Primal–Dual Algorithms for Data Depth

David Bremner, Komei Fukuda, and Vera Rosta

Abstract. The halfspace depth of a point p relative to a data set S in d-dimension is the smallest number of data observations from S in any closed halfspace containing p. A point with largest depth was considered as the generalization of the median of S by Tukey. The computation of the halfspace depth of a point is equivalent to the closed hemisphere problem, which was shown to be NP-complete by Johnson and Preparata. We propose primal–dual algorithms that update both an upper bound and a lower bound of the depth and terminate as soon as the two bounds coincide. We report preliminary computational experiments.

1. Introduction

In non-parametric statistics, there is strong motivation to find good definitions of multivariate median and ranks that satisfy certain statistical properties. Donoho and Gasko [**18**], Rousseeuw and Hubert [**40, 41**] give extensive analysis of the necessary properties for meaningful statistical depth measures, see also our previous survey on data depth [**24**]. Over the last 30 years various data depths measures were introduced as multivariate generalizations of ranks to complement classical multivariate analysis, by Tukey (1974), Oja (1983), Liu (1990), Donoho and Gasko (1992), Singh (1992), Rousseeuw and Hubert (1999) among others. It turns out that the halfspace depth of Tukey and the regression depth of Rousseeuw and Hubert have all the required statistical properties. The simplicial depth introduced by Liu [**32**] is another very well studied data depth measure. As these measures are essentially geometrical, computational geometers were active in finding efficient algorithms for their computation, as fast computation of any statistical measure is crucial for any relevant practical statistical application. For bivariate data optimal computational methods could be worked out, but in higher dimensions there is no satisfactory algorithm. In this paper we suggest new types of algorithms to overcome some of the difficulties.

The *halfspace depth*, also known as *location depth* or *Tukey depth* introduced by Tukey [**47**] is perhaps the best known among the data depth measures. The

The first author was supported in part by an NSERC Canada Discovery Grant (RGPIN 228095). Equipment was provided in part by a Canada Foundation for Innovation New Opportunities project (3507).

The second author was supported in part by NSERC grant (RGPIN 249612), Canada.

The third author was supported in part by NSERC grant (RGPIN 249756), Canada.

halfspace depth of a point p relative to a data set S of n points in $\mathbb{R}^d$, is the smallest number of points of S in any closed halfspace with boundary through p. It is easy to see that open halfspaces containing p can be considered instead of closed ones. The measure of halfspace data depth has good statistical properties, for example it is affine equivariant, i.e. it remains the same after affine transformations of the data, and it is robust in any dimension. Robustness is measured by how difficult it is to break down the measure in the following sense [**18**]. The *breakdown point* $\epsilon^*(T, X)$ is the smallest contamination fraction under which the estimator breaks down:

$$\epsilon^* = \min\{\frac{m}{n+m} : \sup_{Y^{(m)}} ||T(X^{(n)} \cup Y^{(m)}) - T(X^{(n)})|| = \infty\}$$

where $X^{(n)}$ denotes a given dataset of size n, T is the estimator of interest, and another dataset $Y^{(m)}$ of size m is added to $X^{(n)}$. If it is possible to make $T(X^{(n)} \cup Y^{(m)}) - T(X^{(n)})$ arbitrarily large, we say that the estimator breaks down under contamination fraction $m/(n+m)$. For example the mean has breakdown point $\frac{1}{(n+1)}$ and the one-dimensional median has breakdown point $\frac{1}{2}$. We can say that the median is a robust measure but not the mean. Among location estimates the median has the best achievable breakdown point. The breakdown point of maximum halfspace depth is close to $1/3$ for centrosymmetric distributions and it is at least $\frac{1}{d+1}$ for X in general position [**18**].

For $d = 2$ the problem of computing halfspace depth has been solved to optimal efficiency [**15**], [**37**], [**35**], [**29**], [**36**] and [**43**]. For higher dimensions the situation is very different. Let S be a set of n points on the d-dimensional sphere centered at the origin. The *closed (open) hemisphere problem* is to find the closed (open) hemisphere containing the largest number of points of S. Johnson and Preparata [**26**] proved in 1978 that the closed (open) hemisphere problem is NP-hard, which clearly implies that the computation of the halfspace depth of a given point is also NP-hard. As a consequence very little effort was given to design exact and deterministic algorithm to compute the halfspace depth of a point relative to a given data set in higher dimensions. For the special case when the dimension d is considered fixed, or constant, Johnson and Preparata [**26**] gave an algorithm of $\Theta(n^{d-1} \log n)$ time complexity which they did not consider practical if the dimension d is more than 4. No analysis of memory requirement or implementation is given. Fixed dimension algorithms are rarely implemented in practice for higher than $d = 3$, as a large power of the fixed dimension might be hidden in the constant thus computation might be too time consuming and additionally the memory requirement might be also large. Later Rousseeuw, Ruts and Struyf [**43, 45**] rediscovered the same algorithm and implemented it for $d = 2$ and $d = 3$. There is no improvement to date when the dimension is fixed.

For practical purposes it is necessary to approach the problem of computing halfspace depth in some novel ways. NP-hardness results concentrate on the worst case, that might hinder the possibility of computing partial information quickly in practice. For statistical applications, *primal–dual* algorithms might be more valuable, i.e. those algorithms that update both upper and lower bounds of the depth, and terminate as soon as the bounds coincide. Such an algorithm can provide useful information on the target depth even when the user cannot afford to wait for the algorithm to terminate. In addition, unlike "enumeration-based" algorithms whose termination depends on the completion of enumeration (of all halfspace partitions), primal–dual algorithms might terminate much earlier than the worst-case time bound. In this paper we present a primal–dual type algorithm for computing the halfspace depth.

Johnson and Preparata reformulated the closed (open) hemisphere problem to the equivalent form of finding the maximum feasible subsystem of a system of strict (weak) homogeneous linear inequalities, Max FS. Independently, the infeasibility analysis for Linear Programming (LP) is also concerned by the same problem. Chakravarti [**12**] has also proved that MaxFS is NP-hard, moreover Amaldi and Kahn showed that approximation of MaxFS is NP-hard, as well. There is an extensive literature where the complexity of MaxFS was analized and various algorithms were suggested, using results of integer programming and combinatorial optimization, independently from the computational geometry or statistics literature for data depth. A wide range of optimization techniques were used that might work well for specific applications. But in general, the computations elaborated in the MaxFS literature may not be appropriate for the data depth problem, see our survey [**24**] for more detailed analysis of the relations between data depth and MaxFS computations and results. The exact algorithms we presented in [**23**] for the computation of halfspace depth can easily be adapted to solve homogeneous Max FS problems. The interest in the computations of Max FS goes beyond LP, they have been applied extensively in many areas, for example in telecommunications [**39**], neural networks [**1**], linear classification problems in machine learning [**9**], image processing [**4**] and in computational biology [**48**].

For a given dataset S, the set D_k of all points in $\mathbb{R}^d$ with depth at least k is called the *contour* of depth k in statistics [**18**], though this expression is sometimes used just for the boundary of D_k. The depth regions are convex and nested [**18, 24**]. For any dimension d, as a consequence of Helly's theorem [**16**], the deepest location is at least $\lceil \frac{n}{d+1} \rceil$. The set of points with at least this depth is called the *center* in computational geometry, and its computation in the plane drew considerable attention, [**35, 29, 43, 37**]. While the problem is solved to optimality in the plane, in higher dimensions the computation of the center is much more challenging. Using Radon partitions Clarkson et al. [**13**] compute approximate center points in any dimension, finding a point of depth at least n/d^2 with high probability. This terminology of center might be misleading as for symmetric data the maximum depth is $n/2$, as observed by Donoho and Gasko [**18**]. Miller et al. [**36**] proposed a method to compute the depth regions and their boundaries in the plane using topological sweep. Unfortunately their method seems not to generalize easily to higher dimensions. Although the boundary of the halfspace depth regions D_k have no satisfactory characterization, it is still possible to design an algorithm to compute D_k, for all k, that can be based on the enumeration of those cells in the dual hyperplane arrangement that correspond to k-sets. A subset X of an n-point set S is called a *k-set* if it has cardinality k and there is an open halfspace H such that $H \cap S = X$. Therefore there is a hyperplane h, the boundary of H, that "cleanly" separates X from the remaining points of S. It is not a surprise that the exact algorithm for computing the depth regions mentioned in [**23**] using k-sets is also quite inefficient in higher dimensions. Some ideas for a heuristic algorithm are in [**24**].

In statistics the computation of a multidimensional median would be perhaps more important than the computation of the halfspace depth of a given point. But this is not an easier task. The complexity results of Johnson and Preparata were extended by Teng [**46**]. Testing whether the halfspace depth of a point is at least some constant is coNP-complete even in the special case of testing if a point is a center point. Hence there is little hope of finding a polynomial algorithm to compute a multivariate median. Struyf and Rousseeuw [**45**] propose a heuristic approximation algorithm for high-dimensional computation of the deepest location. There is no measure of how good this approximation is and there is no proof that the algorithm converges to a Tukey median. This is not surprising, as there are studies by Amaldi and Kahn [**2, 3**], showing that even the approximation of the equivalent problem, finding a maximum feasible subsystem of a linear inequality system, is NP-hard.

Some of the main additional data depth measures have strong connection to the halfspace depth. Linear regression is optimal if the errors are normally distributed, but when outliers are present or the error distributions are not normal linear regression is not considered robust. Rousseeuw and Hubert [**41**] introduced the notion of regression depth by generalizing the halfspace depth to the regression setting. In [**41**] it is shown that the deepest fit is robust with breakdown value that converges to $1/3$ for a large semiparametric model in any dimension $d \geq 2$. For general linear models they derive a monotone equivariance property. Rousseeuw, Hubert, Ruts and Struyf [**43, 41, 42, 44**] described algorithms for testing the regression depth of a given hyperplane, requiring time $O(n^3)$ in the plane [**41**] or time $O(n^{2d-1} \log n)$ in fixed dimensions $d \geq 3$ [**42, 44**]. In the plane van Kreveld et al. [**27**] gave an algorithm for finding the optimum regression line in time $O(n \log^2 n)$ and it was improved by Langerman and Steiger to $O(n \log n)$ [**30**]. By a breadth-first search of the dual hyperplane arrangement one can find theoretically the hyperplane of greatest depth for a given point set in time $\Theta(n^d)$, [**27**]. There is no implementation to this algorithm as the memory requirement is also $\Theta(n^d)$, which is prohibitive in higher dimensions. The exact algorithm presented in [**23**] for the computation of the halfspace depth of a point is based on a memory efficient hyperplane arrangement enumeration algorithm and can be immediately applied to compute the regression depth of a hyperplane, see [**24**].

Another important statistical property for good data depth measure require that the data depth regions be nested. This property is essential to design statistical estimation theory based on non-parametric multivariate ranks and follows easily for the halfspace depth. The simplicial depth of a point $p \in \mathbb{R}^d$ with respect to a given dataset S was defined by Liu [**32**], as the number (proportion) of simplices formed with points in S, that contain the point p. Discrete geometry gives the theoretical background for many algorithms. Helly's theorem and its relatives [**16**] are essential for the unification of different data depth measures, and to describe their properties. In a previous survey [**24**] we describe bounds relating halfspace depth and simplicial depth based on discrete geometry theorems. The simplicial depth is affine equivariant, but surprisingly the corresponding depth regions are not necessarily nested, as shown by examples already in the plane [**11, 49**], hence some modified definitions are suggested [**11, 38**]. It turns out [**24**], that the halfspace depth of a point p is actually the cardinality of the minimum transversal of the simplices containing p. The simplices containing the point p have an important role, we use them to find lower bounds for the halfspace depth in our primal–dual algorithm.

In [**23**], an exact, memory efficient and highly parallelizable algorithm was presented to compute the halfspace depth of a given point p with respect to a given set of n points in $\mathbb{R}^d$. To speed up the computation various techniques were suggested, namely branch-and-bound, that makes the exact algorithm adaptive, parallel processing based on the reverse search paradigm, heuristics algorithm for preprocessing and giving fast upper bounds on the depth and finally integer programming to give fast lower bounds on the depth. These techniques provide a primal–dual type algorithm, the computation stops when the upper and lower bounds coincide or when the exact algorithm finishes computation. There is strong motivation in non-parametric statistics, computational geometry and linear optimization to design efficient algorithms with implementations for halfspace depth related computations. As we mentioned earlier these are hard problems hence there is no hope for polynomial algorithms. We use instead the best available tools from geometric computation, linear and integer programming. In this paper we show how our different techniques and algorithms can work together to overcome the computational difficulties and we present our latest computational experiments.

2. Primal–Dual Algorithm for the Computation of Halfspace Depth

The complexity results mentioned in the introduction suggest that even an ideal algorithm for any of the main problems might need too much time. Hence it is important to have an algorithm that gives useful information even if the computation is stopped before its completion. Upper and lower bounds of the optimal value could be very useful for the user. Additionally, these informations are also valuable for designing exact algorithms that might terminate much earlier than the worst-case complexity bounds. We use the term *primal–dual* for algorithms that update both upper and lower bounds of the target measure and terminate as soon as the two bounds coincide.

In order to guarantee the termination, one has to incorporate an exact enumeration scheme in a primal–dual type algorithm. The algorithm incorporated here is a reverse search algorithm of time complexity $O(n\ LP(n,d)\ |\mathfrak{C}_S^p|)$ and space complexity $O(n\ d)$, that constructs all $|\mathfrak{C}_S^p|$ cells of the dual hyperplane arrangement, for any given S and p, where $LP(n,d)$ denotes the time complexity of solving a linear program of d variables and n inequality constraints. The (worst-case) complexity of the primal–dual algorithm is dominated by the complexity of this enumeration algorithm.

Let $S = \{p^1, \dots, p^n\}$ be an n-point set and p be a given point in $\mathbb{R}^d$. Here we briefly explain how one can design a primal–dual algorithm for the halfspace depth of p, based on the ideas used in [**23**]. First of all, one can easily see that any hyperplane through p gives an upper bound for the halfspace depth of p. Any heuristic algorithm, such as the "LP-walk" method that starts from a random hyperplane and aims to get to a better one in the neighborhood, can find a good candidate hyperplane quickly. In [**23**] such a random walk is used. To compute exactly the halfspace depth of a point p with respect to the n-point set $S = \{p^1, \dots, p^n\}$, an oriented hyperplane h is represented by the signvector $X \in \{+, -, 0\}^n$ of a cell in the dual hyperplane arrangement, so that an index j is in the positive support X^+ of the cell X, if and only if the corresponding point p^j is in the positive halfspace h^+ bounded by h. It is possible to restrict the dual hyperplane arrangement so that the point p is in the positive halfspace, h^+. Then the halfspace depth of the point p is the minimum $|X^+|$ over all restricted cells X. (For more details on the exact algorithm see Section 3.) The greedy heuristic random walk starts at an arbitrary cell and uses LP successively to move to a neighboring cell with smaller cardinality of positive support if it exists. Successive application define a path starting at the first randomly selected cell, through cells whose signvectors have monotone decreasing number of positive signs, until no more local improvement can be made, arriving to a local minimum. Any local minimum gives an upper bound of the halfspace depth of the point p.

Lower bounds of the halfspace depth can be obtained using integer programming techniques and LP relaxation. In fact the halfspace depth problem, as pointed out by Johnson and Preparata [**26**] is an optimization problem and therefore it is natural to use optimization techniques besides geometric or computational ones. A subset R of S is called *minimal dominating set* (MDS) for the point p, if the convex hull of R contains p, and it is minimal with this property. It follows from Charathéodory's theorem [**8**] that an MDS must be a simplex. It is easy to see that the cardinality of a minimum transversal of all MDS's is the halfspace depth of the point p. Assume that the heuristics stops at a cell X. Then, for each $j \in X^+$, there exists at least one MDS R_j such that $R_j \cap X^+ = \{j\}$. Let I be any collection of MDS's containing at least one such R_j for each $j \in X^+$. Then X^+ is a minimal transversal of I. To compute a minimum transversal for I, first the sets $X^- \cup j$ are generated for all $j \in X^+$. Using successive LP's these are reduced to MDS's, forming the set I. Let $\mathbf{c}$ be the characteristic matrix of I, where each row corresponds to an MDS. Let $\mathbf{y}^T = (y_1, \dots, y_n)$ be a 0/1 vector representing a transversal. The minimum transversal of I is a solution to the integer program: $min \sum_{i=1}^n y_i$ subject to $\mathbf{cy} \geq \mathbf{1}, \quad \mathbf{y} \in \{0,1\}^n$. Let us denote by c_I, the cardinality of the minimum transversal of the set I, and by c, the cardinality of the minimum transversal for all MDS's. The optimum value c_L obtained through the LP relaxation (with $0 \leq y_i \leq 1$), satisfies $c_L \leq c_I \leq c = depth$ (the halfspace depth of p). If this lower bound equals the upper bound obtained heuristically, then the global minimum is reached. A lower bound of the halfspace depth can be computed each time an upper bound is obtained by heuristics.

It is not too difficult to see that the lower bounds we obtain by the above procedure are usually far from the depth and we need additional ideas to improve the bounds. Here is an idea that works quite nicely, due to the fact that one can easily check whether a minimum transversal of a set of MDS's is minimum for the complete set of MDS's. More precisely, a minimum transversal T of a set of MDS's is a minimum transversal of all MDS's if and only if $S \setminus T$ does not contain p in its convex hull. Consider the LP: $p = \sum_{i=1}^{n} p^i \lambda_i$ subject to $\sum_{i=1}^{n} \lambda_i = 1, \lambda_i = 0$ for $i \in T$ and $\lambda_i \geq 0$ for all i. This LP has no feasible solution if and only if $|T|$ is the halfspace depth of p. Any basic solution to this LP is an MDS disjoint from T, it can be obtained, for example, by the simplex method. To generate all or some set of MDS's disjoint from T, one can use vertex enumeration on the polytope whose vertices are the basic solutions to this LP. These ideas give a systematic generation of dominating sets and optimality checking. Section 5.1 on computational experiment describes some additional details of MDS generation.

Note that the same vertex enumeration techniques could be used to compute simplicial depth of a point p with respect to a point set S. To compute the simplicial depth it is necessary to compute all MDS's. But the computation quickly becomes overwhelming as the total number of MDS's might be very large. In a simple computational experiment of $n = 50$ points and $d = 5$ dimension the number of MDS's is 119868, which is still not too big, but for a reasonable size problem there are simply too many to enumerate even with the most efficient vertex enumeration codes. In contrast, only a subset of MDS's may be necessary to compute the halfspace depth.

3. Enumeration Scheme

For completeness, in this section we describe in more detail the exact, parallel, memory efficient reverse search enumeration algorithm and the branch and bound technique developed in [**23**]. This algorithm is a modified hyperplane arrangement enumeration algorithm adapted to the halfspace depth problem, hence we first describe the dual hyperplane arrangement and its relation to halfspace depth.

3.1. Hyperplane arrangements. We reformulate the problem of computing the halfspace depth of a point in the setting of hyperplane arrangements. First we consider the data points $S = \{p^1, p^2, \ldots, p^n\}$ in $\mathbb{R}^d$. The dual hyperplane arrangement is such that to each datapoint in the primal there is a corresponding hyperplane in the dual. There is a standard way to orient this dual hyperplane arrangement, which gives a unique sign vector assignment to each face determined by the orientations of its bordering hyperplanes. It will be useful to embed both the primal and the dual into separate d-dimensional hyperplanes in $\mathbb{R}^{d+1}$. Corresponding to a point $(p^i, 1)$ in the primal is the hyperplane whose normal vector is $(p^i, 1)$.

Primal: $\hat{p}^i = (p^i, 1) \in \mathbb{R}^{d+1}, \quad i = 1, 2, \ldots, n$
Dual arrangement A_S: $< \hat{p}^i, \hat{x} >= 0, \quad i = 1, 2, \ldots, n.$

For any $\hat{x} \in \mathbb{R}^{d+1}$, define the *sign vector* $\sigma(\hat{x}) \in \{+, 0, -\}^n$ as the vector whose ith component is the sign of $< \hat{p}^i, \hat{x} >$. We shall use $\mathfrak{F}_S$ to denote the set of all sign vectors

$$\mathfrak{F}_S = \{\sigma(\hat{x}) : \hat{x} \in \mathbb{R}^{d+1}\}. \tag{3.1}$$

The set of vectors having the same sign vector forms a face of the dual hyperplane arrangement. In this sense, we identify the faces of the arrangement A_S and the sign vectors in $\mathfrak{F}_S$. In particular, the full dimensional faces called the *cells* are represented by the sign vectors with no zero components. We use $\mathfrak{C}_S$ to denote the set of cells in terms of their sign vectors. For any sign vector X, we use X^+ to denote the *positive support* $\{i : X_i = +\}$. Similarly, we use the *zero support* X^0, the *negative support* X^- and the *support* $\underline{X} := X^+ \cup X^-$. The following is the basic duality.

PROPOSITION 3.1. *For each $X \in \{+, 0, -\}^n$, $X \in \mathfrak{F}_S$ if and only if there is an oriented hyperplane h in $\mathbb{R}^d$ such that $X^+ \subseteq h^+$, $X^0 \subseteq h$ and $X^- \subseteq h^-$.*

A slightly modified version of the duality above is convenient for characterizing the halfspace depth of any fixed point $p \in \mathbb{R}^d$.

PROPOSITION 3.2. *Let p be any fixed point in $\mathbb{R}^d$, and define*

$$\mathfrak{F}_S^p = \{\sigma(\hat{x}) : \hat{x} \in \mathbb{R}^{d+1} \text{ and } <\hat{p}, \hat{x}>= 1\}. \tag{3.2}$$

For each $X \in \{+, 0, -\}^n$, $X \in \mathfrak{F}_S^p$ if and only if there is an oriented hyperplane h in $\mathbb{R}^d$ such that $X^+ \cup \{p\} \subseteq h^+$, $X^0 \subseteq h$ and $X^- \subseteq h^-$.

In this proposition, the set $\mathfrak{F}_S^p$ represents the faces of the affine arrangement A_S^p, the intersection of A_S and the hyperplane $<\hat{p}, \hat{x}>= 1$. Recall that the halfspace depth of p is the smallest number of points of S in any open halfspace containing p. By Proposition 3.2, the halfspace depth of p therefore is the minimum cardinality of the positive support in any of the sign vectors in $\mathfrak{C}_S^p$, the set of sign vectors of $\mathfrak{F}_S^p$ without any zero components. To compute the halfspace depth of a point p, is equivalent of finding a cell in the affine arrangement A_S^p which lie in the smallest number of positive sides of its hyperplanes.

PROPOSITION 3.3. *Computing the halfspace depth of a point is equivalent to finding a cell sign vector of smallest positive support in a hyperplane arrangement.*

3.2. Cell enumeration of an arrangement. Consider the central arrangement A_S of n hyperplanes in $\mathbb{R}^{d+1}$ given by a set S of points in $\mathbb{R}^d$. Let p be any fixed point in $\mathbb{R}^d$. As we discussed in the previous section, computing the halfspace depth of p can be reduced to listing all cells (i.e. determination of $\mathfrak{C}_S^p$) in the affine arrangement A_S^p, that is a d-dimensional arrangement of n hyperplanes.

There are several different algorithms for the cell enumeration for a general d-dimensional arrangement. The incremental algorithm [**19**] is theoretically optimal for fixed d. However, it requires space as large as the output size and is quite hard to implement as well. There is a reverse search algorithm by Avis and Fukuda [**7**] which is memory efficient (i.e. the space complexity is polynomially bounded by the input size). The time complexity is not optimal for the worst case output but it runs in time polynomial in both input size and output size that might be considered reasonable for enumeration problems with exponential outputs or NP-hard.

We shall use here a recent modification [**20**] of the reverse search algorithm with an improved complexity. Here, $LP(n, d)$ denotes the time to solve an LP with n inequalities in d variables.

THEOREM 3.4. *There is a reverse search algorithm of time complexity $O(n\ LP(n, d)\ |\mathfrak{C}_S^p|)$ and space complexity $O(n\ d)$ that computes $\mathfrak{C}_S^p$ for any given S and p.*

We define two key functions that determine a reverse search algorithm. These are a *finite local search function* f and an *adjacency oracle function* Adj. An adjacency oracle defines implicitly a graph G on the set $\mathfrak{C}_S^p$ as its vertex set by returning the set of neighbor cells of any given cell. A local search function maps any cell in $\mathfrak{C}_S^p$ to an adjacent cell. A local search is called *finite* if there exists a special cell $C \in \mathfrak{C}_S^p$ such that for any $X \neq C$, there is a positive integer k such that $f^k(X) = C$. Once a finite local search f is fixed, we have a uniquely defined directed tree T_f rooted at C with edge set $\{(X, f(X)) | X \in \mathfrak{C}_S^p \setminus \{C\}\}$ spanning all elements in $\mathfrak{C}_S^p$. The reverse search algorithm is a procedure to visit all members of $\mathfrak{C}_S^p$ by tracing the spanning tree T_f from the "optimal" element C, relying only on the two functions f and Adj.

To define an adjacency oracle, let X be any cell in $\mathfrak{C}_S^p$. We call an index $j \in \{1, \dots, n\}$ *flippable* in X if the vector obtained from X by reversing the jth sign is again a cell. Our *adjacency oracle* $Adj(X, j)$ is defined to return this new cell if j is flippable, and NULL otherwise, for $j = 1, 2, \dots, n$. The obvious upper bound of the maximum degree of a vertex is n. Clearly one LP of size $n \times d$ is sufficient to decide whether j is flippable at any given cell X.

To define a local search function, let C be any cell in $\mathfrak{C}_S^p$ that is known at the beginning of computation. To find one cell is easy: for example by selecting a random point in $\mathbb{R}^d$ and check its signature. The probability that a random point lies on any hyperplane is zero. Now, for any $X \in \mathfrak{C}_S^p \setminus \{C\}$, we must define an easily computable next cell X' which is adjacent to X and is "closer" to the goal C. Any such systematic rule will define f (i.e. $f(X) = X'$). It is easy to show that any cell X different from C has a flippable index j such that $X_j = -C_j$, flipping such an index leads us closer to C.

In order to define f uniquely, we use so-called ray shooting. For this, we need two points, one interior point c of the goal cell C and one interior point x of a cell X. Now, shoot a ray from x to c. It will hit all hyperplanes separating X and C. We select the first hyperplane hit by the ray. In case of tie, we employ the standard symbolic perturbation to resolve it. This gives us a way to move to a neighbor of X. It is important to note that an interior point we select for each cell must be uniquely defined.

With the ray shooting local search f, we only need to solve one LP to evaluate it, since the point c can be computed at the beginning and stored through the course of the algorithm. By applying Theorem 2.4 in [**7**], we immediately obtain Theorem 3.4. The resulting reverse search algorithm generates all cells by reversing f in all possible ways. For our purpose of computing the halfspace depth we can truncate some parts of the enumeration tree by using an elementary logic, similar to the used in *branch and bound* technique. For $X \in \mathfrak{C}_S^p$, define $g(X) = |X^+ \cap C^-|$. It is easy to see that $g(X)$ is the lower bound on the cardinality of Y^+ of all cells Y under the subtree rooted at X. The reason is simple, the positive components of X which can be turned to negative in any cells below X have to be in C^+.

The following algorithm computes the halfspace depth of p, essentially by the enumeration of all cells, with $\delta = n$ and C being any cell. The size $|C^+|$ of positive support of C is the initial upper bound of the depth of p, and thus it is expected that the algorithm performs better if one finds a reasonably good cell as C. One can for example try a random walk greedy heuristic algorithm to minimize the cardinality of positive support by a local search over the arrangement.

```
procedure BestCellSearch(Adj,δ,C,f);
    X := C; depth = |X⁺|; j := 0; (* j: neighbor counter *)
    repeat
        while j < δ do
            j := j + 1;
            next := Adj(X, j);
            if next ≠ 0 and g(X) < depth then
                if f(next) = X then  (* reverse traverse *)
                    X := next; j := 0;
                    if |X⁺| < depth then depth = |X⁺| endif
                endif
            endif
        endwhile;
        if X ≠ C then (* forward traverse *)
            X′ := X;    X := f(X);
            j := 0;    repeat j := j + 1 until Adj(X, j) = X′ (* restore j *)
        endif
    until X = C and j = δ;
    output depth.
```

The worst case complexity is $O(n\ LP(n,d)\ |\mathfrak{C}_S^p|)$ which is the time complexity to generate all cells. The algorithm traces down to the lower subtree only if there is some hope to find a cell better than the currently best found with the cardinality of positive support stored as *depth*. If $g(X) \geq$ *depth*, the current best depth, no improvement can be made below X, hence computation of the enumeration of cells below X stops.

At any time the workload of searching below a cell X can be given to another processor. Memory is not shared, only the current best *depth*, which should be updated centrally. The algorithm is highly parallelizable and there are several existing tools that can be used, in particular, *ZRAM, the parallel computation library* [**10**] takes care of the reverse search mechanism, and provides efficient parallelization for no additional implementation effort. See also Section 4.6 for parallel computation implementation details.

4. Computational Experiments

In this section we describe an implementation of the primal–dual algorithm discussed in Section 2, along with some experimental results. The primal–dual algorithm can be thought of as a *walk-generate-solve* cycle, namely:

(1) Random walk in the dual arrangement to (hopefully) improve the upper bound (see Section 4.2).
(2) Generate dominating sets that are obstructions to optimality of the current best known solution (see Section 4.3).

(3) Solve an integer program to (hopefully) improve the lower bound (see Section 4.4).

This cycle is repeated until upper and lower bounds match, or a time limit is exceeded. If the primal–dual search does not complete with matching upper and lower bounds, the code switches to a reverse-search enumeration, using the ZRAM based implementation described [**23**]. We give a brief review of that implementation, and some experimental results related to parallelizability, in Section 4.6. In order to facilitate discussion of the performance of the primal–dual implementation, we give a summary of our experiments in Section 4.1. This is followed by a discussion of various aspects of the implementation. Because of the volume of data, we relegate detailed experimental results to an appendix.

4.1. Summary of Experiments. All of our datasets consist of (pseudo) randomly generated points near the surface of a hypersphere of fixed radius 10[1]. The task is then to compute the hyperplane depth of the mean of the generated points (typically close to the origin). It turns out that in high dimensions the origin is very near the boundary of the convex hull, so the problems become quite trivial. In order to generate more challenging inputs in higher dimension, we applied a symmetrization operation to the randomly generated points (i.e., for each x, add $-x$ to the set). Since the pure centrally symmetric examples are also trivial for our code, we apply a coordinate-wise perturbation to the negated vectors. The effect of this symmetrization operation is noticeable even in relatively low dimensions: compare Figure 1 with the symmetrized experiments in the same dimension in Figure 2. The x-axis is in these figures is only approximate; the number of points can be read off the label on the graph, i.e. 60b is the second random trial with 60 points.

It will be convenient to refer to various experimental datasets by name, rather than repeating the parameters involved. Table 4.1 lists the datasets used in our primal–dual experiments, along with the corresponding parameters. We also mention some figures and tables where these datasets are discussed. In general the name `dN` indicates dimension N fixed, while nK denotes number of points K fixed. A prefix of s indicates the symmetrization operation was applied. The one exception to this rule is the so-called `regression` set, which contains a variety of problems used for testing during program development.

For all the non-symmetrized examples, the solver was given 10 minutes[2] to run primal–dual search (actually, primal–dual search was terminated after the first complete walk-generate-solve cycle to push the total time above 10 minutes). The reverse-search enumeration (described in Section 3) was then allowed to run until a limit of 20 minutes was reached. For the symmetrized examples, 10 minutes was added to both of these limits. All primal–dual experiments were done on the LIDS AlphaServer cluster at the University of New Brunswick. Each compute node used was a 667 MHz AlphaServer DS10 with 256M of memory. Very roughly speaking each node is equivalent to 1.5 GHz Pentium 4, although the memory bandwidth is significantly higher. All parallel speedup tests reported in Section 4.6 were performed on the cluster of workstations at McGill University, each running FreeBSD 4.6 with dual PIII 733Mhz processors with 512 MB of RAM, and connected via 100baseTx-FD (full duplex 100 megabit) ethernet.

[1]Since we generate integer coordinates, degeneracy tends to increase with the dimension

[2]All job limits are in terms of wall time, so that e.g. a badly swapping job will still finish.

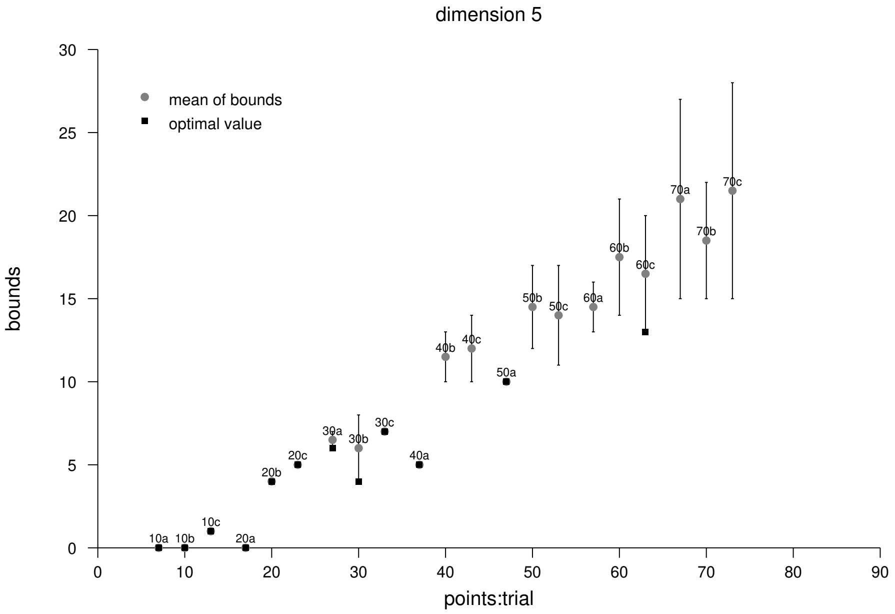

FIGURE 1. Gap between upper and lower bounds, dataset `d5`.

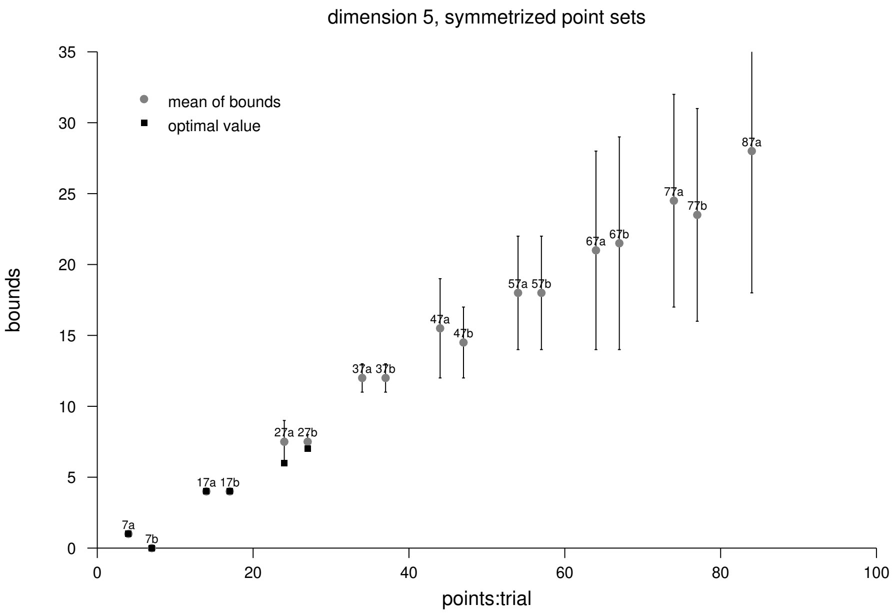

FIGURE 2. Gap between upper and lower bounds, dataset `sd5`.

The reader can get a broad overview of the performance of our implementation by consulting Table 4.2 (although it should be noted that there can be large variation in performance for identical parameters). Table 4.2 also serves as a gloss of the other tables of experimental results described in Appendix A.

Name	dimension	number of points	symmetrized	references
d5	5		no	Figure 1
sd5	5		yes	Figure 2
d10	10		no	Figure 4
n50		50	no	Figure 3
regression	$10\ldots30$	$3\ldots20$	no	Table 4.2

TABLE 4.1. A summary description of the datasets used in experiments.

n[a]	d[b]	done[c]	HLB[d]	HUB[e]	itr[f]	pre[g]	ip[h]	dom[i]	walk[j]	RUB[k]
10	3	yes	2	3	3	yes	0.20	0.00	0.04	
10	3	yes	3	5	3	yes	0.38	0.00	0.05	
30	5	yes	6	7	9	yes	14.08	0.42	0.59	
30	6	yes	5	5	6	yes	2.81	0.28	0.47	
30	7	yes	4	4	14	yes	7.63	0.66	1.61	
30	8	yes	1	1	2	yes	0.04	0.03	0.16	
50	4	yes	3	3	3	yes	0.43	0.41	0.40	
50	5	yes	13	14	10	yes	414.13	1.80	1.08	
50	6	yes	3	3	3	yes	0.44	0.55	0.56	
60	5	no	13	16	9	no	714.09	3.88	2.10	16
70	4	no	18	23	8	no	611.64	5.66	1.61	20
80	3	no	24	33	8	no	491.31	5.99		
100	3	yes	11	17	6	yes	0.27	9.95	3.73	
100	4	yes	13	15	8	yes	2.66	15.80	7.38	
100	20	no	5	19	9	no	674.77	161.76	116.21	19

[a]Number of input points.
[b]Input dimension.
[c]Run completed.
[d]Lower bound achieved by primal–dual search.
[e]Upper bound achieved by primal–dual search.
[f]Iterations of primal–dual search.
[g]Run completed by primal–dual search.
[h]CPU time spent solving integer programs.
[i]CPU time spent generating dominating sets.
[j]CPU time spent on random walks. Blank means job terminated before start of enumeration.
[k]Upper bound achieved by (reverse-search) enumeration.

TABLE 4.2. Results for the `regression` dataset.

4.2. Random Walk in Dual Arrangements. The main purpose of the random walk in dual arrangements is to find a cell X which has a large number $|X^-|$ of negative components, as presented in Section 2. An ideal implementation starts with a cell that is chosen uniformly randomly from the set of all cells and keeps moving to a randomly chosen neighboring cell that has a larger number of negative components.

Unfortunately, selecting a cell randomly is itself a hard problem. Our current implementation selects a uniformly random point from a large cube in the dual arrangement and selects the cell containing the point to be a starting cell. Of course this might be very far from selecting a random cell, as a cell with large volume is more likely to be chosen.

Our implementation of moving from the current cell X to a neighboring cell performs a deterministic search that selects the first index j such that $X_j = +$ and j determines a facet. We use the LP solver of `cddlib` package [**22**] to check whether each index j determines a facet or not. The solver allows one to use either `gmp` [**25**] exact arithmetic or hardware floating point arithmetic. All experiments reported in this paper used the latter which is considerably faster.

4.3. Finding Minimum Dominating Sets. Given a set $S \subset \mathbb{R}^d$ of points and a point p, we need to generate sets $Q \subset S$ such that $p \in \mathrm{conv}(Q)$ and Q is minimal. Consider the polytope P defined by the following set of inequalities

$$\begin{aligned} \lambda S &= p \\ \sum_i \lambda_i &= 1 \\ \lambda_i &\geq 0 \end{aligned} \tag{*}$$

In this setting, Q being minimal corresponds to a maximal set of nonnegativity inequalities (*) being tight, i.e. to a vertex of P. Since P has dimension $n - d - 1$, each set Q of size $d + 1$ corresponds to a non-degenerate vertex of P, while smaller dominating sets correspond to degenerate vertices. In general we need to find at most k vertices P, where k is a user parameter; for this reason we use Avis's `lrslib` [**5**], an implementation of reverse search for vertex enumeration. `Lrslib` performs extremely well in the absence of degeneracy. Since our experiments use pseudorandomly generated points, it is not surprising that the corresponding vertex enumeration problems seem relatively easy for `lrslib`. In Figure 3 we see that the dominating set computation is dominated by the time to perform random walks, see Section 4.2. This is actually stronger than it looks, since the dominating set is using `gmp` [**25**] to do exact arithmetic, while the random walk (see Section 4.2) is done using hardware floating point. Figure 4 shows that both of these costs are dominated by the time to solve integer programs.

4.4. Solving Integer Programs. Viewed purely as a combinatorial problem, the dominating set transversal problem is a version of the Hitting Set problem. While this is not especially theoretically advantageous (Hitting Set is known to be NP-Hard and hard to approximate), it does have a straightforward formulation as an integer program. We have implemented a branch and cut solver for the resulting integer programs based on the SYMPHONY framework [**28**]. The linear program relaxations are solved using the COIN-OR linear program solver CLP [**21**]. For depth problems with n points in d dimensions, the integer programs

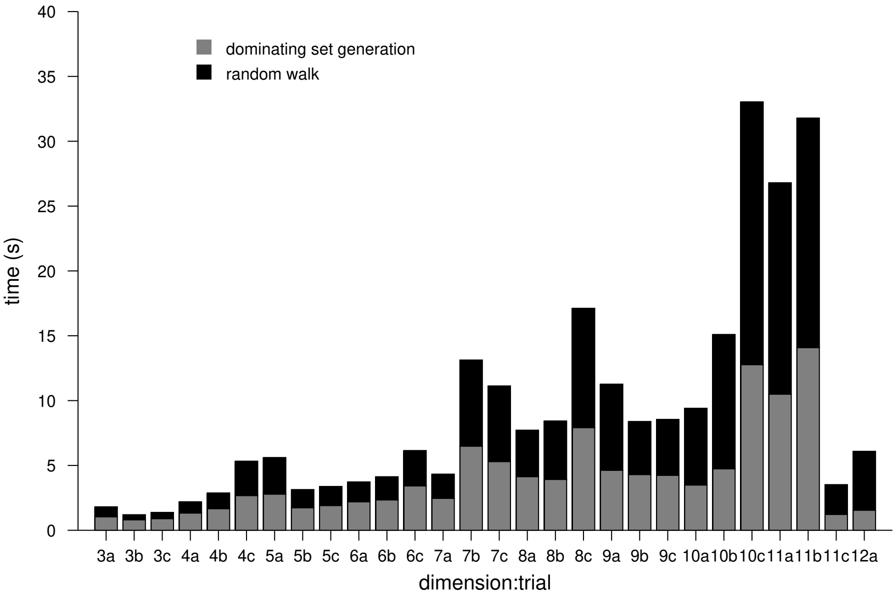

FIGURE 3. A comparison of time to generate dominating sets versus time to random walk, dataset n50.

have n binary variables; on the other hand Charathéodory's Theorem tells us that at most $d+1$ entries (i.e. the entries corresponding to one MDS) will be non-zero in any constraint. We have tried to take advantage of this sparseness property (and to prevent the LP solver from being bogged down with huge constraint sets) by implementing a process of *row generation.* The solver starts with some small subset of the constraints (about the square root of the total number in the experiments described here). The remaining constraints are then added during cut generation. Any proposed integer solution must also be checked against the full set of constraints. The remaining set of constraints not added to the initial formulation is kept in queue, with recently violated constraints pushed to the back of the queue. On many problems this strategy seems quite effective. For example, `domset50_5`, a randomly generated problem with 50 points in 5 dimensions, has a total of 119868 constraints in ambient dimension 50. This problem is solvable in 100 seconds on one of our AlphaServer reference nodes (see Section 4.1) using row generation, whereas the full constraint set takes over 3 hours to solve (i.e. using the same solver without row generation). Although improvement in this row generation implementation is certainly possible (e.g. the data structures are completely naive), it is probably more fruitful to consider generating cuts directly from the geometric data as described in Section 4.3. As Figure 4 suggests, solving integer programs is definitely a bottleneck with the current approach.

4.5. Integration. Since the current version (4.0b as of this writing) of SYMPHONY does not support use as a callable library, the integer program solver

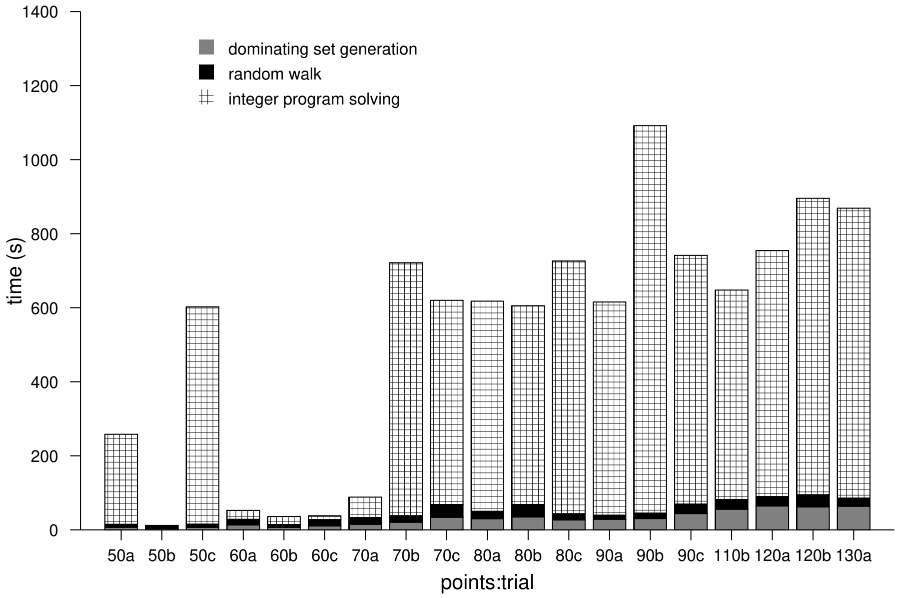

FIGURE 4. A comparison of time to generate dominating sets versus time to random walk and solve integer programs, dataset d10.

described in Section 4.4 is implemented as a separate executable, which communicates with the main program via pipes and files. Although this introduces a certain amount of overhead, it is not significant compared to the time to actually solve the integer programs. A more significant loss of efficiency is likely due to solving many integer programs that consist of adding constraints to the previous ones. In order to partially alleviate this, we pass the solution to the previous minimization as a lower bound to successive integer programs. Unfortunately, we know of examples (e.g. the example `domset50_5` discussed in Section 4.4) where this can actually slow down the solver on the resulting integer program. Another limitation of the current design is that the integer program solver does not have any knowledge of the geometry of the point set. It seems likely that the vertex enumeration techniques discussed in Section 4.3 can be applied to cut generation in solving the integer programs.

The dominating set generation code is currently used in two different ways. Initially the solver tries to generate dominating sets whose minimal transversal coincides with the point set attaining the current best upper bound. If this fails, it tries to find any dominating set that misses the current best lower bound solution. In a typical branch and cut fashion, the failure of the latter separation step provides a certificate of optimality for the lower bound solution. This is why many of the experiments listed as solved by the primal–dual search do not have matching upper and lower bounds.

4.6. Exact and Parallel Solutions Using Reverse Search. We implemented the reverse-search enumeration algorithm described in Section 3 in C using

ZRAM [**10, 34**]. This implementation is based on an earlier code developed for the zonotope construction problem [**20**]. In particular, it relies heavily on the linear programming solver in the `cddlib` package [**22**] which is efficient to solve dense LPs in relatively small dimensions. We have not employed any special programming tricks to optimize the code. Thus one can consider it as a faithful implementation of the algorithm `BestCellSearch`.

ZRAM's reverse search engine is known to be very efficient in automatically balancing computational loads across a large number of processors. The speedup rate can differ from application to application. But, for our application of cell enumeration, good news is that the search tree (having $\Omega(n^{d-1})$ nodes when the data points are in general position) is wide, as its depth is known to be linearly bounded (in fact bounded by the number n of data points). This indicates that the enumeration algorithm is expected to have good speedups, as it was observed in [**10**].

Our experiment was performed to test whether this expected behavior is in fact valid for our particular application or not. Our reverse search is a sort of partial reverse search, and somewhat different from those reported in [**10**] in the sense that it may not trace the whole cell enumeration tree. It truncates certain branches using the function g, as we explained in Section 3.

We chose two instances of the halfspace depth problem. Both are essentially the same problem but with two different points as reference point p. The number of data points is 100 in dimension 4. The fact is that if the depth of p is small, then the search tree itself tends to be small as well. In the first instance p has depth 6, and 14 in the second. The search tree for the second instance is much larger than that of the first instance.

In Table 4.3, we compare the speedup of the execution time with the number of processors varying from 2 to 16. Here the execution time is measured as the time past from the start to the end of execution by the system with a specified number of processors. We measure the speedup relative to the two processor case and we set the speedup of two processor case to be exactly 2.

As one sees immediately, the second (harder) instance gives better speedups. While the speedup stalls around 10 processors for the first instance, it continues to grow steadily for the second instance. This coincides with our expectation.

It is important to note that we do not need to take average over many instances to observe the general behavior of the enumeration algorithm. The algorithm behaves almost identically for the instances of the same characteristics (e.g. the same d, n and depth).

5. Acknowledgement

We would like to thank both Andrew Bogecho and Ron Simpson of the School of Computer Science at McGill University for their great support in setting up an environment for parallel computing with MPI.

References

[1] E. Amaldi, On the complexity of training perceptrons, *Artificial neural networks, T. Kohonen, K. Makosara, O. Simula and J. Kangas, eds, Elsevier, Amsterdam* (1991), 55–60.

[2] E. Amaldi and V. Kann, The complexity and approximability of finding maximum feasible subsystems of linear relations, *Theoretical Computer Science* **147** (1995), 181-210.

n^a	d^b	depth^c	#proc^d	exec time^e	speedup^f
100	4	6	2	43.90	2.00
100	4	6	4	25.10	3.50
100	4	6	6	19.11	4.59
100	4	6	8	16.54	5.31
100	4	6	10	11.78	7.46
100	4	6	12	10.99	7.99
100	4	6	14	10.15	8.65
100	4	6	16	10.71	8.19
100	4	14	2	1153.62	2.00
100	4	14	4	600.31	3.84
100	4	14	6	408.29	5.65
100	4	14	8	343.73	6.71
100	4	14	10	246.28	9.36
100	4	14	12	210.69	10.95
100	4	14	14	178.81	12.90
100	4	14	16	166.94	13.82

[a] Number of input points
[b] Input dimension
[c] Depth of the reference point
[d] Number of processors
[e] Execution time
[f] Speedup compared with 2 processor case

Table 4.3. Parallel speedup tests with two instances.

[3] E. Amaldi and V. Kahn, On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems, *Theor. Comput. Sci.* **209(1-2)** (1998), 237–260.

[4] E. Amaldi and M. Mattavelli, The MIN PCS problem and piecewise linear model estimation, *Discrete Appl. Math.* **118** (2002), 115–143.

[5] D. Avis. Computational experience with the reverse search vertex enumeration algorithm. *Optim. Methods Softw.*, 10(2):107–124, 1998. Dedicated to Professor Masao Iri on the occasion of his 65th birthday.

[6] D. Avis and K. Fukuda, A pivoting algorithm for convex hulls and vertex enumeration of arrangements of polyhedra, *Disc. Comput. Geom.* **8** (1992), 295–315.

[7] D. Avis and K. Fukuda, Reverse search for enumeration, *Discrete Appl. Math.* **65(1-3)** (1996), 21–46.

[8] I. Bárány, A generalization of Carathéodory's theorem, *Discrete Math* **40** (1982), 141–152.

[9] K.P. Bennett and E.J. Bredensteiner, A parametric optimization method for machine learning, *INFORMS J. Comput.* **9**, no 3 (1997), 311–318.

[10] A. Brüngger, A. Marzetta, K. Fukuda and J. Nievergelt, The parallel search bench ZRAM and its applications, *Annals of Operations Research* (1999), 45–63.

[11] M.A. Burr, E. Rafalin and D.L. Souvaine, Simplicial Depth: An Improved Definition, Analysis, and Efficiency for the Finite Sample Case, *DIMACS, Technical Reports* (2003), 2003–28.

[12] N. Chakravarti, Some results concerning post-infeasibility analysis, *Eur. J. Oper. Res.* **73**(1994), 139–143.

[13] K.L. Clarkson, D. Eppstein, G.L. Miller, C. Sturtivant and S-H. Teng, Approximating center points with iterative Radon points, *International Journal of Computational Geometry and Applications*, and *ACM Symp. Computational Geometry* (1993), 1–21.

[14] R. Cole, Slowing down sorting networks to obtain faster sorting algorithms, *Journal of the ACM* **34** (1987), 200-208.
[15] R. Cole, M. Sharir, and C. Yap, On k-hulls and related problems, *SIAM Journal on Computing*, **16(1)** (1987), 61–67.
[16] L. Danzer, B. Grünbaum and V. Klee, Helly's theorem and its relatives, *Proceedings of Symposia in Pure Mathematics 7, AMS* (1963), 101–180.
[17] D.L. Donoho, Breakdown properties of multivariate location estimators, Ph.D. qualifying paper, Dept. Statistics, Harvard Univ. (1982).
[18] D.L. Donoho and M. Gasko, Breakdown properties of location estimates based on halfspace depth and projected outlyingness, *The Annals of Statistics*, **20/4** (1982), 1803–1827.
[19] H. Edelsbrunner, J. O'Rourke, and R. Seidel, Constructing arrangements of lines and hyperplanes with applications, *SIAM J. Comput.***15** (1986), 341–363.
[20] J-A. Ferrez, K. Fukuda and T. Liebling, Solving the fixed rank convex quadratic maximization in binary variables by a parallel zonotope construction algorithm, *European Journal of Operations Research*(2003), to appear.
[21] J. Forest, D. de la Nuez, and R. Lougee-Heimer. CLP user manual. `http://www-124.ibm.com/developerworks/opensource/coin/Clp/userguide`, Sep. 2004.
[22] K. Fukuda, *cddlib reference manual, cddlib Version 093c*, Swiss Federal Institute of Technology, Lausanne and Zurich, Switzerland, 2003.
[23] K. Fukuda and V. Rosta, Exact parallel algorithms for the location depth and the maximum feasible subsystem problems, *Frontiers In Global Optimization, C.A. Floudas and P.M. Pardalos, editors, Kluwer Academic Publishers* (2003).
[24] K. Fukuda and V. Rosta, Data depth and maximum feasible subsytems, *Graph Theory and Combinatorial Optimization, D. Avis, A. Hertz, O. Marcotte eds, GERAD 25th ann., Kluwer Acad. Publ.* (2005), to appear.
[25] T. Grandlund, The Gnu MP bignum library, `http://www.swox.com/gmp`.
[26] D.S. Johnson and F.P. Preparata, The densest hemisphere problem, *Theoretical Computer Science* **6** (1978), 93–107.
[27] M. van Kreveld, J.S.B. Mitchell, P.J. Rousseeuw, M. Sharir, J. Snoeyink and B. Speckmann, Efficient algorithms for maximum regression depth, *Proceedings of the 15th Symposium on Computational Geometry, ACM June 1999* (1999), 31–40.
[28] L. Ladányi, T. K. Ralphs, and L. E. Trotter Jr., Branch, cut, and price: Sequential and parallel, In Michael Jünger and Denis Naddef, editors, *Computational combinatorial optimization*, volume 2241 of *Lecture Notes in Computer Science*, pages 224–260. Springer-Verlag, Berlin, 2001. Optimal or provably near-optimal solutions, Papers from the Spring School held in Schloß Dagstuhl, May 15–19, 2000.
[29] S. Langerman and W. Steiger, Computing a maximal depth point in the plane, *Proceedings of the Japan Conference on Discrete and Computational Geometry (JCDCG 2000)*, (2000).
[30] S. Langerman and W. Steiger, The complexity of hyperplane depth in the plane, *Symposium on Discrete Algorithms, ACM and SIAM* (2000).
[31] R. Liu, On a notion of data depth based on random simplices, *The Annals of Statistics* **18** (1990), 405–414.
[32] R.Y. Liu, On a notion of data depth based on random simplices, *Ann. Statist.* **18** (1990), 405–414.
[33] R.Y. Liu, Data depth and multivariate rank tests, *Statistical Analysis and Related Methods, ed. Y. Dodge, Amsterdam, Elsevier* (1992), 279–294.
[34] A. Marzetta, ZRAM homepage, *http://www.cs.unb.ca/profs/bremner/zram/*, and *http://www.cs.unb.ca/profs/bremner/zram/*.
[35] J. Matoušek, Computing the center of a planar point set, *Discrete and Computational Geometry: Papers from the DIMACS Special Year Amer.Math. Soc., J.E. Goodman, R. Pollack, W. Steiger, Eds.* (1992), 221-230.
[36] K. Miller, S. Ramaswami, P. Rousseeuw, T. Sellares, D. Souvaine, I. Streinu and A. Struyf, Fast implementation of depth contours using topological sweep, *Proceedings of the Twelfth ACM-SIAM Symposium on Discrete Algorithms, Washington, DC* (2001), 690–699.
[37] N. Naor and M. Sharir, Computing a point in the center of a point set in three dimensions, *Proc.2. Canadian Conference on Computational Geometry* (1990).

[38] E. Rafalin and D.L. Souvaine, Computational Geometry and Statistical Depth Measures, Theory and Applications of Recent Robust Methods, M. Hubert, G. Pison, A. Struyf and S. Van Aelst eds. *Series: Statistics for Industry and Technology, Birkhauser, Basel*, preprint.

[39] F. Rossi, A. Sassano and S. Smriglio, Models and algorithms for terrestrial digital broadcasting, *Annals of Operations Research* **107(3)** (2001), 267–283.

[40] P.J. Rousseeuw, Multivariate estimation with high breakdown point, *Mathematical Statistics and Applications, Vol B* (W. Grossman, G. Ch. Pflug, I. Vincze and W. Wertz, eds., Reidel, Dordrecht) (1985), 283–297.

[41] P.J. Rousseeuw and M. Hubert, Regression depth, *Journal of American Statistical Association* **94** (1999), 388–402.

[42] P.J. Rousseeuw and M. Hubert, Depth in an arrangement of hyperplanes, *Discrete and Computational Geometry* **22** (1999), 167–176.

[43] P.J. Rousseeuw and Ruts, Computing depth contours of bivariate clouds, *Computational Statistics and Data Analysis* **23** (1996), 153–168.

[44] P.J. Rousseeuw and A. Struyf, Computing location depth and regression depth in higher dimensions, *Statistics and Computing* **8** (1998), 193–203.

[45] A. Struyf and P. Rousseeuw, High-dimensional computation of the deepest location, *Computational Statistics and Data Analysis*, **34**, (2000) 415-426.

[46] S.H. Teng, Points, Spheres and Separators: a unified approach to graph partitioning, Ph.D. Thesis, Carnegie-Mellon Univ. School of Computer Science, August 1991.

[47] J.W. Tukey, Mathematics and the picturing of data, *Proceedings of the International Congress of Mathematicians, Vancouver* **2** (1975), 523–531.

[48] M. Wagner, J. Meller and R. Elber, Large-scale linear programing techniques for the design of protein folding potentials, *Tech. Report TR-2002-02, Old Dominion University* (2002).

[49] Y. Zuo and R. Serfling, General notions of statistical depth functions, *Ann. of Statist.,* **28(2)** (2000), 461–482.

Appendix A. Selected Experimental Results

In this appending we list more detailed experimental results for the datasets discussed in Section 4. For explanation of the tables, see Table 4.2 and the notes attached to it.

Faculty of Computer Science, University of New Brunswick, Fredericton, Canada
E-mail address: bremner@unb.ca

Institutes of Operations Research and Theor. Comp. Sci., ETH Zurich, and Institute of Mathematics, EPF Lausanne, Switzerland
E-mail address: fukuda@ifor.math.ethz.ch

Dept. of Mathematics and Statistics, McGill University, Montreal, Canada, and Rényi Institute of Mathematics, Hungarian Academy of Sciences
E-mail address: rosta@math.mcgill.ca (corresponding author)

n	trial	done	HLB	HUB	itr	pre	ip	dom	walk	RUB
20	0	yes	0	0	2	yes	0.00	0.01	0.09	
20	1	yes	0	0	2	yes	0.00	0.01	0.13	
20	2	yes	0	0	2	yes	0.00	0.01	0.14	
30	0	yes	2	2	9	yes	0.85	0.57	1.97	
30	1	yes	2	2	3	yes	0.42	0.15	0.44	
30	2	yes	1	2	4	yes	0.08	0.21	0.83	
40	0	yes	3	3	8	yes	1.19	1.42	3.58	
40	1	yes	2	2	3	yes	0.43	0.37	1.10	
40	2	yes	1	1	2	yes	0.04	0.08	0.75	
50	0	yes	6	6	13	yes	243.32	6.71	8.01	
50	1	yes	3	4	8	yes	2.36	3.35	6.20	
50	2	no	6	7	13	no	587.15	6.55	8.72	7
60	0	yes	5	8	14	yes	24.35	13.06	15.08	
60	1	yes	6	6	8	yes	21.75	6.48	7.69	
60	2	yes	4	6	13	yes	10.65	10.61	16.86	
70	0	yes	6	6	12	yes	55.34	14.88	18.17	
70	1	no	8	13	12	no	683.43	20.37	17.76	13
70	2	no	7	8	21	no	551.67	33.83	34.37	8
80	0	no	8	16	11	no	567.86	30.21	19.67	15
80	1	no	7	11	14	no	536.94	34.65	33.78	11
80	2	no	8	17	11	no	682.92	26.65	16.82	17
90	0	no	9	27	7	no	576.22	28.45	10.91	26
90	1	no	9	24	8	no	1046.77	30.46	14.64	24
90	2	no	9	15	11	no	671.74	43.94	25.63	15
100	0	no	9	28	11	no	410.19	36.12		
100	1	no	8	32	11	no	57.87	28.09		
100	2	no	9	25	11	no	285.64	44.71		

TABLE 1.4. Results for datase `d10`

n	trial	done	HLB	HUB	itr	pre	ip	dom	walk	RUB
12	0	yes	0	0	2	yes	0.00	0.00	0.02	
12	1	yes	0	0	2	yes	0.00	0.00	0.02	
22	0	yes	3	3	7	yes	0.90	0.15	0.72	
22	1	yes	2	2	3	yes	0.10	0.06	0.20	
32	0	yes	5	5	10	yes	81.25	0.92	2.20	
32	1	no	6	7	29	no	1232.53	2.29	6.93	7
42	0	no	7	11	12	no	1380.68	2.80	3.27	11
42	1	no	7	11	13	no	1501.98	3.06	4.25	11
52	0	no	7	17	8	no	1557.98	4.22	4.24	17

TABLE 1.5. Results for datase `sd20`

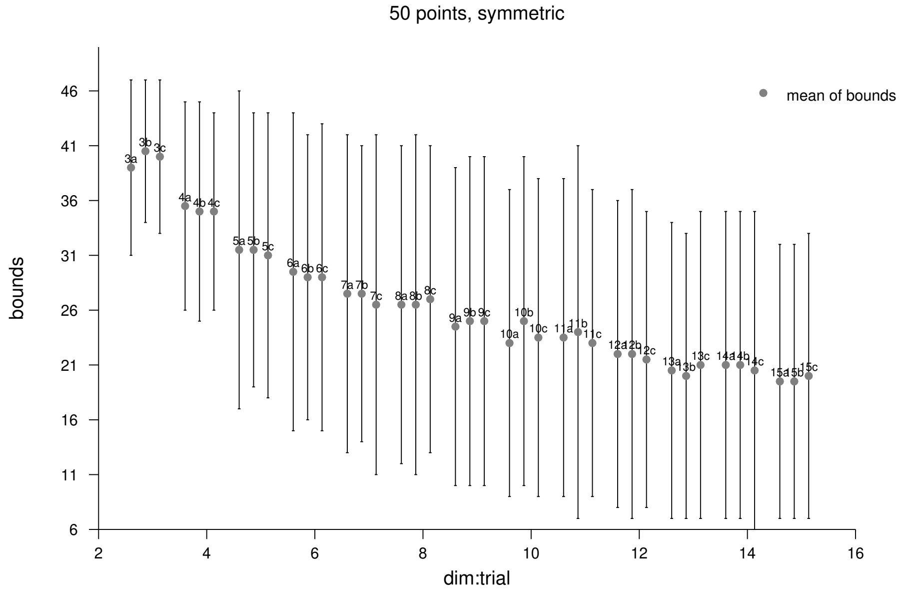

FIGURE 5. Gap between upper and lower bounds, dataset sn50.

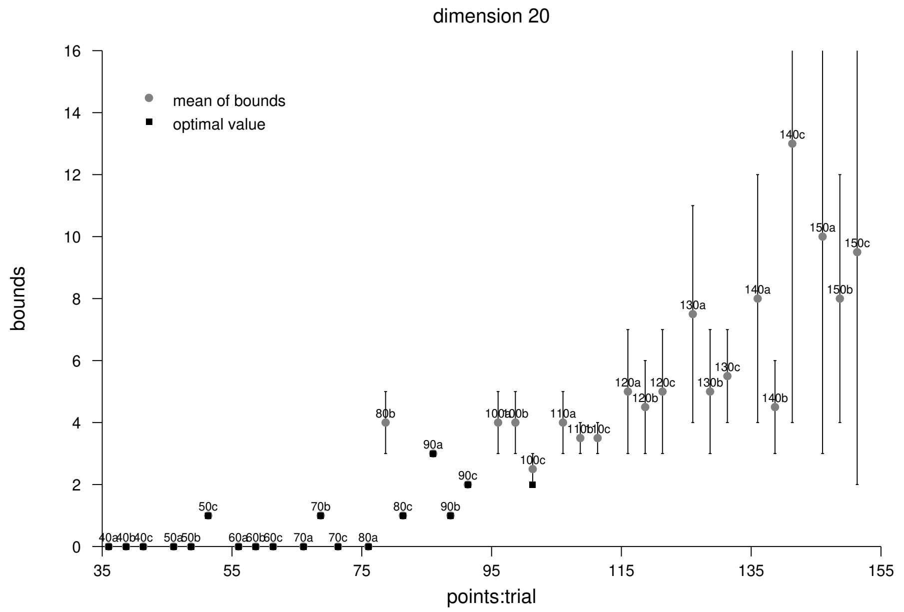

FIGURE 6. Gap between upper and lower bounds, dataset d20.

d	trial	done	HLB	HUB	itr	pre	ip	dom	walk	RUB
3	0	yes	18	20	10	yes	578.69	1.04	0.77	
3	1	yes	12	12	5	yes	2.80	0.81	0.39	
3	2	yes	18	18	6	yes	344.32	0.90	0.48	
4	0	no	15	17	7	no	1226.60	1.33	0.87	17
4	1	yes	12	15	9	yes	236.00	1.67	1.21	
4	2	yes	11	12	18	yes	138.06	2.68	2.65	
5	0	yes	9	12	15	yes	156.69	2.79	2.82	
5	1	no	11	13	8	no	737.90	1.74	1.40	13
5	2	no	12	16	9	no	791.12	1.91	1.47	16
6	0	no	10	15	8	no	1090.95	2.20	1.53	15
6	1	no	10	14	9	no	752.98	2.34	1.79	13
6	2	no	9	12	12	no	759.58	3.42	2.73	12
7	0	no	9	16	8	no	647.63	2.46	1.87	16
7	1	yes	7	9	18	yes	282.17	6.49	6.64	
7	2	no	8	11	15	no	644.03	5.30	5.83	11
8	0	no	8	14	11	no	635.77	4.14	3.58	13
8	1	yes	4	6	9	yes	8.25	3.92	4.51	
8	2	yes	5	6	20	yes	34.28	7.92	9.20	
9	0	yes	5	7	12	yes	38.51	4.63	6.64	
9	1	no	7	10	10	no	590.95	4.30	4.09	10
9	2	no	7	8	10	no	625.77	4.24	4.31	8
10	0	yes	4	4	9	yes	2.36	3.49	5.92	
10	1	yes	5	5	11	yes	13.04	4.74	10.36	
10	2	yes	6	7	25	yes	474.64	12.77	20.27	
11	0	yes	6	6	18	yes	365.53	10.50	16.30	
11	1	yes	5	8	22	yes	373.27	14.09	17.70	
11	2	yes	2	2	3	yes	0.44	1.22	2.30	
12	0	yes	2	2	4	yes	0.53	1.55	4.54	
12	1	yes	3	3	6	yes	1.08	2.98	5.96	
12	2	yes	4	4	8	yes	5.84	3.85	8.39	
13	0	yes	3	3	5	yes	0.76	1.85	6.29	
13	1	yes	3	5	13	yes	8.20	8.50	16.51	
13	2	yes	3	4	14	yes	13.29	7.78	18.87	

TABLE 1.6. Results for datase `n50`; dimensions above 13 are very easy.

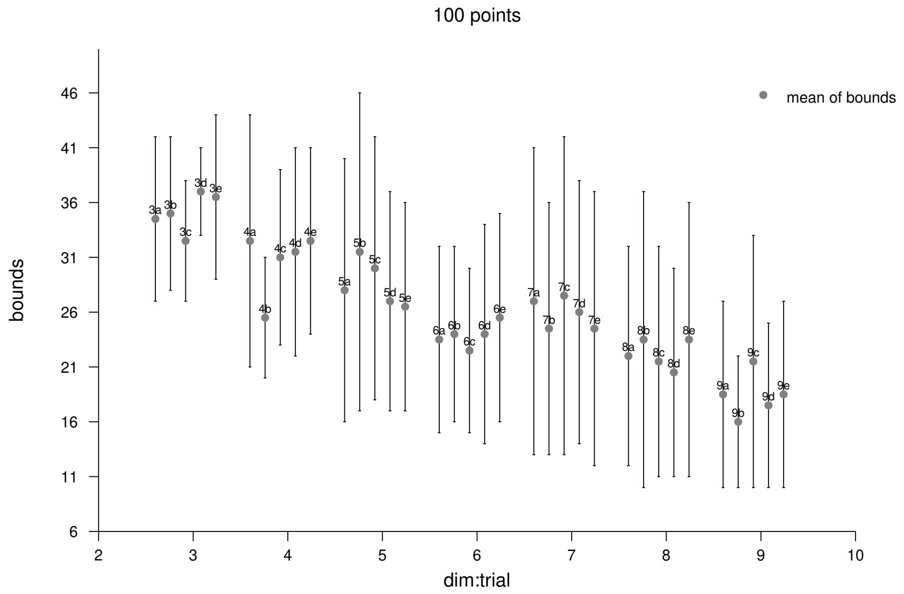

FIGURE 7. Gap between upper and lower bounds, dataset `n100`.