several aspects: First, it does not involve the refining second resampling stage of Chazelle and Friedman [10] (and of others), which leads to a slight improvement in the number of cells (or, alternatively, to a smaller required sample size). More significantly, in $d \geq 5$ dimensions there are no sharp known bounds on the complexity of the vertical decomposition, even for arrangements of hyperplanes (see [9, 24] for the general case, and [21, 25] for the case of hyperplanes). Nevertheless, these issues are irrelevant for the technique employed here (mainly because, as already mentioned, we will not construct the entire vertical decomposition), and the coarser method reviewed above serves our purposes just fine.
4. Finally, we note that, in principle, we could have also used the ε-net theory of [23] to ensure the ε-cutting property of the resulting decomposition. However, the VC-dimension of the suitably defined corresponding range space is much larger than $2d$. Concretely, it follows from the analysis in the full version [16] that the VC-dimension is $O(d^3)$ and $\Omega(d^2)$. Since the size of the random sample in the theory in [23] has to be (slightly more than) proportional to the VC-dimension, this approach results in much poorer bounds, which will cause our algorithm to be at least as slow as the one in [7].

3.2 Key Properties in the Proof of Theorem 5

Following the presentation in Section 2, we first analyze the complexity of the vertical decomposition of a single cell of $\mathcal{A}(H)$, and then derive a global bound for the entire arrangement. Due to lack of space, we only present here a key property of the analysis, which is also crucial for the analysis of our k-SUM algorithm presented in Section 4.

Let C be a fixed cell of $\mathcal{A}(H)$. With a slight abuse of notation, denote by n the number of its facets (that is, the number of hyperplanes of H that actually appear on its boundary), and consider the procedure of constructing its vertical decomposition, as described in Section 2. As we recall, the first stage produces vertical prisms, each having a fixed floor and a fixed ceiling. We take each such prism, whose ceiling and floor are contained in two respective hyperplanes h_1, h_2 of H, project it onto the hyperplane $x_d = 0$, and decompose the projection C_{d-1} recursively.

The $(d-2)$-hyperplanes that bound C_{d-1} are projections of intersections of the form $h \cap h_1$, $h \cap h_2$, for $h \in H \setminus \{h_1, h_2\}$, including also $h_1 \cap h_2$, if it arises. In principle, the number of such hyperplanes is at most $2n-1$, but, as shown in the following lemma the actual number is smaller:

▶ **Lemma 8.** *Let τ be a first-stage prism, whose ceiling and floor are contained in two respective hyperplanes h_1, h_2. Then for each hyperplane $h \in H$, $h \neq h_1, h_2$, the following holds.*

(a) *If h is nonvertical then only one of $g_1 := h_1 \cap h$ or $g_2 := h_2 \cap h$ can appear on $\partial\tau$. It is g_1 if C lies below h, and g_2 if C lies above h.*

(b) *If h is vertical, both g_1 and g_2 can appear on $\partial\tau$, but their projections onto $x_d = 0$ coincide.*

Proof.

(a) Assume that h is nonvertical. Then either C lies fully above h or it lies fully below h. Without loss of generality, assume that the former case holds. Since C lies below h_1, the intersection $g_1 = h \cap h_1$, if it shows up on ∂C at all, must bound an equator facet φ of C. If φ appears on $\partial\tau$, then the interior of τ must contain a vertical segment whose endpoints lie on h (bottom) and on h_1 (top), contradicting the fact that the floor of τ lies on h_2. Hence only g_2 can appear on $\partial\tau$. The case where C lies below h is handled symmetrically.

The proof of (b) is straightforward; it follows from the fact that h_1 and h_2 are nonvertical, and in fact both projections of g_1 and g_2 coincide with that of the entire h. ◀

▶ Remark. An important feature of the proof is that it also holds when τ is *any* convex vertical prism, obtained at any recursive step of the decomposition, and, in particular, when τ is the vertical prism obtained at the final step.

It is straightforward to verify that Lemma 8 implies that the projection of τ onto $x_d = 0$ has at most $n-1$ facets. Using this property we derive a recurrence relation to bound the complexity of the vertical decomposition of a single cell C, and then, using axioms (i)–(ii), we obtain a bound for the entire arrangement. These details appear in the full paper [16].

4 The Algorithm

4.1 Algorithm outline

The high-level approach of our algorithm can be regarded as an optimized variant of the algorithm of Cardinal *et al.* [7], which is inspired by the point-location mechanism of Meiser [28]. We choose $\varepsilon > 0$ to be a constant, smaller than, say, $1/2$, and apply the ε-cutting machinery, as reviewed in Theorem 7. For a given input point $\mathbf{x}$, the algorithm proceeds as follows.

- **(i)** Construct a random sample R of $r := O\left(\frac{n}{\varepsilon}\log\frac{n}{\varepsilon}\right) = O(n\log n)$ hyperplanes of H, with a suitable absolute constant of proportionality (recall that in our application, the dimension of the underlying space is n). If R violates the ε-cutting property asserted in Theorem 7, discard R and repeat the process with a new sample.
- **(ii)** Construct the prism $\tau = \tau_{\mathbf{x}}$ of $\mathcal{V}(R)$ that contains the input point $\mathbf{x}$. If at that step we detect an original hyperplane of H that contains $\mathbf{x}$, we stop and return "YES".
- **(iii)** Construct the conflict list $CL(\tau)$ of τ (the subset of hyperplanes of H that cross τ), and recurse on it.
- **(iv)** Stop as soon as $|CL(\tau)|$ is smaller than the sample size $r = O\left(\frac{n}{\varepsilon}\log\frac{n}{\varepsilon}\right)$ (we use the same sample size in all recursive steps). When that happens, test $\mathbf{x}$, in brute force, against each original hyperplane of H in $CL(\tau)$; return "YES" if one of the tests results in an equality, and "NO" otherwise.

We note that those parts of the algorithm that do not depend[11] on $\mathbf{x}$, which are costly in the RAM model, are performed here for free. That is, our goal at this point is only to bound the number of linear queries performed by the algorithm. We also note that although the construction of the prism containing $\mathbf{x}$ (described below) is conceptually simple, it involves several technical details, which mainly follow from the fact that H may contain vertical hyperplanes (vs. the simpler scenario where all hyperplanes are in general position).

[11] By this we mean that they do not compute any explicit expression that depends on the coordinates of $\mathbf{x}$. They might (and in general, will) depend on previously computed discrete data that does depend on $\mathbf{x}$, but accessing this data in our model, once computed, is for free.

We next describe the details of implementing step (ii). We comment that step (i) costs nothing in our model, so we do not bother with its implementation details. The tests in step (iii), although being very costly in the "full" standard RAM model of computation, are independent of the specific coordinates of $\mathbf{x}$, and thus cost nothing in our model. We present this step in detail in the full version of this paper.

Constructing the prism containing $\mathbf{x}$. Since the overall complexity of a prism (the number of its faces of all dimensions) is exponential in the dimension n, we do not construct it explicitly. Instead we only construct explicitly its at most $2n$ bounding hyperplanes, consisting of a floor and a ceiling (or only one of them in case the cell $C_{\mathbf{x}}$ in $\mathcal{A}(R)$ containing $\mathbf{x}$ is unbounded), and at most $2n-2$ vertical walls (we have strictly fewer than $2n-2$ walls in cases when either the floor of τ intersects its ceiling (on $\partial\tau$), or when this happens in any of the projections τ^* of τ in lower dimensions, or when the current subcell becomes unbounded at any of the recursive steps). The prism τ, as defined in step (ii), is then implicitly represented as the intersection of the halfspaces bounded by these hyperplanes and containing $\mathbf{x}$. Let $H_{\mathbf{x}}$ denote this set of at most $2n$ hyperplanes. From now on we assume, to simplify the presentation but without loss of generality, that $C_{\mathbf{x}}$ is bounded, and that τ has exactly $2n-2$ vertical walls (and thus exactly $2n$ bounding hyperplanes).

The following recursive algorithm constructs $H_{\mathbf{x}}$, and also detects whether $\mathbf{x}$ lies on one of the bounding hyperplanes of τ. Let $r = O\left(\frac{n}{\varepsilon}\log\frac{n}{\varepsilon}\right)$ denote the (expected) size of our sample R. Initially, we set $H_{\mathbf{x}} := \emptyset$. We first perform r linear queries with $\mathbf{x}$ and each of the hyperplanes of R, resulting in a sequence of r output labels "above" / "below" / "sideways" / "on". At the top level of recursion (before reducing the dimension), encountering a label "on" means that $\mathbf{x}$ lies on an original hyperplane of H, and thus there is a positive solution to our instance of k-SUM, and we stop the entire procedure and output "YES". At deeper recursive levels (in lower-dimensional spaces), when we encounter "on", we need to check that the relevant (now lower-dimensional) hyperplane is fully contained in an original hyperplane of H, in order to output "YES" (the full containment condition is addressed later on). As will be discussed below, such a hyperplane does not have to belong to R, so the procedure for performing this test, and in particular its analysis, is rather elaborate.

We thus assume, without loss of generality, that all labels are "above", "below", or "sideways". We next partition the set of the hyperplanes in R according to their labels, letting R_1 denote the set of hyperplanes lying above $\mathbf{x}$, R_2 the set of hyperplanes below it, and R_0 the set of vertical hyperplanes to the side of $\mathbf{x}$. We then identify the upper hyperplane $h_1 \in R_1$ and the lower hyperplane $h_2 \in R_2$ with shortest vertical distances from $\mathbf{x}$. We do this by computing the minimum of these vertical distances, each of which is a linear expression in $\mathbf{x}$, using $(|R_1|-1)+(|R_2|-1) < r$ additional comparisons. The hyperplanes h_1 and h_2 contain the ceiling and the floor of τ, respectively, and we thus insert them into $H_{\mathbf{x}}$.

In order to produce the hyperplanes containing the vertical walls of τ, we recurse on the dimension n. This process somewhat imitates the one producing the entire vertical decomposition of $C_{\mathbf{x}}$ described above. However, the challenges in the current construction are to build only the single prism containing $\mathbf{x}$, to keep the representation implicit, and to do this efficiently.

We generate all pairwise intersections $h_1 \cap h$ and $h_2 \cap h$, for $h \in R$, $h \neq h_1$, $h \neq h_2$, and obtain two collections G_1, G_2 of $(n-2)$-dimensional flats, each of size at most $r-2$, which we project onto the hyperplane $x_n = 0$.

By Lemma 8 (when the input set is now R) and the remark after it, the following holds for each $h \in R \setminus \{h_1, h_2\}$. Put $g_1 := h_1 \cap h$ and $g_2 := h_2 \cap h$. (a) If h is nonvertical then

at most one of g_1, g_2 can appear on $\partial\tau$. (b) If h is vertical, the projections of g_1 and g_2 coincide, and are in fact equal to the projection of the entire h. We can therefore discard one of g_1, g_2 when h is nonvertical (using the simple rule in Lemma 8), and replace both by the single projection of h, when h is vertical. Hence, the subset $G \subseteq G_1 \cup G_2$ of the surviving intersections consists of at most $|R| - 2$ flats (of dimension $n-2$). We denote by $R^{(1)}$ the set of their projections onto the hyperplane $x_n = 0$. (If $h_1 \cap h_2$ is also relevant, we add its projection to $R^{(1)}$, making its size go up to $|R| - 1$.)

We continue the construction recursively on $R^{(1)}$ in $n-1$ dimensions. That is, at the second iteration, we project $\mathbf{x}$ onto the subspace $x_n = 0$; let $\mathbf{x}^{(1)}$ be the resulting point. We first perform at most r linear tests with $\mathbf{x}^{(1)}$ and each of the hyperplanes in $R^{(1)}$. If we encounter "on" for some $h^{(1)} \in R^{(1)}$ then $\mathbf{x}$ lies on a vertical wall of τ passing through $h^{(1)}$. If $h^{(1)}$ is fully contained in a (vertical) hyperplane $h' \in H$, we output "YES". We emphasize that h' does not have to belong to R (see a discussion of this issue in the proof of correctness, given below), so, to determine whether such an h' exists, we simply test $h^{(1)}$ against all hyperplanes of H (which costs nothing in our model). Otherwise, if we encountered "on" for some $h^{(1)} \in R^{(1)}$ in the above test, but $h^{(1)}$ is not contained in any vertical hyperplane of H, then the prism τ containing $\mathbf{x}$ (in the original n-space) is of one lower dimension (or, alternatively, $\mathbf{x}$ lies on a facet of a full-dimensional prism, which projects to a portion of $h^{(1)}$). In this case, we can intersect all the remaining hyperplanes in $R^{(1)}$ with $h^{(1)}$, projecting the whole setting to the hyperplane $x_{n-1} = 0$, and continue the construction recursively within that hyperplane.

The general flow of the recursive procedure is as follows. At each step i, for $i = 1, 2, \ldots, n$, we have a collection $R^{(i-1)}$ of at most $|R|$ hyperplanes of dimension $n-i$, and a point $\mathbf{x}^{(i-1)}$, in the $x_1 \cdots x_{n-i+1}$-hyperplane (for $i = 1$ we have $R^{(0)} = R$ and $\mathbf{x}^{(0)} = \mathbf{x}$). We first test whether $\mathbf{x}^{(i-1)}$ lies on any of the hyperplanes $h^{(i-1)}$ in $R^{(i-1)}$. If so, we test whether $h^{(i-1)}$ is contained in an original hyperplane of H (essentially[12] a hyperplane that is parallel to all the coordinates $x_{n-i+2}, \ldots, x_n$ that we have already processed, that is, vertical in all of them), and, if so, we output "YES". If $\mathbf{x}^{(i-1)}$ lies on some $h^{(i-1)}$ (but no original hyperplane of H contains $h^{(i-1)}$), we recurse in one lower dimension, as described for the case $i = 2$. Otherwise, we assume, without loss of generality, that no "on" label is produced. We find the pair of hyperplanes that lie respectively above and below $\mathbf{x}^{(i-1)}$ in the x_{n-i+1}-direction, and are closest to $\mathbf{x}^{(i-1)}$ in that direction (they support the "ceiling" and "floor" of the recursive prism, in the x_{n-i}-direction), and then produce a set $R^{(i)}$ of fewer than $|R|$ hyperplanes of dimension $(n-i-1)$ in the $x_1 \cdots x_{n-i}$-hyperplane. We also project $\mathbf{x}^{(i-1)}$ onto this hyperplane, thereby obtaining the next point $\mathbf{x}^{(i)}$. The construction of $R^{(i)}$ is performed similarly to the way it is done in case $i = 1$, described above, and Lemma 8 (and the remark following it) continues to apply, so as to ensure that indeed $|R^{(i)}|$ continues to be (progressively) smaller than $|R|$, and that its members are easy to construct.

We stop when we reach $i = n$, in which case we are given a set of at most $|R|$ points on the real line, and we locate the two closest points to the final projected point $\mathbf{x}^{(n)}$.

To complete the construction, we take each of the hyperplanes $h_1^{(i-1)}$, $h_2^{(i-1)}$, obtained at each of the iterations $i = 2, \ldots, n$, and lift it "vertically" in all the remaining directions $x_{n-i+2}, \ldots, x_n$, and add the resulting $(n-1)$-hyperplanes in $\mathbb{R}^n$ to $H_{\mathbf{x}}$. We comment that in case $\mathbf{x}$ lies on a facet (or, more generally, a lower dimensional face) of τ, we need to confine these liftings to the appropriate flat h containing this face(t).

[12] To be precise, it could also be that, accidentally, some other original hyperplane contains $h^{(i-1)}$.

▶ Remark. The importance of Lemma 8 is that it controls the sizes of the sets $R^{(i)}$, $i \geq 1$. Without the filtering that it provides, the size of each $R^{(i)}$ would be roughly twice the size of $R^{(i-1)}$, and the query would then require exponentially many linear tests. This doubling effect shows up in the original analysis of the complexity of vertical decompositions [9].

Algorithm correctness. Let h be a hyperplane of H that contains $\mathbf{x}$. Clearly, h must intersect the interior or the boundary of τ. In the former case, h belongs to $CL(\tau)$, and will be passed down the recursion. In the latter case, either h is the floor or ceiling of τ, and then the first stage of constructing τ will detect that $\mathbf{x} \in h$. Otherwise h must be an x_d-vertical hyperplane. Indeed, no other original hyperplane can meet τ unless it intersects its floor or ceiling; since $\mathbf{x}$ was found not to lie on the floor or the ceiling, it cannot lie on any nonvertical h. If h is in R, then the algorithm outputs "YES". It is possible, though, that $h \notin R$, in which case, since h does not intersect the interior of τ, it does not belong to $CL(\tau)$, and we risk missing h altogether. (Clarkson's theory, in the context used in this paper, does not control the number of hyperplanes that touch τ without crossing it.) However, if $\mathbf{x}$ does lie on h then $\mathbf{x}$ must lie on $\partial\tau$, and one of the recursive steps in the construction of τ will detect this fact. In the full version of this paper we describe a procedure, already alluded to several times earlier, that tests for this property, and establish its correctness.

We emphasize that at each recursive step we construct the prism τ only with respect to the conflict list of its parent cell τ_0 (initially, $\tau_0 = \mathbb{R}^d$ and $CL(\tau_0) = H$), implying that τ is not necessarily contained in τ_0. In other words, the sequence of cells τ constructed in our algorithm are spatially in no particular relation to one another (except that all of them contain $\mathbf{x}$). Still, this does not harm the correctness of the search process, a claim that is argued as follows. The fact that $\mathbf{x}$ lies in τ_0 and in τ implies that it can only lie either on one of the hyperplanes in $CL(\tau_0)$ or on one of the (at most $2d$) bounding hyperplanes of τ. The latter situation will be detected during the non-recursive processing of τ, during which the algorithm (step (ii)) will test $\mathbf{x}$ against each of the bounding hyperplanes of τ (once again, we describe this in more detail in the full paper). If it finds that $\mathbf{x}$ lies on such a hyperplane, it then determines, as mentioned above, whether that hyperplane is an original hyperplane of H (or, more precisely, of $CL(\tau_0)$). Hence, if $\mathbf{x}$ lies on some hyperplane $h \in H$, and this fact has not yet been detected, h will be passed to the recursion as an element of $CL(\tau)$ at step (iii). The case where τ is lower-dimensional is handled in a similar manner.

We next claim that the algorithm terminates (almost surely). Indeed, at each recursive step, the sample R is drawn from the corresponding subset $CL(\tau_0)$. Applying Theorem 7 to $CL(\tau_0)$, we obtain that the conflict list of the next prism τ contains (with certainty, due to the test applied at step (i)) only at most $\varepsilon|CL(\tau_0)|$ hyperplanes of $CL(\tau_0)$. Hence, with probability 1, after a logarithmic number of steps (see below for the concrete analysis) we will reach step (iv), and then the algorithm will correctly determine whether $\mathbf{x}$ lies on a hyperplane of H (by the invariant that we maintain, any such hyperplane belongs to the final conflict list $CL(\tau)$).

(The termination is guaranteed only almost surely, because of the possibility of the event (that has probability 0) of repeatedly failing to choose a good sample at some recursive application of step (i).)

The query complexity. Due to lack of space, we describe the analysis of the query complexity very briefly, and postpone the remaining details to the full paper. Roughly speaking, at each recursive step in the construction of τ we need to perform $O(r)$ linear queries in order to determine, for each hyperplane $h \in R$ whether it lies above, below, on, or sideways from $\mathbf{x}$,

and then find the ceiling and floor hyperplanes h_1 and h_2. In order to test, for a nonvertical hyperplane $h \in R \setminus \{h_1, h_2\}$, which of $g_1 := h_1 \cap h$ or $g_2 := h_2 \cap h$ can appear on $\partial\tau$ (or, more specifically, on the boundary of the *undecomposed* convex prism $\bar{\tau}$ containing τ), we use the simple rule provided in Lemma 8 (which holds in any dimension $i \leq n$). This eventually implies that we spend a total of $O(n^2 \log n)$ linear queries over all n steps of the recursion (on the dimension), for a grand total of $O(n^2 \log n \log |H|) = O(kn^2 \log^2 n)$ linear queries, over all $O(\log |H|)$ steps of the algorithm.

This completes the proof of Theorem 3 for the k-SUM problem. The analysis proceeds more or less verbatim to the more general case of k-LDT with the same performance bound, and generalizes, also trivially, to the cases of SubsetSum and Knapsack. We omit the easy details in this version. ◀

Concluding remarks and open problems. It looks likely that the number of queries can be brought down to $O(n^2 \log n)$. To fit into the general theory of Clarkson, we have drawn a sample R of size $O(n \log n)$. The logarithmic factor is needed if we want to ensure (with constant, high probability) that the ε-cutting property holds for *all* prisms that arise in $\mathcal{V}(R)$, but we only need this property to hold for the single prism that contains $\mathbf{x}$. With a smaller sample size $O(n)$, and with some extra care, we seem to obtain an expected number of $O(n^2 \log n)$ queries. We plan to present this improvement in the full version [16], where this under-sampling technique is referred to as *optimal sampling*. Applying this improvement to the Knapsack or the SubsetSum problems, the number of queries goes down to $O(n^3)$.

We show in the full version that our algorithm, when cast into the RAM model, has an implementation whose expected running time is $n^{k+O(1)}$ (but still using only $O(n^2 \log^2 n)$ linear queries on $\mathbf{x}$). An interesting open problem is whether the running time can be improved to roughly $O(n^{\lceil k/2 \rceil})$, while still using only nearly-quadratically many linear queries. A similar result was obtained in the previous work of Cardinal *et al.* [7] (but with a nearly-cubic number of linear queries). We hope to obtain a similar improvement for the approach used in this paper.

Acknowledgments. The authors would like to thank Shachar Lovett, Sariel Har-Peled, and Haim Kaplan for many useful discussions.

References

1 P. K. Agarwal and M. Sharir, Arrangements and their applications, In *Handbook of Computational Geometry*, (J. Sack and J. Urrutia, eds.), Elsevier, Amsterdam, pp. 973–1027, 2000.

2 N. Ailon and B. Chazelle, Lower bounds for linear degeneracy testing, *J. ACM*, 52(2):157–171, 2005.

3 B. Aronov and S. Har-Peled, On approximating the depth and related problems, *SIAM J. Comput.* 38:899–921, 2008.

4 G. Barequet and S. Har-Peled, Polygon-containment and translational min-Hausdorff-distance between segment sets are 3SUM-hard, *Int'l J. Comput. Geometry Appl.*, 11(4):465–474, 2001.

5 M. Ben-Or, Lower bounds for algebraic computation trees, In *Proc. 16th Annual ACM Symp. Theory Comput. (STOC)*, pp. 80–86, 1983.

6 R. G. Bland, D. Goldfarb, and M. J. Todd, The ellipsoid method: A survey, *Operations Research*, 29(6):1039–1091, 1981.

7 J. Cardinal, J. Iacono, and A. Ooms, Solving k-SUM using few linear queries, In *Proc. 24th European Symp. Alg. (ESA)*, 2016, 25:1–25:17.

8 T. M. Chan and M. Lewenstein, Clustered integer 3SUM via additive combinatorics, In *Proc. 47th Annual ACM Symp. Theory Comput. (STOC)*, pp. 31–40, 2015.

9 B. Chazelle, H. Edelsbrunner, L. Guibas and M. Sharir, A singly exponential stratification scheme for real semi–algebraic varieties and its applications, *Theoret. Comput. Sci.*, 84:77–105, 1991. Also in *Proc. 16th Int'l Colloq. on Automata, Languages and Programming*, 1989, pp. 179–193.

10 B. Chazelle and J. Friedman, A deterministic view of random sampling and its use in geometry, *Combinatorica*, 10:229–249, 1990.

11 K. L. Clarkson, New applications of random sampling in computational geometry, *Discrete Comput. Geom.*, 2:195–222, 1987.

12 K. L. Clarkson and P. W. Shor, Applications of random sampling in computational geometry, II, *Discrete Comput. Geom.*, 4:387–421, 1989.

13 D. Dobkin and R. Lipton, A lower bound of $n^2/2$ on linear search programs for the Knapsack problem, *J. Comput. Syst. Sci.*, 16(3):413–417, 1978.

14 D. Dobkin and R. Lipton, On the complexity of computations under varying set of primitives, *J. Comput. Syst. Sci.*, 18:86–91, 1979.

15 J. Erickson, Lower bounds for linear satisfiability problems, *Chicago. J. Theoret. Comput. Sci.*, 8:388–395, 1997.

16 E. Ezra, S. Har-Peled, H. Kaplan and M. Sharir, Decomposing arrangements of hyperplanes: VC-dimension, combinatorial dimension, and point location, Manuscript, 2017.

17 A. Freund, Improved Subquadratic 3SUM, *Algorithmica*, 77(2):440–458, 2017.

18 A. Gajentaan and M. H. Overmars, On a class of $O(n^2)$ problems in computational geometry, *Comput. Geom. Theory Appl.*, 5:165–185, 1995.

19 O. Gold and M. Sharir, Improved bounds on 3SUM, k-SUM, and linear degeneracy, *CoRR abs*/1512.05279v2, 2017.

20 A. Grønlund and S. Pettie, Threesomes, degenerates, and love triangles, In *Proc. 55th Annual Symp. Found. Comput. Sci.*, pp. 621–630, 2014.

21 L. J. Guibas, D. Halperin, J. Matoušek, and M. Sharir, On vertical decomposition of arrangements of hyperplanes in four dimensions, *Discrete Comput. Geom.*, 14:113–122, 1995.

22 S. Har-Peled, *Geometric Approximation Algorithms*, Mathematical Surveys and Monographs, Vol. 173, AMS Press, Providence, RI, 2011.

23 D. Haussler and E. Welzl, ε-nets and simplex range queries, *Discrete Comput. Geom.*, 2:127–151, 1987.

24 V. Koltun, Almost tight upper bounds for vertical decompositions in four dimensions, *J. ACM* 51(5):699–730, 2004.

25 V. Koltun, Sharp bounds for vertical decompositions of linear arrangements in four dimensions, *Discrete Comput. Geom.*, 31(3):435–460, 2004.

26 D. Liu, A note on point location in arrangements of hyperplanes, *Inform. Process. Letts.*, 90(2):93–95, 2004.

27 J. Matoušek, Cutting hyperplane arrangements, *Discrete Comput. Geom.*, 6:385–406, 1991.

28 S. Meiser, Point location in arrangements of hyperplanes, *Information Comput.*, 106(2):286–303, 1993.

29 F. Meyer auf der Heide, A polynomial linear search algorithm for the n-dimensional knapsack problem, *J. ACM*, 31:668–676, 1984.

30 M. Sharir, The Clarkson–Shor technique revisited and extended, *Combinat. Probab. Comput.*, 12:191–201, 2003.

31 M. Steele and A. Yao, Lower bounds for algebraic decision trees, *J. Alg.*, 3:1–8, 1982.