Computing Tukey depth based on linear programming

Pavlo Mozharovskyi
Institute of Econometrics and Statistics, University of Cologne
Albertus-Magnus-Platz, 50923 Cologne, Germany

November 13, 2014

Abstract

Determining the representativeness of a point within a data cloud has recently become a desirable task in multivariate analysis. The concept of statistical depth function, which reflects centrality of an arbitrary point, appears to be useful, and has been studied intensively in the last decades. Here the issue of computing the classical Tukey data depth is considered. The paper suggests an algorithm based on iterative application of linear programming. The algorithm exploits the idea of the cone segmentation of the multivariate space and allows for efficient implementation in applications due to the special search structure. A simulation study provides a comparison with the existing analog and gives an additional insight into the constituents of the algorithm.

Keywords: Tukey depth; Linear programming; Cone segmentation; Breadth-first search algorithm; Exact computation; Simplex algorithm.

1 Introduction

Determining the representativeness of a point within a bunch of data or a probability measure has recently become a desirable task in multivariate analysis. Nowadays it finds applications in different domains of economics, biology, geography, medicine, cosmology and many others. In his celebrated work, Tukey (1975) introduced an idea to order multivariate data, which has later been developed by Donoho & Gasko (1992) and is known as the *Tukey* (=*halfspace*, *location*) *depth*. Generally, the statistical data depth is a function determining how centrally a point is located in a data cloud. The upper-level sets it generates — trimmed regions — are set-valued statistics. They trim data w.r.t. the degree of centrality. For more information on the data depth the reader is referred to Zuo & Serfling (2000), Dyckerhoff (2004), Mosler (2013) and mentioned there references.

The Tukey data depth is one of the most important depth notions and is historically the first one. Regard a random vector X distributed as P, in particular empirically on $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$, in $\mathbb{R}^d$. The Tukey depth of a point $\mathbf{z} \in \mathbb{R}^d$ w.r.t. X, further $D(\mathbf{z}|X)$, is defined as the smallest probability mass of a closed halfspace containing $\mathbf{z}$:

$$D(\mathbf{z}|P) = \inf\{P(H) \,|\, H \text{ closed half-space}, \mathbf{z} \in H\}. \tag{1}$$

The Tukey depth possesses many desirable properties: it is affine invariant, tends to zero at infinity, is monotone on rays from any deepest point, quasiconcave and upper semicontinuous. By that it satisfies all the postulates imposed on a depth function (Zuo & Serfling, 2000, Dyckerhoff, 2004, Mosler, 2013). If P is absolutely continuous, the Tukey depth is a continuous function of $\mathbf{z}$ achieving maximum value of $\frac{1}{2}$, for angularly symmetric distributions at the center of symmetry. If P has no Lebesgue density, the Tukey depth is a discrete function of $\mathbf{z}$ and can have a non-unique maximum. By definition, its empirical version vanishes beyond the convex hull of the data. The Tukey depth determines uniquely empirical distribution (Koshevoy, 2002), taking a finite number of values in the interval from 0 (for the points lying outside the convex hull of the data) to $\frac{1}{2}$, increasing by a multiple of $\frac{1}{n}$. Naturally, it has attractive breakdown properties and converges for a sample from P almost surely to the depth w.r.t. P (Donoho & Gasko, 1992). The Tukey depth has a direct connection to such concepts as regression depth of Rousseeuw & Hubert (1999) and the separating hyperplane with the smallest empirical risk in binary supervised classification; it can be extended to functional settings (López-Pintado & Romo, 2011, Claeskens *et al.*, 2014).

For a data cloud $D(\mathbf{z}|X)$ can be expressed as the smallest portion of X to be cut off by a hyperplane through $\mathbf{z}$ so that the remaining points lie in an open halfspace not containing $\mathbf{z}$:

$$D(\mathbf{z}|X) = \frac{1}{n} \min_{\mathbf{r}\in S^{d-1}} \#\{i|\mathbf{x}_i'\mathbf{r} \geq \mathbf{z}'\mathbf{r}, \mathbf{x}_i \in X\}. \tag{2}$$

The Tukey depth is defined by the combinatorial structure of the data. For example, shifting X to get $\mathbf{z}$ in the origin and projecting X onto S^{d-1} after that does not influence the value of the depth. This coincides with the densest hemisphere problem, see Johnson & Preparata (1978). Exact calculation of the Tukey depth is a computationally challenging task of non-polynomial complexity. For this reason, great part of the literature on the Tukey depth concerns its computational aspects. The reader is referred to Liu & Zuo (2014a) for the exact algorithm and a reference to some preceding works. Dyckerhoff (2004) introduced the weak projection property, which is satisfied inter alia by the Tukey depth. This allows to approximate the depth as the minimum over univarite depths in the projections onto one-dimensional spaces. For the latest research in this area see Chen *et al.* (2013) and contained there references.

In the current paper an algorithm based on linear programming is introduced. Here, the idea of the conic segmentation of $\mathbb{R}^d$, introduced by Mosler *et al.* (2009) for constructing zonoid trimmed region, is exploited. It has been applied by Liu & Zuo (2014a) to computing the Tukey depth as follows. The entire space is divided into polyhedral cones, each having — in the projection onto any direction in its interior — the same subset of X above (below) the projection of $\mathbf{z}$, and thus delivering the same univariate Tukey depth. For each of the cones this depth value is calculated, and the Tukey depth is then the minimum over all these depths. For any cone, Liu & Zuo (2014a) employ the convex hull algorithm (Barber *et al.*, 1996) (further QHULL) to detect neighboring ones. Then, starting from an arbitrary cone, all cones are regarded by means of the breadth-first search algorithm.

In the proposed procedure, linear programming is used for finding cone's neighbors, and each cone is coded by a binary sequence. First, this gives possibility to examine candidates for the neighbors separately, and not at once as it is done when applying the QHULL algorithm. Second, the number of the candidates to be checked can be

substantially reduced. Third, when employing the binary coding, one does not need to find out where a cone is located in $\mathbb{R}^d$, which further saves computational expenses. Also, the calculations are performed in the spaces of dimension $d-1$ by a simplex algorithm. Finally, linear programming allows for caching by remembering (last) found basis.

The rest of the paper is organized as follows. Section 2 provides the theoretical results to the proposed algorithm, which is given in Section 3. Some experimental results regarding computation time are stated in Section 4, including a brief comparison of linear programming and the QHULL algorithm in light of the calculation of the Tukey depth. Section 5 concludes.

2 Theoretical background

Given a data sample $X = \{\mathbf{x}_1, ..., \mathbf{x}_n\} \in \mathbb{R}^d$, $d < n$, and a point $\mathbf{z} \in \mathbb{R}^d$, the Tukey depth of $\mathbf{z}$ w.r.t. X shall be calculated. We assume w.l.o.g. that $\mathbf{z} = \mathbf{0}$ and that $\{\mathbf{z}\} \cup X$ are in general position, i.e., every subset of $k+1$ points $\in (\{\mathbf{z}\} \cup X)$ spans a subspace of dimension $\min\{k, d\}$. Violation of these assumptions can be compensated by a location shift and a slight perturbation of the data. The Tukey depth is discrete, so such a perturbation can be potentially harmful, as only a small shift of one point can change the depth value of $\mathbf{z}$ in a non-continuous way. Before performing such a perturbation, we suggest to first check whether $\mathbf{z} \in \mathrm{conv}(X)$ (if not, $D(\mathbf{z}|X) = 0$), and only then calculate the depth of $\mathbf{z}$ using perturbed data. When n is not very small and the zero-depth case is specially treated, possible perturbation damage is negligible.

Consider a direction, i.e. a point on the unit sphere $\mathbf{r} \in S^{d-1}$. It yields an ordered sequence, a permutation $\pi_\mathbf{r}$ on $\mathcal{N} = \{1, ..., n\}$ such that $\mathbf{x}'_{\pi_\mathbf{r}(1)}\mathbf{r} \leq \mathbf{x}'_{\pi_\mathbf{r}(2)}\mathbf{r} \leq ... \leq \mathbf{x}'_{\pi_\mathbf{r}(n)}\mathbf{r}$. If the data are in general position a vector $\mathbf{r}$ can be found such that all inequalities hold strictly $\mathbf{x}'_{\pi_\mathbf{r}(1)}\mathbf{r} < \mathbf{x}'_{\pi_\mathbf{r}(2)}\mathbf{r} < ... < \mathbf{x}'_{\pi_\mathbf{r}(n)}\mathbf{r}$, and $\mathbf{x}'_{\pi_\mathbf{r}(i)}\mathbf{r} \neq 0, i = 1, ..., n$. Then such $\mathbf{r}$ splits X into two disjoint subsets (by its normal hyperplane $H_\mathbf{r}$ through $\mathbf{0}$ yielding two open halfspaces $H_\mathbf{r}^+$ and $H_\mathbf{r}^-$ in $\mathbb{R}^d$), $X_\mathbf{r}^+ = \{\mathbf{x} \in X | \mathbf{x}'\mathbf{r} > 0\}$ and $X_\mathbf{r}^- = \{\mathbf{x} \in X | \mathbf{x}'\mathbf{r} < 0\}$ containing the points with strictly positive, respectively negative, projections on $\mathbf{r}$. Let us call the closure of the set of all $\lambda\mathbf{r}, \lambda \geq 0$, maintaining the same $X_\mathbf{r}^+$ and $X_\mathbf{r}^-$, a *direction cone* C (yielding $X_C^+ = \{\mathbf{x} \in X | \mathbf{x}'\mathbf{r} > 0 \, \forall \, \mathbf{r} \in \mathrm{int}(C)\}$ and $X_C^- = \{\mathbf{x} \in X | \mathbf{x}'\mathbf{r} < 0 \, \forall \, \mathbf{r} \in \mathrm{int}(C)\}$ respectively). This is because its form constitutes an infinite polyhedral cone with the apex in the origin. The entire $\mathbb{R}^d$ is then filled by the set of all direction cones, say $\mathcal{C}(X)$, while each cone $C \in \mathcal{C}(X)$ defines some portion of the sample, that can be cut off by the hyperplane normal to any $\mathbf{r} \in \mathrm{int}(C)$. Denote this portion $D_C(\mathbf{0}|X) = \frac{1}{n}\min\{\sharp X_C^+, \sharp X_C^-\}$ ($\sharp$ stands for the set's cardinality), then the Tukey depth is $D(\mathbf{0}|X) = \min_{C \in \mathcal{C}(X)} D_C(\mathbf{0}|X)$. Below $\mathcal{C}(X)$ will be mentioned as *cone segmentation*; see Figure 1 left for a cone segmentation on the unit cube for ten standard normal deviates. One of the direction cones can be seen in the unit cube's corner directed to the reader.

The further task is then to go through all such cones and to find the one(s) delivering the smallest $\frac{1}{n}\min\{\sharp X_C^+, \sharp X_C^-\}$, i.e., the Tukey depth. Starting with Mosler *et al.* (2009), the usual way to proceed is:

(1) choose an arbitrary direction cone,

(2) move from each direction cone to the neighbors,

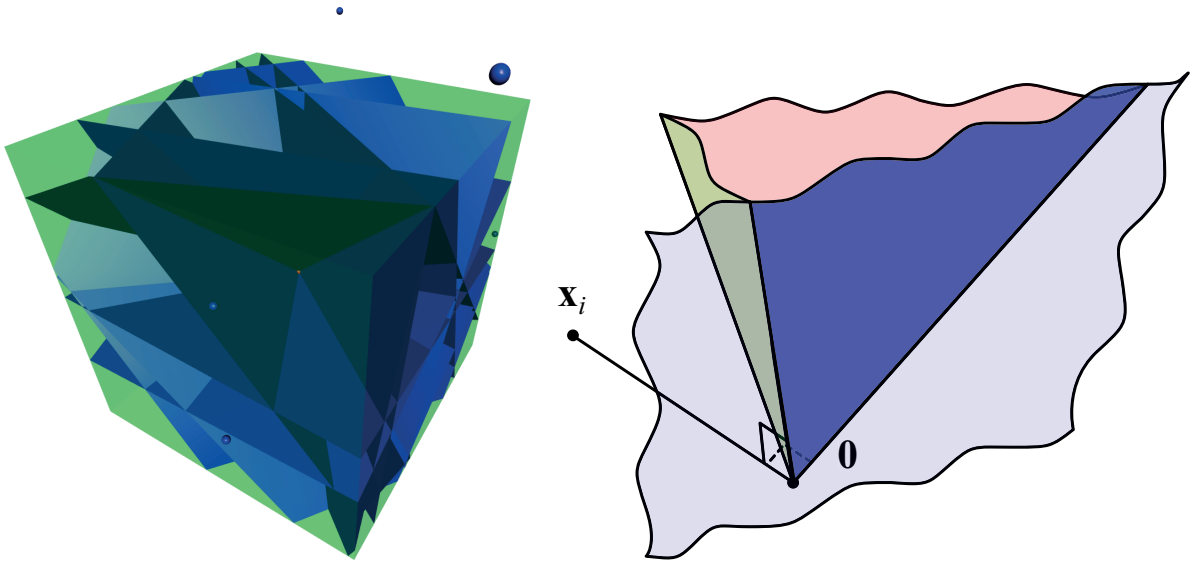

Figure 1: Cone segmentation on the unit cube (left) and a cone's facet defined by a point (right)

(3) by that cover the entire $\mathbb{R}^d$ using breath-first search algorithm,

(4) on each step check whether a direction cone has already been considered, i.e. saved in a structure maintaining fast search (usually a binary search tree).

Ad (1), the task is trivial: a direction $\mathbf{r} \in S^{d-1}$ maintaining the ordering with strict inequalities $\mathbf{x}'_{\pi_\mathbf{r}(1)}\mathbf{r} < \mathbf{x}'_{\pi_\mathbf{r}(2)}\mathbf{r} < ... < \mathbf{x}'_{\pi_\mathbf{r}(n)}\mathbf{r}$ and no projection coinciding with $\mathbf{z}'\mathbf{r} = 0$ has to be generated. When drawing $\mathbf{r}$ randomly, the theoretical probability of this event $= 1$. As in practice draw concerns only a finite number of digits, it can (though extremely rarely) happen that one needs more than one drawing.

2.1 Identification of neighboring cones

Ad (2), identifying neighboring direction cones (2a) and transition to each of them if new (2b) is to be done. Let us take a closer look at the direction cone. Two different cones C_1 and C_2 differ in their corresponding set pairs $(X^+_{C_1}, X^-_{C_1})$ and $(X^+_{C_2}, X^-_{C_2})$. So, if a point $\mathbf{r} \in S^{d-1}$ moves from one direction cone to another, projections of one or more points on $\mathbf{r}$ *migrate passing the origin*, i.e., change the sign. Let C_1 and C_2 be two cones, such that a direct (i.e., not crossing other cones) rotational movement of $\mathbf{r}$ from C_1 to C_2 (and vice-versa) is possible. That means that C_1 and C_2 have an intersection of affine dimension between 1 and $d-1$. If transition of $\mathbf{r}$ from C_1 to C_2 involves changing the halfspace (from $H^+_\mathbf{r}$ to $H^-_\mathbf{r}$ or vice versa) by one point $\in X$ only (correspondingly changing the sign of its projection on $\mathbf{r}$), then C_1 and C_2 intersect in affine dimension $d-1$. This intersection constitutes the cones' common facet. We call such two cones *neighboring cones*.

So, the transition of a single point $\mathbf{x}_i \in X$ from $X^+_{C_1}$ to $X^-_{C_2}$ means traversing of $\mathbf{r}$ from C_1 to a neighboring cone C_2 through a facet, and thus the facet is defined by

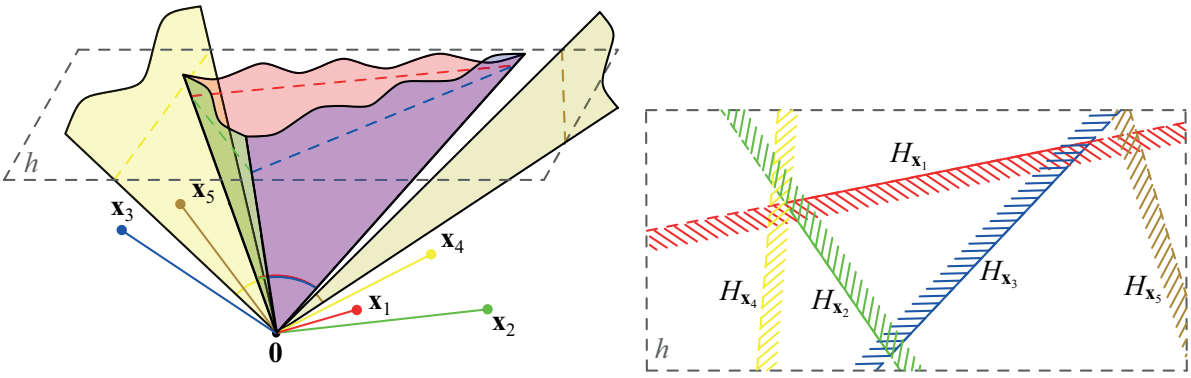

Figure 2: A direction cone in $\mathbb{R}^3$ defined by the points $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$, halfspaces formed by $\mathbf{x}_4$ and $\mathbf{x}_5$ are not directly involved (left); arbitrary cutting hyperplane h visualizing how the hyperplanes are involved (right).

this point $\mathbf{x}_i$, see Figure 1 right. Naturally, given a cone C, any facet of C lies in a hyperplane, normal to the line, connecting a point $\in X$ with $\mathbf{z} = \mathbf{0}$, as it is shown in Figure 1 right, but not each point $\in X$ generates a facet of C, see Figure 2. A direction cone C is defined by the intersection of closed halfspaces $\{\mathbf{y}|\mathbf{y}'(\mathbf{x}-\mathbf{z}) \geq 0, \mathbf{x} \in X_C^+\}$ and $\{\mathbf{y}|\mathbf{y}'(\mathbf{x}-\mathbf{z}) \leq 0, \mathbf{x} \in X_C^-\}$. Hyperplanes directly involved in the intersection (generated by points $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ in Figure 2) contain the cone's facets and those outside (generated by points $\mathbf{x}_4, \mathbf{x}_5$ in Figure 2) do not. Thus, given a direction cone, a natural question is: Which points $\in X$ define its facets, and which do not? This is summarized in Theorem 1.

Theorem 1 *Given $X = \{\mathbf{x}_1, ..., \mathbf{x}_n\} \in \mathbb{R}^d$, assume that $\{\mathbf{0}\} \cup X$ are in general position, and let C be a direction cone. Also, for a point $\mathbf{x} \in X$ let $X_{H_\mathbf{x}}$ be the orthogonal projection of X onto the $(d-1)$-dimensional linear subspace $H_\mathbf{x}$ normal to $\mathbf{x}$, and $X^+_{H_\mathbf{x},C}$ and $X^-_{H_\mathbf{x},C}$ be the two subsets of $X_{H_\mathbf{x}} \setminus \{\mathbf{0}\}$ corresponding to X_C^+ and X_C^-, respectively. Then:*

(i) $H_\mathbf{x}$ contains a facet of C if and only if $X^+_{H_\mathbf{x},C}$ and $X^-_{H_\mathbf{x},C}$ are linearly strictly separable through $\mathbf{0} \in \mathbb{R}^{d-1}$, i.e., can be separated by a (d-2)-hyperplane $\subset H_\mathbf{x}$ containing the origin and no points from $X^+_{H_\mathbf{x},C} \cup X^-_{H_\mathbf{x},C}$,

(ii) if $\mathbf{r} \in S^{d-1}$ moves from C to a neighboring direction cone through a facet $\subset H_\mathbf{x}$, the projection of $\mathbf{x}$ only on the line through $\mathbf{r}$ changes sign.

Proof:

(i) "$\Longrightarrow$": If $\mathbf{x} \in X$ defines a facet of C, then $H_\mathbf{x}$ contains this facet, and thus there should exist some direction $\mathbf{v} \in S^{d-1} \cap H_\mathbf{x}$ such that X_C^+ and X_C^- projected onto $\mathbf{v}$ maintain their signs, except for the single point $\mathbf{x}$ being projected into $\mathbf{0} \in \mathbb{R}^{d-1}$. So, these projections of $X_C^+ \setminus \{\mathbf{x}\}$ and X_C^- (or X_C^+ and $X_C^- \setminus \{\mathbf{x}\}$) are separated in $H_\mathbf{x}$ by the hyperplane normal to $\mathbf{v}$ through $\mathbf{0}$.

"$\Longleftarrow$": Strict linear separability of $X^+_{H_\mathbf{x},C}$ and $X^-_{H_\mathbf{x},C}$ through $\mathbf{0}$ means that there exists some $\mathbf{v} \in S^{d-1} \cap H_\mathbf{x}$, such that $\mathbf{x}'\mathbf{v} > 0 \ \forall \ \mathbf{x} \in X^+_{H_\mathbf{x},C}$ and $\mathbf{x}'\mathbf{v} < 0 \ \forall$

$\mathbf{x} \in X^-_{H_\mathbf{x},C}$. Then a slight infinitesimal rotation of $\mathbf{v}$ towards (and inside of) the cone does not cause the projections to change signs, and thus maintains X^+_C and X^-_C.

(ii) Let $\mathbf{x}$ define a common facet of C and C'. As $\mathbf{x}$ is projected into $\mathbf{0} \in \mathbb{R}^{d-1}$ for all $\mathbf{r} \in S^{d-1} \cap H_\mathbf{x}$, then obviously when (slightly) deviating $\mathbf{v}$ to different sides of $H_\mathbf{x}$, the signs of $\mathbf{x}'\mathbf{v}$ will be opposite. All points $\in \{\lambda\mathbf{x}, \lambda \in \mathbf{R}\}$ change sign in their projection on $\mathbf{v}$, but as $\{\mathbf{0}\} \cup X$ are in general position, $\mathbf{x}$ is the only one.

2.2 Optimization of the breadth-first search algorithm

In Section 2.1 we have addressed (2a) and (2b) by Theorem 1. From the first part, one can easily find out which points define the cone's facets. Then, following the second part, moving the direction $\mathbf{r}$ to a neighboring cone by traversing their common facet constitutes in changing sign of the projection on $\mathbf{r}$ of the point which defines this facet.

Ad (3), we use the results from above to describe the *breadth-first search* algorithm: generate an initial direction cone (ad (1)) and move to the neighboring cones (ad (2)), calculating the depth in each of them, till the entire $\mathbb{R}^d$ is covered. Note, that covering a cone segmentation of $\mathbb{R}^d$ by a breadth-first search is general for some depth-calculating algorithms (Liu & Zuo (2014a,b)) and algorithms constructing trimmed regions (Mosler *et al.* (2009), Paindaveine & Šiman (2012a,b), Bazovkin & Mosler (2012)) when $d \geq 3$. Below the algorithm is summarized to be referenced it in further explanations. The algorithms of Paindaveine & Šiman (2012a,b), Liu & Zuo (2014a,b) differ from this one in that they store cones' facets and not cones while employing the convex hull algorithm.

The *breadth-first search algorithm* on a cone segmentation of $\mathbb{R}^d$ proceeds in following steps:

(a) Draw an initial cone and store it in a queue.

(b) Pop one cone from the head of the queue, process it, remember it, and for each of its neighboring cones do:

  (c) If the cone has not been processed till now push it into the tail of the queue.

(d) If the queue is not empty, go to Step (b).

Further, let us introduce the notion of the cone's *generation*, a number given to each cone (in Step c) when it is pushed into the queue. The initially drawn cone (in Step a) is given the initial number, say 1. (The generation can be thought of as the 'depth' of the current searching path of the algorithm.) Obviously, when covering a cone segmentation of $\mathbb{R}^d$ with the breadth-first search algorithm, for processing cones of the i-generation, only cones of the $(i-1)$-, i- and $(i+1)$-generation have to be remembered. While on starting (low) generations the number of the cones from one generation to another grows rapidly, on close to 'equatorial' generations (these basically constitute the segmentation) the increase is much less. Also, though the store-search structure for the cones is usually a binary tree, the computational time for search can be saved either, especially when the search is frequently performed.

Ad (4): When calculating the Tukey depth under the general position assumption of $\{\mathbf{z}\} \cup X$, one can step much further in this direction, and save less than tree generations. First, to simplify further presentation, let us code the cones. As mentioned above, the

interior of each cone C maintains the disjoint division of X into X_C^+ and X_C^- according to the signs in X's projection onto any $\mathbf{r} \in \text{int}(C)$, and thus is uniquely defined by this division. So, binary identifiers for the cones can be used: a cone is coded by a binary sequence ("0" and "1" say) of length n, where each bit represents a point $\in X$ w.r.t. some initial ordering of the points $\in X$ that is kept constant during the entire procedure. Points belonging to X_C^+ are coded by "1", those belonging to X_C^- by "0".

After coding the initial cone (C_0 say) this way, other cones can be coded either the same way, or by another binary sequence identifying whether a point has changed the sign w.r.t. C_0 ("1") or not ("0"). Then any cone's code can be obtained as the code of C_0 with those bits inverted that have been switched on in this sequence. This leads to Lemma 1.

Lemma 1 *Let us start the breadth-first search algorithm with an arbitrary initial cone C_0, and in Step b, when checking for neighboring cones, always regard only cones defined by points which have not changed their sign in the projection onto $\mathbf{r} \in int(C_0)$ yet. Then in processing cones of the i-th generation, only cones of the i-th generation have to be remembered to check for neighboring cones and of the $(i+1)$-th generation to check whether a new cone has already been seen.*

Proof: If the cones defined by already processed points, i.e. those having changed their sign in the projection, are not considered, then only cones of the $(i+1)$-th generation can be taken into account when deciding whether a cone has already been seen. No cones of the $(i-1)$-th or i-th generation can be found because points defining them are not checked at all. Then one can go through all the cones of the i-th generation, and add those newly found from the $(i+1)$-th generation to the queue.

Theorem 1 (ii) and Lemma 1 lead to Lemma 2. Note that $\lfloor u \rfloor$ stands for the largest integer $\leq u$.

Lemma 2 *When starting the breadth-first search algorithm with an arbitrary initial cone C_0, and in Step b regard only neighboring cones defined by points which have not changed their sign in the projection onto $\mathbf{r} \in int(C_0)$ yet, only $\lfloor \frac{n+2}{2} \rfloor$ generations have to be considered.*

Proof: From Theorem 1 (ii), each point may define a cone's facet, changing its sign in projections on all directions of the neighboring cone. If, following Lemma 1, on each new step only not yet considered points are taken into account, then in each new generation exactly one point more has its sign on projection changed (compared to C_0). The maximum generation (if C_0 is denoted as 1st generation) is then $(n+1)$-th generation.

Each cone has its mirror-copy cone, where projections of X on all directions have exactly opposite signs; these cones need not be considered, of course. Then, if n is odd, exactly $\frac{n+1}{2}$ generations have to be considered, if n is even, $\lfloor \frac{n+1}{2} \rfloor$ +1 generations have to be considered, as the mirror-copy cones of the 'equatorial' (having number $\lfloor \frac{n+1}{2} \rfloor + 1$) generation also belong to the equatorial generation. Thus, at most $\lfloor \frac{n+2}{2} \rfloor$ generations have to be regarded.

3 Algorithm

Basically, Algorithm 1 is the application of the breadth-first-search algorithm for searching over the direction cones covering the entire $\mathbb{R}^d$. We will need some notation. As described above, let $b^{\mathbf{r}}$ be a binary sequence of length n where each bit $b^{\mathbf{r}}(i)$ corresponds to a point $\mathbf{x}_i \in \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\} = X$ with $b^{\mathbf{r}}(i) = I(\mathbf{x}_i'\mathbf{r} > 0)$ for any $\mathbf{r}$ that maintains strict ordering of $\mathbf{x} \in X$ in the projection on it. Also, let b_i^0 be a zero-filled binary sequence with the i-th bit set to "1", $\oplus$ denote the binary 'exclusive disjunction'="XOR" operation, ! be the bit inversion operator, and $\sum b^{\mathbf{r}}$ be the number of "1"s in $b^{\mathbf{r}}$ (Hamming distance between $b^{\mathbf{r}}$ and b^0).

Algorithm 1 Input: *$X = \{\mathbf{x}_1, ..., \mathbf{x}_n\} \in \mathbb{R}^d$, $d < n$, $\{\mathbf{0}\} \cup X$ in general position.*

1. *Initialization: Calculate $X_{H_{\mathbf{x}_i}} = \{\mathbf{x}_1^{H_{\mathbf{x}_i}}, \mathbf{x}_2^{H_{\mathbf{x}_i}}, ..., \mathbf{x}_n^{H_{\mathbf{x}_i}}\}, i = 1, ..., n$, set $D = n$. Draw $\mathbf{r}_0 \in S^{d-1}$ yielding a permutation $\pi_{\mathbf{r}_0}$ on $\mathcal{N} = \{1, 2, ..., n\}$ maintaining strict order $\mathbf{x}'_{\pi_{\mathbf{r}_0}(1)}\mathbf{r}_0 < \mathbf{x}'_{\pi_{\mathbf{r}_0}(2)}\mathbf{r}_0 < ... < \mathbf{x}'_{\pi_{\mathbf{r}_0}(n)}\mathbf{r}_0$ and let $b^{\mathbf{r}_0}$ be the corresponding binary code. Initialize $B = \{b_1, b_2, ..., b_n\}$ with $b_i = b^{\mathbf{r}_0}\ \forall\, i = 1, ..., n$. Initialize a queue $\mathcal{B}_{topical}$ containing $b^{\mathbf{r}_0}$ only and an empty searchable storage $\mathcal{B}_{future}$ (e.g., binary tree).*

2. *For $i = 1 : n$ do:*

   (a) *For $j = 1 : n$ do:*

      i. *If $b^{\mathbf{r}_0}(j) = 0$ then $\mathbf{x}_j^{H_{\mathbf{x}_i}} = -1 \cdot \mathbf{x}_j^{H_{\mathbf{x}_i}}$.*

3. *For $i = 1 : \lfloor \frac{n+2}{2} \rfloor$ do:*

   (a) *Pop b = head of $\mathcal{B}_{topical}$, $D = \min\{D, \sum b, n - \sum b\}$.*

   (b) *If $i = \lfloor \frac{n+2}{2} \rfloor$, then go to Step 3d.*

   (c) *For $j = 1 : n$ do:*

   If $(b \oplus b^{\mathbf{r}_0})(j) = 0$ then

      i. *For $k = 1 : n$ do:*

         If $(b_j \oplus b)(k) = 1$ then $\mathbf{x}_k^{H_{\mathbf{x}_j}} = -1 \cdot \mathbf{x}_k^{H_{\mathbf{x}_j}}$, $b_j(k) =!b_j(k)$.

      ii. *If (i) $\mathbf{0} \in conv(X_{H_{\mathbf{x}_j}} \setminus \{\mathbf{0}\})$ and (ii) $(b \oplus b_j^0) \notin \mathcal{B}_{future}$*

         then add $(b \oplus b_j^0)$ to $\mathcal{B}_{future}$.

   (d) *If $\mathcal{B}_{topical} \neq \varnothing$, then go to Step 3a, else $\mathcal{B}_{topical} = \mathcal{B}_{future}$, $\mathcal{B}_{future} = \varnothing$.*

4. **Return:** *D/n.*

Nontrivial is the check of condition (i) in Step 3(c)ii, i.e. whether $\mathbf{0} \in \text{conv}(X_{H_{\mathbf{x}_j}} \setminus \{\mathbf{0}\})$ ($\mathbf{x}_j$ is projected into $\mathbf{0}$ in $H_{\mathbf{x}_j}$; it is excluded). In other words, given a cone C unambiguously defined by the corresponding b_j, the linear separability (through the origin) of $X^+_{H_{\mathbf{x}_j},C}$ and $X^-_{H_{\mathbf{x}_j},C}$ has to be checked, i.e. whether $\exists\, \mathbf{r} \in S^{d-1} \cap H_{\mathbf{x}_j}$ such that $\mathbf{r}'\mathbf{x} > 0\ \forall\ \mathbf{x} \in X^+_{H_{\mathbf{x}_j},C}$ and $\mathbf{r}'\mathbf{x} < 0\ \forall\ \mathbf{x} \in X^-_{H_{\mathbf{x}_j},C}$. This can be done by means of linear programming as follows.

Let $\mathbf{Y}$ be the $(n-1) \times (d-1)$ matrix, which rows are the points $\in (X_{H_{\mathbf{x}_j}} \setminus \{\mathbf{0}\})$ for an iteration of Step 3(c)ii of the algorithm. The task from above narrows down to finding a feasible solution Λ^0 satisfying the constraints:

$$\begin{aligned} \mathbf{Y}'\Lambda &= \mathbf{0}_{d-1}, \\ \Lambda'\mathbf{1}_{n-1} &= 1, \\ \Lambda &\geq \mathbf{0}_{n-1}, \end{aligned}$$

with $\Lambda = (\lambda_1, ..., \lambda_{n-1})'$ and $\mathbf{0}_k$ ($\mathbf{1}_k$) being a vector-column of k zeros (ones). This is what is done in the first phase of the simplex algorithm.

In Step 1 the $X_{H_{\mathbf{x}_i}}, i = 1, ..., n$ — projections of X onto zero hyperplanes normal to data points $\in X$ — are cached, an on each following step of the Algorithm for each i these projections signs of several points have to be changed only, which computationally is a cheap operation. Please note, that the simplex algorithm is executed in these hyperplanes, i.e. in dimension $d-1$. This mechanism allows for further caching as well. If on Step 3(c)ii for some j $\mathbf{0} \in \text{conv}(X_{H_{\mathbf{x}_j}} \setminus \{\mathbf{0}\})$, a basis consisting of d points will be found. If, on the next iteration of the Algorithm, on Step 3(c)i for the same j the points changing sign do not belong to the previously found basis, clearly $\mathbf{0} \in \text{conv}(X_{H_{\mathbf{x}_j}} \setminus \{\mathbf{0}\})$ again, an no new execution of the simplex algorithm is needed. A more complicated caching scheme can be used here, though. One can see in Step 3, that the outer cycle of the Algorithm is completely deterministic and is always executed $\lfloor \frac{n+2}{2} \rfloor$ iterations only, independent of the exact positioning of the data.

4 Experiments

In this section we give a short experimental reference on the computational efficiency of the developed algorithm. First, in Section 4.1 we compare the computational load of the algorithm with the existing analog from Liu & Zuo (2014a). Second, in Section 4.2 the convex-hull-constructing algorithm is contrasted with linear programming for the task of computation of the Tukey depth.

4.1 Execution time

Table 1 indicates the execution times of the proposed algorithm (line 'LP') and this of Liu & Zuo (2014a) (line 'QHULL') when calculating the Tukey depth of the origin w.r.t. a sample of cardinality n from $X \sim N(\mathbf{0}_d, \mathbf{I}_d)$, with $\mathbf{0}_d$ being a vector of length d consisting of 0s and $\mathbf{I}_d$ being the diagonal matrix of 1s of dimension d. The grid of n and d values coincides with this used by Liu & Zuo (2014a). Unlike Liu & Zuo (2014a) we do not compute the depth of the further points, because all the direction cones have to be checked anyway, and, under the assumption of general position, their number depends on d and n only (in fact it equals $2\sum_{i=0}^{d-1} \binom{n-1}{i}$). The experiments from Liu & Zuo (2014a) and a few runs of our algorithm show that these time differences are rather small. The time of each algorithm has been averaged over ≤ 10 execution, and one experiment for each pair (n, d) and for each algorithm never last longer than 24 hours. Thus, e.g., for $n = 80$ and $d = 6$ only two executions of the proposed algorithm have been done. The '—' sign indicates that an algorithm was not able to compute the depth at least one time during a day. Both algorithms have been implemented in C++.

For the execution times of the Matlab implementation of the algorithm of Liu & Zuo (2014a) the reader is referred to their paper. In all the experiments a single kernel of processor Core i7-2600 (3.4 GHz) have been used, accessing 16 GB physical memory.

One can conclude that in most of the considered cases the proposed algorithm is faster, especially in higher dimensions. On the other hand for $d = 3$ starting with $n = 320$ points it is slower than the competitor, and for $d = 4$ the difference in the execution times becomes smaller when n increases. This can be explained by application of the convex hull algorithm, which is faster in identifying the convex hull than linear programming. That is, for given d, if n is sufficiently large, the algorithm of Liu & Zuo (2014a) outperforms the proposed one. But given d, how large should this n be? Fort this d, is the depth w.r.t. a sample of such size n computable in a reasonable time at all? To give some insights we conduct a comparison of the linear programming and of the convex hull algorithm for identifying the convex hull of a data cloud in Section 4.2.

4.2 Linear programming *vs.* QHULL

To take a closer look at the behavior of the basic constituents of the both algorithms discussed above, we compare the execution times of identifying the points forming the convex hull of a data cloud by linear programming and by the convex hull algorithm. When obtaining these via linear programming each point is checked whether it lies in the convex of the rest; for the convex hull implementation downloaded from `www.qhull.org` is employed. For the both algorithms C++ implementations have been used. The data cloud is generated from $N(\mathbf{0}_d, \mathbf{I}_d)$, and all the experimental settings are as before. (As the number of the vertices of the convex hull of the data cloud from the normal distribution is rather moderate, the convex hull algorithm is slightly favored.)

Table 2 presents the corresponding execution times for $3 \leq d \leq 10$ and $n = 40, 80, ..., 20480$ for linear programming (line 'LP') and for the convex hull algorithm (line 'QHULL'). The sign '—' denotes the situation when the whole available physical memory has been consumed. Clearly, for given d, the convex hull algorithm outperforms linear programming if $n \geq n_d^{thr.}$. From Table 2, $n_d^{thr.} < 40$ for $d < 6$, $640 < n_6^{thr.} < 1280$, $2560 < n_7^{thr.} < 5120$ and $20480 < n_d^{thr.}$ for $d > 7$. As each of these should be executed for each direction cone, $n_d^{thr.}$ for the entire depth-computing algorithm can be very large, what explains why the proposed one shows satisfactory times. One should notice that, for fixed n, time increase with d is much lower when the linear programming is used. Also, as discussed above, linear algorithm is not executed n times for each direction cone (due to the special cone-coding scheme), as it was done in the current experiment, and it is executed in dimension $d - 1$. In addition, we employ cashing of the basis for the simplex algorithm, which further reduces the number of the executions. All this proves the reasonability of the proposed algorithm.

5 Conclusions and outlook

The paper presents an algorithm computing the Tukey depth by finding a global minimum over a finite range of variants. The task of computing the Tukey depth is NP-complete while all separations of X into two subsets by hyperplanes through $\mathbf{z}$ are regarded. The algorithm follows the traditions of the cone segmentation of a finite-dimensional space and regards candidate hyperplanes for the Tukey depth according to

Table 1: Execution times (in seconds) of the proposed algorithm (line 'LP') and of this from Liu & Zuo (2014a) (line 'QHULL') when computing Tukey depth of the origin w.r.t. a d-variate standard normal data cloud of n points.

d	Algorithm	$n = 40$	80	160	320	640	1280	2560
3	LP	0.028	0.228	1.888	17.371	174.744	1789.335	18436.420
	QHULL	0.072	0.367	2.186	15.179	103.763	830.077	17372.290
4	LP	0.403	7.022	119.010	2035.505	35924.400	—	—
	QHULL	3.133	77.007	1880.025	57512.800	—	—	—
5	LP	4.752	174.848	5974.114	—	—	—	—
	QHULL	424.366	55674.000	—	—	—	—	—
6	LP	48.170	3877.931	—	—	—	—	—
	QHULL	39176.570	—	—	—	—	—	—
7	LP	368.112	67897.800	—	—	—	—	—
	QHULL	—	—	—	—	—	—	—
8	LP	2441.332	—	—	—	—	—	—
	QHULL	—	—	—	—	—	—	—
9	LP	12703.730	—	—	—	—	—	—
	QHULL	—	—	—	—	—	—	—
10	LP	58767.700	—	—	—	—	—	—
	QHULL	—	—	—	—	—	—	—

Table 2: Execution times (in seconds) of linear programming (line 'LP') and of the convex hull algorithm (line 'QHULL') when identifying the convex hull of a d-variate standard normal data cloud of n points.

d	Algorithm	$n = 40$	80	160	320	640	1280	2560	5120	10240	20480
3	LP	0.000	0.001	0.003	0.010	0.040	0.164	0.648	2.574	10.448	41.724
	QHULL	0.000	0.000	0.000	0.000	0.001	0.000	0.001	0.001	0.003	0.005
4	LP	0.000	0.001	0.003	0.012	0.044	0.175	0.722	2.997	11.823	46.577
	QHULL	0.000	0.000	0.000	0.001	0.001	0.002	0.003	0.005	0.007	0.013
5	LP	0.000	0.000	0.004	0.015	0.063	0.222	0.875	3.666	14.285	57.347
	QHULL	0.001	0.002	0.003	0.005	0.010	0.014	0.021	0.030	0.045	0.066
6	LP	0.000	0.001	0.005	0.019	0.075	0.288	1.163	4.569	18.085	71.154
	QHULL	0.003	0.010	0.023	0.046	0.087	0.162	0.274	0.439	0.675	0.985
7	LP	0.001	0.002	0.006	0.023	0.100	0.378	1.466	5.772	22.676	90.108
	QHULL	0.013	0.053	0.150	0.403	0.844	1.659	3.044	4.385	7.513	11.532
8	LP	0.000	0.002	0.006	0.027	0.113	0.431	1.750	6.844	26.608	104.440
	QHULL	0.044	0.344	1.347	3.478	8.457	16.867	35.086	56.434	102.224	188.282
9	LP	0.001	0.002	0.009	0.032	0.138	0.531	2.472	8.875	38.024	136.243
	QHULL	0.088	1.228	6.678	26.489	68.860	175.306	358.894	740.124	1507.160	—
10	LP	0.002	0.002	0.010	0.040	0.159	0.663	2.808	11.029	43.587	172.089
	QHULL	0.306	5.044	33.808	164.224	544.704	1454.072	—	—	—	—

a first-breadth order of direction cones. It employs the initial idea of Liu & Zuo (2014a) by identifying a facet using linear programming, and by exploiting the fact that each point $\in X$ changes the halfspace only once during the entire execution of the breadth-first search algorithm. Linear programming is executed in $\mathbb{R}^{d-1}$ and the found basis can be cached for each of these n $(d-1)$-dimensional projections. Also, binary coding of the cones does not require their spacial positioning. This yields a substantial acceleration. The algorithm saves physical memory by storing only two layers of the direction cones in RAM, too.

The algorithm presented here can be modified to solve related tasks, such as computing regression depth (Rousseeuw & Hubert, 1999) or finding a linear classification rule separating two training classes with a minimal number of errors (=empirical risk). Ghosh & Chaudhuri (2005) investigate the connection between the Tukey (also regression) depth and binary supervised classification. In a different way, the algorithm can be used for finding a hyperplane through a fixed point minimizing empirical risk. When adding an artificial coordinate equaling zero for all observations (i.e. yielding $(\mathbf{x}_i', 0)'$, $i = 1, ..., n$) and letting the hyperplane go through $(\mathbf{0}_d', 1)'$ say, its $(d-1)$-dimensional trace achieves the risk minimizing separation. After removing erroneous points, an optimal margin classifier (Boser *et al.*, 1992) can be applied to find the optimal separation hyperplane. By a (say, polynomial) extension of the space nonlinear classification rules may be involved.

The ideas considered in this paper can be applied to a wider range of tasks. Thus, the way of covering the space by a breadth-first search algorithm can be applied to many tasks involving a cone segmentation. Application of the linear programming, used here, can be a good alternative to the QHULL algorithm (used by Paindaveine & Šiman (2012a,b), Liu & Zuo (2014a,b)), while it allows to check whether a single point is a vertex of the convex hull of a data cloud. For instance, in the current algorithm, close to the equatorial generations, after filtering already seen points and unreachable neighbors (of the same generation), the number of points to be checked is almost halved.

Acknowledgements

The author wants to thank Xiaohui Liu for providing the Matlab source code of the algorithm from the work by Liu & Zuo (2014a) and for his insightful comments on the earlier version of the manuscript. The author is grateful to his supervisor Karl Mosler for his valuable comments on the earlier version of the paper and to his colleagues Rainer Dyckerhoff and Pavel Bazovkin for their helpful remarks.

References

Barber C.B., Dobkin, D.P. and Huhdanpaa, H. (1996). The Quickhull algorithm for convex hulls. *Computational Geometry* **46** 566–573.

Boser, B.E., Guyon, I.M. and Vapnik, V.N. (1992). A training algorithm for optimal margin classifiers. In: Haussler, D. (Ed.), *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, ACM Press, New York, 144–152.

Bazovkin, P. and Mosler, K. (2012). An exact algorithm for weighted-mean trimmed regions in any dimension. *Journal of Statistical Software* **47**.

Chen, C., Morin, P. and Wagner, U. (2013). Absolute approximation of Tukey depth: Theory and experiments. *Computational Geometry* **46** 566–573.

Claeskens, G., Hubert, M., Slaets, L. and Vakili K. (2014). Multivariate functional halfspace depth. *Journal of the American Statistical Association* **109** 411–423.

Donoho, D.L. and Gasko, M. (1992). Breakdown properties of location estimates based on halfspace depth and projected outlyingness. *The Annals of Statistics* **20** 1803–1827.

Dyckerhoff, R. (2004). Data depths satisfying the projection property. *AStA - Advances in Statistical Analysis* **88** 163–190.

Ghosh, A.K. and Chaudhuri, P. (2005). On data depth and distribution free discriminant analysis using separating surfaces. *Bernoulli* **11** 1–27.

Johnson, D.S. and Preparata, F.P. (1978). The densest hemisphere problem. *Theoretical Computer Science* **6** 93–107.

Koshevoy, G.A. (2002). The Tukey depth characterizes the atomic measure. *Journal of Multivariate Analysis* **83** 360–364.

Mosler, K. (2013). Depth statistics. In: Becker, C., Fried, R., Kuhnt, S. (eds.), *Robustness and Complex Data Structures, Festschrift in Honour of Ursula Gather*, Springer-Verlag, Berlin, 17–34.

Liu, X. and Zuo, Y. (2014a). Computing halfspace depth and regression depth. *Communications in Statistics - Simulation and Computation* **43** 969–985.

Liu, X. and Zuo, Y. (2014b). Computing projection depth and its associated estimators. *Statistics and Computing* **24** 51–63.

López-Pintado, S. and Romo, J. (2011). A half-region depth for functional data. *Computational Statistics and Data Analysis* **55** 1679–1695.

Mosler, K., Lange, T. and Bazovkin, P. (2009). Computing zonoid trimmed regions of dimension $d > 2$. *Computational Statistics and Data Analysis* **53** 2500–2510.

Paindaveine, D. and Šiman, M. (2012a). Computing multiple-output regression quantile regions. *Computational Statistics and Data Analysis* **56** 840–853.

Paindaveine, D. and Šiman, M. (2012b). Computing multiple-output regression quantile regions from projection quantiles. *Computational Statistics* **27** 29–49.

Rousseeuw, P.J. and Hubert, M. (1999). Regression depth. *Journal of the American Statistical Association* **94** 388–433.

Rousseeuw, P.J. and Struyf, A. (2004). Characterizing angular symmetry and regression symmetry. *Journal of Statistical Planning and Inference* **122** 161–173.

Tukey, J.W. (1975). Mathematics and the picturing of data. *Proceeding of the International Congress of Mathematicians*, Vancouver, 523–531.

Zuo, Y.J. and Serfling, R. (2000). General notions of statistical depth function. *The Annals of Statistics* **28** 461–482.