Algorithms for Colourful Simplicial Depth and Medians in the Plane

Olga Zasenko and Tamon Stephen

Simon Fraser University, British Columbia, Canada
{ozasenko,tamon}@sfu.ca

Abstract. The *colourful simplicial depth* (CSD) of a point $x \in \mathbb{R}^2$ relative to a configuration $P = (P^1, P^2, \ldots, P^k)$ of n points in k colour classes is exactly the number of closed simplices (triangles) with vertices from 3 different colour classes that contain x in their convex hull. We consider the problems of efficiently computing the colourful simplicial depth of a point x, and of finding a point in $\mathbb{R}^2$, called a *median*, that maximizes colourful simplicial depth.

For computing the colourful simplicial depth of x, our algorithm runs in time $O(n \log n + kn)$ in general, and $O(kn)$ if the points are sorted around x. For finding the colourful median, we get a time of $O(n^4)$. For comparison, the running times of the best known algorithm for the monochrome version of these problems are $O(n \log n)$ in general, improving to $O(n)$ if the points are sorted around x for monochrome depth, and $O(n^4)$ for finding a monochrome median.

1 Introduction

The *simplicial depth* of a point $x \in \mathbb{R}^2$ relative to a set P of n data points is exactly the number of simplices (triangles) formed with the points from P that contain x in their convex hull. A *simplicial median* of the set P is any point in $\mathbb{R}^2$ which is contained in the most triangles formed by elements of P, i.e. has maximum simplicial depth with respect to P. Here we consider a set P that consists of k colour classes $P^1, \ldots, P^k$. The *colourful simplicial depth* of x with respect to configuration P is the number of triangles with vertices from 3 different colour classes that contain x. A *colourful simplicial median* of a configuration $P = (P^1, P^2, \ldots, P^k)$ is any point in the convex hull of P with maximum colourful simplicial depth.

The monochrome simplicial depth was introduced by Liu [16]. Up to a constant, it can be interpreted as the probability that x is in the convex hull of a random simplex generated by P. The colourful version, see [7], generalizes this to selecting points from k distributions. Then medians are central points which are in some sense most representative of the distribution(s). Our objective is find efficient algorithms for finding both the colourful simplicial depth of a given point x with respect to a configuration, and a colourful simplicial median of a configuration.

1.1 Background

Both monochrome and colourful simplicial depth extend to $\mathbb{R}^d$ and are natural objects of study in discrete geometry. For more background on simplicial depth and competing measures of data depth, see [2] and [11]. Monochrome depth has seen a flurry of activity in the past few years, most notably relating to the *First Selection Lemma*, which is a lower bound for the depth of the median, see e.g. [17]. Among the recent work on colourful depth are proofs of the lower [20] and upper [1] bounds conjectured by Deza et al., with the latter result showing beautiful connections to Minkowski sums of polytopes.

The monochrome simplicial depth can be computed by enumerating simplices, but in general dimension, it is quite challenging to compute it more efficiently [2], [5], [11]. Several authors have considered the two-dimensional version of the problem, including Khuller and Mitchell [14], Gil, Steiger and Wigderson [13] and Rousseeuw and Ruts [19]. Each of these groups produced an algorithm that computes the monochrome depth in $O(n \log n)$ time, with sorting the input as the bottleneck. If the input points are sorted, these algorithms take linear time.

We consider a *simplicial median* to be *any* point $x \in \mathbb{R}^2$ maximizing the simplicial depth. Aloupis et al. [3] considered this question, and found an $O(n^4)$ algorithm to do this. This is arguably as good as should be expected, following the observation of Lemma 2 in Section 3.1 that shows that there are in some sense $\Theta(n^4)$ candidate points for the location of the colourful median.

1.2 Organization and Main Results

In Section 2, we develop an algorithm for computing colourful simplicial depth that runs in $O(n \log n + kn)$ time. This retains the $O(n \log n)$ asymptotics of the monochrome algorithms when k is fixed. As in the monochrome case, sorting the initial input is a bottleneck, and the time drops to $O(kn)$ if the input is sorted around x. In this case, for fixed k, it is a linear time algorithm.

In Section 3, we turn our attention to computing a colourful simplicial median. We develop an algorithm that does this in $O(n^4)$ time using a topological sweep. This is independent of k and matches the running time from the monotone case. Section 4 contains conclusions and discussion about future directions.

2 Computing Colourful Simplicial Depth

2.1 Preliminaries

We consider a family of sets P^1, P^2, ... , $P^k \subseteq \mathbb{R}^2$, $k \geq 3$, where each P^i consists of the points of some particular colour i. Refer to the j^{th} element of P^i as P^i_j. We generally use superscripts for colour classes, while subscripts indicate the position in the array. We will sometime perform arithmetic operations on the subscripts, in which case the indices are taken modulo the size of the array i.e. (mod n_i).

We denote the union of all colour sets by P: $P = \bigcup_{i=1}^{k} P^i$. The total number of points is n, where $|P^i| = n_i$, $\sum_{i=1}^{k} n_i = n$. We assume that points of $P \bigcup \{x\}$ are in general position to avoid technicalities. Without loss of generality, we can take $x = \mathbf{0}$, the zero vector.

Definition 1. *A* colourful triangle *is a triangle with one vertex of each colour, i.e. it is a triangle whose vertices v_1, v_2, v_3 are chosen from distinct sets P^{i_1}, P^{i_2}, P^{i_3}, where $i_i \neq i_2, i_3; i_2 \neq i_3$.*

Definition 2. *The* colourful simplicial depth $\hat{D}(x, P)$ *of a point x relative to the set P in $\mathbb{R}^2$ is the number of colourful triangles containing x in their convex hull. We reserve $D(x, P)$ for the (monochrome) simplicial depth, which counts all triangles from P regardless of the colours of their vertices.*

Remark 1. We are checking containment in *closed* triangles. With our general position assumption, this will not affect the value of $\hat{D}(x, P)$. It is more natural to consider closed triangles than open triangles in defining colourful medians; the open triangles version of this question may also be interesting.

Throughout the paper we work with polar angles θ^i_j formed by the data points P^i_j and a fixed ray from x. We remark that simplicial containment does not change as points are moved on rays from x, see for example [23]. Thus we can ignore the moduli of the P^i_j, and work entirely with the θ^i_j, which lie on the unit circle $\mathcal{C}$ with x as its origin. We will at times abuse notation, and not distinguish between P^i_j and θ^i_j.

Note that the ray taken to have angle 0 is arbitrary, and may be chosen based on an underlying coordinate system if available, or set to the direction of the first data point P_1. We can sort the input by polar angle, in other words, we can order the points around x. (Perhaps it is naturally presented this way.) We reduce the θ^i_j to lie in the range $[0, 2\pi)$.

The *antipode* of some point α on the unit circle is $\bar{\alpha} = (\alpha + \pi) \bmod 2\pi$. A key fact in computing CSD is that a triangle $\triangle abc$ does *not* contain x if and only if the corresponding polar angles of points a, b and c lie on a circular arc of less than π radians. This is illustrated in Fig. 1, and is equivalent to the following lemma, stated by Gil, Steiger and Wigderson [13]:

Lemma 1. *Given points a, b, c on the unit circle $\mathcal{C}$ centred at x, let $\bar{a}$ be antipodal to a. Then $\triangle abc$ contains x if and only if $\bar{a}$ is contained in the minor arc (i.e. of at most π radians) with endpoints b and c.*

2.2 Outline of Strategy

Recall that we denote the ordinary and colourful simplicial depth by $D(x, P)$ and $\hat{D}(x, P)$ respectively. We can compute $\hat{D}(x, P)$ by first computing $D(x, P)$

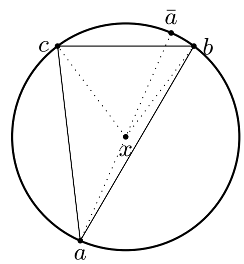


Fig. 1. Antipode $\bar{a}$ falls in the minor arc between b and c and, therefore, the triangle $\triangle abc$ contains x.

and then removing all triangles that contain less than three distinct colours. To this end, we denote the number of triangles with at least two vertices of colour i as $D^i(x, P)$. When x and P are clear from the context, we will abbreviate these to D, $\hat{D}$ and D^i.

Since we can compute $D(x, P)$ efficiently using the algorithms mentioned in the introduction [13], [14], [19], the challenge is to compute $D^i(x, P)$ for each $i = 1, 2, \ldots, k$. Then we conclude $\hat{D}(x, P) = D(x, P) - \sum_{i=1}^{k} D^i(x, P)$. To compute D^i efficiently for each colour i, we walk around the unit circle tracking the minor arcs between pairs of points of colour i, and the number of antipodes between them. We do this in linear time in n by moving the front and back of the interval once around the circle, and adjusting the number of relevant antipodes with each move. This builds on the approach of Gil, Steiger and Wigderson [13] for monochrome depth.

Remark 2. When computing D^i, we count antipodes of all k colours; the triangles with three vertices of colour i will be counted three times: $\triangle abc$, $\triangle bca$ and $\triangle cab$. Thus the quantity obtained by this count is in fact $D^i_* := D^i + 2\sum_{i=1}^{k} D(x, P^i)$. We separately compute $\sum_{i=1}^{k} D(x, P^i)$, allowing us to correct for the overcounting at the end.

2.3 Data Structures and Preprocessing

We begin with the arrays θ^i of polar angles, which we sort if necessary. All elements in $\bigcup_{i=1}^{k} \theta^i$ are distinct due to the general position requirement. By construction we have:

$$0 \leq \theta^i_0 < \theta^i_1 < \ldots < \theta^i_{n_i-1} < 2\pi, \quad \text{for all } 1 \leq i \leq k \ . \tag{1}$$

Let $\bar{\theta}^i$ be the array of antipodes of θ^i, also sorted in ascending order. We generate $\bar{\theta}^i$ by finding the first $\theta^i_j \geq \pi$, moving the part of the array that begins with that

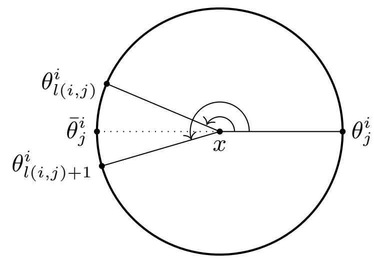


Fig. 2. Index $l(i,j)$ and index $(l(i,j)+1)$

element to the front, and hence the front of the original array to the back; π is subtracted from the elements moved to the front and added to those moved to the back. This takes linear time.

We merge all $\bar{\theta}^i$ into a common sorted array denoted by A. Now we have all antipodes ordered as if we were scanning them in counter-clockwise order around the circle $\mathcal{C}$ with origin x. Let us index the n elements of A starting from 0. Then, for each colour $i = 1, \ldots, k$, we merge A and θ^i into a sorted array A^i. Once again, this corresponds to a counter-clockwise ordering of data points around $\mathcal{C}$.

While building A^i, we associate pointers from the elements of array θ^i to the corresponding position (index) in A^i. This is done by updating the pointers whenever a swap occurs during the process of merging the arrays. Denote the index of some θ^i_j in A^i by $p(\theta^i_j)$. Then the number of the antipodes that fall in the minor arc between two consecutive points θ^i_j and θ^i_{j+1} on $\mathcal{C}$ is $\left(p\left(\theta^i_{j+1}\right) - p\left(\theta^i_j\right) - 1\right)$, if $p\left(\theta^i_j\right) < p\left(\theta^i_{j+1}\right)$, or $\left(n + n_i - p\left(\theta^i_j\right) + p\left(\theta^i_{j+1}\right) - 1\right)$, if $p\left(\theta^i_j\right) > p\left(\theta^i_{j+1}\right)$. Note that $p\left(\theta^i_j\right)$ is never equal to $p\left(\theta^i_{j+1}\right)$.

Now, for each point θ^i_j, we find the index $l(i,j)$ in the corresponding array θ^i such that $\angle\theta^i_j, x, \theta^i_{l(i,j)} < \pi$ and $\angle\theta^i_j, x, \theta^i_{l(i,j)+1} > \pi$ (Fig. 2). Thus the sequence of points $\theta^i_j, \theta^i_{j+1}, \ldots, \theta^i_{l(i,j)}$ is maximal on an arc shorter than π. Viewing the minor arc between two points as an interval, the intervals with left endpoint θ^i_j and right end point from this sequence overlap and can be split into small disjoint intervals as follows:

$$\left[\theta^i_j, \theta^i_t\right) = \bigcup_{h=j+1}^{t} \left[\theta^i_{h-1}, \theta^i_h\right), \text{ where } t = j+1, \ldots, l(i,j) \ . \tag{2}$$

2.4 Computing D^i_*

Let us denote the count of the antipodes within the minor arc between a and b by $c(a,b)$. Then D^i_* can be written as follows:

$$D^i_* = \sum_{j=0}^{n_i-1} \sum_{t=j+1}^{l(i,j)} c\left(\theta^i_j, \theta^i_t\right) \ . \tag{3}$$

Note that index t is taken modulo n_i. From (2) we have:

$$c\left(\theta_j^i, \theta_t^i\right) = \sum_{h=j+1}^{t} c\left(\theta_{h-1}^i, \theta_h^i\right), \quad \text{for } t = j+1, \ldots, l(i,j) \ . \tag{4}$$

Due to (3) and (4), we have:

$$D_*^i = \sum_{j=0}^{n_i-1} \sum_{t=j+1}^{l(i,j)} \sum_{h=j+1}^{t} c\left(\theta_{h-1}^i, \theta_h^i\right) \ . \tag{5}$$

Let $C_h^i = c\left(\theta_{h-1}^i, \theta_h^i\right)$, $|C^i| = n_i$. Then (5) can be rewritten as:

$$D_*^i = \sum_{j=0}^{n_i-1} \sum_{t=j+1}^{l(i,j)} \sum_{h=j+1}^{t} C_h^i \ . \tag{6}$$

Let us create an array of prefix sums: S^i, where $S_t^i = \sum_{h=0}^{t} C_h^i$, $|S^i| = n_i$. This array can be filled in $O(n_i)$ time and proves to be very useful when we need to calculate a sum of the elements of C^i between two certain indices. In fact, such sum can be obtained in constant time using the elements of array S^i:

$$\sum_{h=j+1}^{t} C_h^i = \begin{cases} S_t^i - S_j^i, & \text{if } t \geq j+1, j \neq n_i - 1 \ , \\ S_{n_i-1}^i + S_t^i - S_j^i, & \text{if } t < j+1, j \neq n_i - 1 \ , \\ S_t^i, & \text{if } j = n_i - 1 \ . \end{cases} \tag{7}$$

Combining (6) and (7), we get:

$$D_*^i = \sum_{j=0}^{n_i-1} \sum_{t=j+1}^{l(i,j)} S_t^i - \sum_{j=0}^{n_i-1} \left((l(i,j) - j) \bmod n_i\right) \cdot S_j^i + \begin{cases} 0, \text{ if } t \geq j+1 \text{ or } \ j = n_i - 1 \ , \\ \sum_{j=0}^{n_i-1} \sum_{t=j+1}^{l(i,j)} S_{n_i-1}^i, \text{if } t < j+1 \ . \end{cases} \tag{8}$$

Let us create another array of prefix sums T^i, where $T_j^i = \sum_{t=0}^{j} S_t^i$, $|T^i| = n_i$. This array is used to retrieve the sum of elements of S^i between the indices $j+1$ and $l(i,j)$ in $O(1)$ time:

$$\sum_{t=j+1}^{l(i,j)} S_t^i = \begin{cases} T_{l(i,j)}^i - T_j^i, & \text{if } l(i,j) \geq j+1, j \neq n_i - 1 \ , \\ T_{n_i-1}^i + T_{l(i,j)}^i - T_j^i, & \text{if } l(i,j) < j+1, j \neq n_i - 1 \ , \\ T_{l(i,j)}^i, & \text{if } j = n_i - 1. \end{cases} \tag{9}$$

Also note that the index t runs from $j+1$ to $l(i,j)$. So $t < j+1$ in (8) is only possible if initially $j+1 > l(i,j)$ and we wrapped around the array. In other words, $t < j+1$ is equivalent to $j+1 > l(i,j)$ and $t = 0, \ldots, l(i,j)$.

After simplifying, we obtain:

$$D^i_* = \sum_{j=0}^{n_i-1} \left(T^i_{l(i,j)} - T^i_j - ((l(i,j)-j) \bmod n_i) \cdot S^i_j\right) + \begin{cases} n_i \cdot \left(T^i_{n_i-1} + ((l(i,j)+1) \bmod n_i) \cdot S^i_{n_i-1}\right), & \text{if } l(i,j) < j+1 \ , \\ 0, & \text{otherwise} \ . \end{cases} \tag{10}$$

2.5 Algorithm and Analysis

Algorithm 1 $\mathtt{CSD}(\mathtt{x},\mathtt{P})$

```
Input: x, P = (P^1, ..., P^k). Output: D̂(x, P).
Sum1 ← 0, Sum2 ← 0;
for i ← 1, k do
    for j ← 0, n_i − 1 do
        θ^i_j ← polar angle of (P^i_j − x) mod 2π;
        θ̄^i_j ← (θ^i_j + π) mod 2π;
    end for
    Sort(θ^i);                                          ▷ while permuting θ̄^i
    Restore the order in θ̄^i;
    Sum1 ← Sum1 + D(x, θ^i);                            ▷ use the algorithm from [19]
end for
A ← Merge(θ̄^1, ..., θ̄^k);                              ▷ A is sorted
D ← D(x, A);                                            ▷ use the algorithm from [19]
for i ← 1, k do
    B ← Merge(A, θ^i);                  ▷ update p(θ^i_j) the pointers of θ^i_j,
                                                        ▷ B stands for A^i
    for j ← 1, n_i do                                   ▷ j = j mod n_i
        if p(θ^i_{j−1}) < p(θ^i_j) then
            C_j ← p(θ^i_j) − p(θ^i_{j−1}) − 1;    ▷ C = C^i - array of antipodal counts
        else
            C_j ← n + n_i − p(θ^i_{j−1}) + p(θ^i_j) − 1;
        end if
    end for
    Find l(i,0) using binary search in θ^i;
    S_0 ← C_0; T_0 ← S_0;                         ▷ S = S^i, T = T^i - prefix sum arrays
    for j ← 1, n_i − 1 do
        Find l(i,j);
        S_j ← S_{j−1} + C_j;
        T_j ← T_{j−1} + S_j;
    end for
    Sum2 ← Sum2 + D^i_*(x, P) obtained from the formula (10);
    delete B, C, S, T;
end for
return D̂(x, P) = D − (Sum2 − 2 ∗ Sum1);             ▷ Sum1 = Σ_{i=1}^{k} D(x, P^i)
```

First, we find all polar angles and their antipodes, which takes $O(n)$ in total. Second, we sort the arrays of polar angles θ^i and their corresponding antipodal elements $\bar{\theta}^i$, which gives us $O\left(\sum_{i=1}^{k} n_i \log n_i\right)$. Third, we need to rotate $\bar{\theta}^i$, so that they are in ascending order. This will take $O(n)$ time. Then we compute for each i the number of triangles with all three vertices of colour i that contain x using the algorithm of Rousseeuw and Ruts [19] for sorted data. This will run in $O(n_i)$, for each i, or $O(n)$ in total. Hence lines 2-10 of the Algorithm 1 take $O\left(\sum_{i=1}^{k} n_i + \sum_{i=1}^{k} n_i \log n_i\right) = O(n \log n)$ time to complete. This follows from the facts that $\sum_{i=1}^{k} n_i = n$ and $n \log n$ is convex.

To generate the sorted array A of antipodes, we merge the k single-coloured arrays using a heap (following e.g. [6]) in $O(n \log k)$ time. We need to compute the monochrome depth $D(x, P)$ of x with respect to all points in P, regardless of colour. For this we can use the sorted array of antipodes rather than sorting the original array. Thus we again use the linear time monochrome algorithm [19] with x and A. Note that working with the antipodes is equivalent due to the fact that the simplicial depth of x does not change if we rotate the system of data points around the centre x.

After that, we execute a cycle of k iterations – one for each colour. It starts with merging two sorted arrays A and θ^i, which is linear in the size of arrays we are merging and takes $O\left(\sum_{i=1}^{k} (n + n_i)\right) = O(kn)$ in total. Filling the arrays C is linear. Since the $l(i, j)$ appear in sequence in the array θ^i, we find the first one $l(i, 0)$ using a binary search that takes $O(\log n_i)$, and $O(k \log n)$ in total. Then we find the rest of $l(i, j)$ in $O(n)$ time for each i by scanning through the array starting from the element $\theta^i_{l(i,0)}$. The remaining operations take constant time to execute. Therefore, total running time of Algorithm 1 is $O(n + n \log n + n + n \log k + kn + k \log n + kn) = O(n \log n + kn)$. The $n \log n$ term corresponds to the initial sorting of the data points, if they are presented in sorted order, the running time drops to $O(kn)$.

As for space, arrays θ^i, $\bar{\theta}^i$ and A take $O(3n) = O(n)$ space in total. Note that merging k sorted arrays into A can be done in place [12]. At each iteration i, we create B of size $O(n + n_i)$, and C, S, T of size $O(n_i)$ each. Fortunately, we only need these arrays within the i^{th} iteration, so we can delete them in the end (line 31 of the Algorithm 1) and reuse the space freed. To store the indices $l(i, j)$, we need $O(n)$ space, which again can be reallocated when i changes. Thus the amount of space used by our algorithm is $O(n)$.

An implementation of this algorithm is available on-line [22].

Remark 3. In Section 3, we will want to compute the colourful simplicial depth of the data points themselves. This can be done by computing $\hat{D}(x, P \setminus \{x\})$ and counting colourful simplices which have x as a vertex. This is the number of pairs of vertices of some other colour, and can be computed in linear time.

3 Computing Colourful Simplicial Medians

3.1 Preliminaries

Consider a family of sets $P^1, P^2, \ldots, P^k \in \mathbb{R}^2$, $k \geq 3$, where each P^i consists of the points of some particular colour i. Define $n_i = |P^i|$, for $i = 1, \ldots, k$. Let P be the union of all colour sets: $P = \bigcup_{i=1}^{k} P^i$. Recall that we denote the CSD of a point $x \in \mathbb{R}^2$ relative to P by $\hat{D}(x, P)$.

Our objective is to find a point x inside the convex hull of P, denoted $conv(P)$, maximizing $\hat{D}(x, P)$. Call the depth of such a point $\hat{\mu}(P)$. Let S be the set of line segments formed by all possible pairs of points (A, B), where $A \in P^i$, $B \in P^j$, $i < j$. The following lemma (from [3]) is here adapted to a colourful setting:

Lemma 2. *To find a point with maximum colourful simplicial depth it suffices to consider the intersection points of the colourful segments in S.*

Proof. The segments of S partition $conv(P)$ into cells[1] of dimension 2, 1, 0 of constant colourful simplicial depth [7]. Consider a 2-dimensional cell. Let p be a point in the interior of this cell, q a point on the interior of an edge and v a vertex, so that q and v belong to the same line segment (Fig. 3). Then the following inequality holds: $\hat{D}(p, P) \leq \hat{D}(q, P) \leq \hat{D}(v, P)$, since any colourful simplices containing p also contain q, and any containing q also contain v.

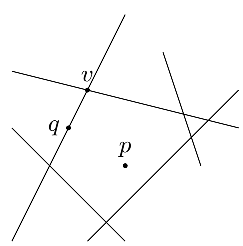


Fig. 3. An example of a cell

Let $col(A)$ denote the colour of a point A. We store the segments in S as pairs of points: $s = (A, B)$, $col(A) < col(B)$. It is helpful to view each segment as directed, i.e. a vector, with A as the tail and B as the head. Each segment s extends to a directed line h dividing $\mathbb{R}^2$ into two open half-spaces: s^+ and s^-, where s^+ lies to the right of the vector s, and s^- to the left (Fig. 4). We denote the set of lines generated by segments by H, so that every segment $s \in S$ corresponds to a line $h \in H$.

We call the intersection points of the segments in S *vertices*. Note that drawing the colourful segments is equivalent to generating a rectilinear drawing of the

[1] Some points of $conv(P)$ may fall outside any cell.

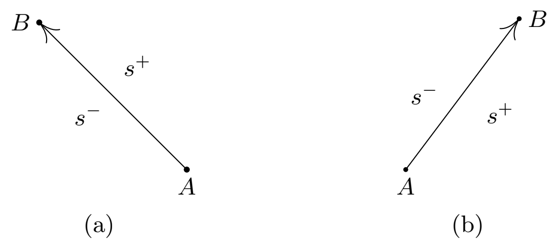


Fig. 4. s^+ and s^- of the segment $s = (A, B)$

complete graph K_n with a few edges removed (the monochrome ones). Thus, unless the points are concentrated in a single colour class, the Crossing Lemma (see e.g. [18]) shows that we will have $\Theta(n^4)$ vertices. Computing the CSD of each of these points gives an $O(n^4 \log n)$ algorithm for finding a simplicial median.

To improve this, we follow Aloupis et al. [3], and compute the monochrome simplicial depth of most vertices based on values of their neighbours and information about the half-spaces of local segments.

Denote the number of points in s^+ that have colours different from the endpoints of s by $r(s)$, and those in s^- by $l(s)$. Let $r^i(s)$ and $l^i(s)$ be the number of points of a colour i in s^+ and s^- respectively. Let $\bar{r}^i(s)$ and $\bar{l}^i(s)$ be the number of points of all k colours except for the colour i in s^+ and s^- respectively. So for segment $s = (A, B)$, we have quantities as follows $\bar{r}^{col(A)}(s) = \sum_{\substack{i=1,\\ i\neq col(A)}}^{k} r^i(s)$, $\bar{l}^{col(A)}(s) = \sum_{\substack{i=1,\\ i\neq col(A)}}^{k} l^i(s)$. Then it follows: $r(s) = \bar{r}^{col(A)}(s) - r^{col(B)}(s)$ and $l(s) = \bar{l}^{col(A)}(s) - l^{col(B)}(s)$. The quantities $\bar{r}^{col(A)}(s)$ and $\bar{l}^{col(A)}(s)$, $r^{col(B)}(s)$ and $l^{col(B)}(s)$, can be obtained as byproducts of running an algorithm that computes half-space depth.

The *half-space depth* $HSD(x, P)$ of a point x relative to data set P is the smallest number of data points in a half-plane through the point x [21]. An algorithm to compute half-space depth is described by Rousseeuw and Ruts [19], it runs in $O(|P|)$ time when P is sorted around x. It calculates the number of points k_i in P that lie strictly to the left of each line formed by x and some point P_i, where x is the tail of the vector $\overrightarrow{xP_i}$. Then the number of points to the right $\overrightarrow{xP_i}$ is $|P| - k_i - 1$. These intermediate calculations are used in our algorithm.

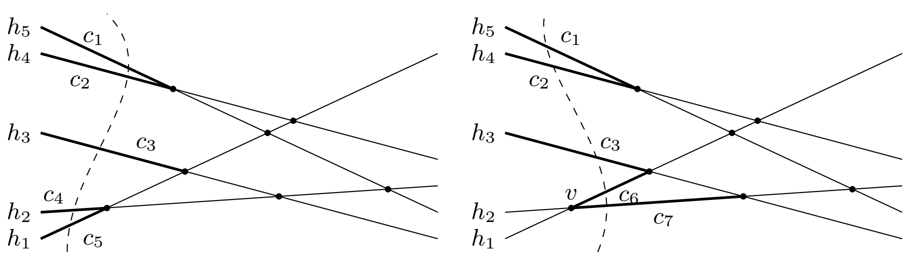
The algorithm of [15] will, for each $P_i \in P$, sort $P \setminus \{P_i\}$ around P_i in $\Theta(|P|^2)$ time. In particular, it assigns every point $P_i \in P$ a list of indices that determine the order of points $P \setminus \{P_i\}$ in the clockwise ordering around P_i. Denote this by $List(P_i)$. These ideas allow us to compute $r(s)$ and $l(s)$ for every segment s. At every iteration i, we form arrays of sorted polar angles $\bar{\theta}^{col(P_i)}$ and $\theta^{i'}$. Together they take $O(2n) = O(n)$ space.

3.2 Computing a Median

To compute the CSD of all vertices, we carry out a topological sweep (see e.g. [9]). We begin by extending the segments in S to a set of lines H. The set V^* of intersection points of lines of H includes the $\Theta(n^4)$ vertices V which are on the interior of a pair of segments of S, points from P, and additional exterior intersections. We call points in $V^* \setminus V$ *phantom vertices*.

Call a line segment of any line in H between two neighbouring vertices, or a ray from a vertex on a line that contains no further vertices an *edge*. A *topological line* is a curve in $\mathbb{R}^2$ that is topologically a line and intersects each line in H exactly once. We choose an initial topological line to be an unbounded curve that divides $\mathbb{R}^2$ into two pieces such that all the finitely many vertices in V lie on one side of the curve, by convention the right side. We call this line the *leftmost cut*. We call a *vertical cut* the list $(c_1, c_2, \ldots, c_m)$ of the $m = |H|$ edges intersecting a particular topological line. For each i, $1 \le i \le m-1$, c_i and c_{i+1} share a 2-cell in the complex induced by H.

The topological sweep begins with the leftmost cut and moves across the arrangement to the right, crossing one vertex at a time. If two edges c_i and c_{i+1} of the current cut have a common right endpoint, we store the index i in the stack I. For example, in Figure 5(a), $I = \{1, 4\}$. An *elementary step* is performed when we move to a new vertex by popping the stack I. In Figure 5(b), we have moved past the vertex v, a common right endpoint of c_4 and c_5 which is the intersection point of h_1 and h_2. The updated stack is $I = \{1, 3\}$.



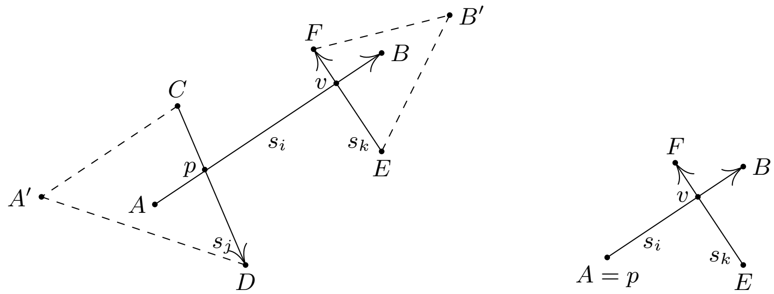
(a) The leftmost cut

(b) An elementary step in a topological sweep

Fig. 5.

We focus on the elementary steps, because at each step we can compute the CSD of the crossed vertex. As it moves, the topological line retains the property that everything to the left of it has already been swept over. That is, if we are crossing vertex v that belongs to segment s, every vertex of the line containing s on the opposite side of the topological line prior to crossing has already been swept. For each segment $s \in S$ we store the last processed vertex and denote it by $ver(s)$, along with its CSD. Since every vertex lies at the intersection of two segments, we also store the crossing segment for s and $ver(s)$, denote it by $cross(ver(s))$. Before starting the topological sweep, for each $s \in S$ we assign $ver(s) = \emptyset$, and $cross(ver(s)) = \emptyset$. After completing an elementary step where we crossed a vertex v that lies at the intersection of s_i and s_j, we assign $ver(s_i) \leftarrow v$, $ver(s_j) \leftarrow v$, $cross(ver(s_i)) = s_j$, $cross(ver(s_j)) = s_i$.

The topological sweep skips through phantom vertices, and computes the CSD of vertices in P directly. We now explain how we process a non-phantom vertex v at an elementary step when we have an adjacent vertex already computed. Assume v is at the intersection of $s_i = \overrightarrow{AB}$ and $s_k = \overrightarrow{EF}$, see Figure 6(a). Without loss of generality we take $ver(s_i) = p$, where $cross(ver(s_i)) = s_j$. We



(a) Two adjacent vertices p and v and their corresponding line segments. A colourful triangle $\triangle CDA'$ contains p but not v, where $col(A') \notin \{col(C), col(D)\}$. Similarly, a colourful triangle $\triangle EFB'$ contains v but not p, where $col(B') \notin \{col(E), col(F)\}$.

(b) Here $ver(s_i) = \emptyset$, hence $cross(ver(s_i)) = \emptyset$, and we can not run Subroutine 2.

Fig. 6. Capturing a new vertex

view this elementary step as moving along the segment s_i from its intersection point with s_j to the one with s_k. Each intersecting segment forms a triangle with every point strictly to one side. Thus when we leave segment $s_j = (C, D)$ behind, we exit as many colourful triangles that contain p as there are points on the other side of s_j of colours different from $col(C)$ and $col(D)$. When we encounter segment $s_k = (E, F)$, we enter the colourful triangles that contain v formed by s_k and each point of a colour different from $col(E)$ and $col(F)$ on the other side of s_k. Let us denote the x and y coordinates of a point A by $A.x$ and $A.y$ respectively. Now, to compute the CSD of v knowing the CSD of p, we execute Subroutine 2.

When both $ver(s_i) = \emptyset$, $ver(s_k) = \emptyset$, i.e. vertex v is the first vertex to be discovered for both segments (Fig. 6(b)), we execute $CSD(v, P)$ to find the depth, and otherwise update in the usual way. Since once a segment s has $ver(s)$ nonempty it cannot return to being empty, we call CSD at most $O(n^2)$ times.

3.3 Running Time and Space Analysis

Algorithm 3 is our main algorithm. First, it computes the half-space counts $r(s)$ and $l(s)$, which has a running time of $O(n^2)$. At the same time, we initialize the structure S that contains the colourful segments, setting $ver(s) = \emptyset$ and

Subroutine 2 Computing $\hat{D}(v)$ from $\hat{D}(p)$

```
Input: D̂(p), p, v, s_j = (C, D), s_k = (E, F). Output: D̂(v).
if (v.x − C.x)(D.y − C.y) − (v.y − C.y)(D.x − C.x) < 0 then
    D̂(v) ← D̂(p) − r(s_j);
else
    D̂(v) ← D̂(p) − l(s_j);
end if
if (p.x − E.x)(F.y − E.y) − (p.y − E.y)(F.x − E.x) < 0 then
    D̂(v) ← D̂(v) + r(s_k);
else
    D̂(v) ← D̂(v) + l(s_k);
end if
```

$cross(ver(s)) = \emptyset$ for all $s \in S$. Note that these as well as H, $List(P_i)$, $r(s)$, $l(s)$ require $O(n^2)$ storage.

Sorting the lines in H according to their slopes while also permuting the segments in S takes $O(n^2 \log n)$ time. We assume non-degeneracy and no vertical lines (these can use some special handling, see e.g. [10]). Computing the CSD of points where no previous vertex is available takes $O(n^2 \log n + kn^2)$ total time. The topological sweep takes linear time in the number of intersection points of H, so $O(n^4)$. We do not store all the vertices, but only one per segment. Steps 15-28 (except for 18) in Algorithm 3 take O(1) time, including the calls to the Subroutine 2. As for step 18, it could happen $O(n)$ times. Therefore, the total time it will take is $O(n^2 \log n + kn^2)$. Hence overall our algorithm takes $O(n^4)$ time and needs $O(n^2)$ storage.

Algorithm 3 returns a point that has maximum colourful simplicial depth along with its CSD. It is simple to modify the algorithm to return a list of all such points if there is more than one.

4 Conclusions and Questions

Our main result is an algorithm computing the colourful simplicial depth of a point x relative to a configuration $P = \left(P^1, P^2, \ldots, P^k\right)$ of n points in $\mathbb{R}^2$ in k colour classes can be solved in $O(n \log n + kn)$ time, or in $O(kn)$ time if the input is sorted. If we assume, as seems likely, that we cannot do better without sorting the input, then for fixed k this result is optimal up to a constant factor. It is an interesting question whether we can improve the dependence on k, in particular when k is large.

Computing colourful simplicial depth in higher dimension is very challenging, in particular because there is no longer a natural (circular) order of the points. Non-trivial algorithms for monochrome depth do exist in dimension 3 [5], [13], but we do not know of any non-trivial algorithms for $d \geq 4$. Algorithms for monochrome and colourful depth in higher dimension are an appealing challenge. Indeed, for $(d{+}1)$ colours in $\mathbb{R}^d$, it is not even clear how efficiently one can exhibit a single colourful simplex containing a given point [4], [8].

Algorithm 3 Computing $\hat{\mu}(P)$

```
Input: P^1, ..., P^k, S, H, r(s), l(s). Output: v, μ̂(P).
Preprocessing: Initialize S, compute r(s), l(s);
Sort H while permuting S;
max ← 0;
for i ← 0, n − 1 do
    θ = polar angles of List(P_i);
    D̂(P_i) ← CSD(P_i, θ);
    if d > max then
        max ← D̂(P_i);
        median ← P_i;
    end if
end for
I ← ∅;
Push common right endpoints of the edges of the leftmost cut onto I;
while I ≠ ∅ do                                  ▷ Start of the topological sweep.
    v ← pop(I);  ▷ v lies at the intersection of s_i = (A, B) and s_k = (E, F)
    if v lies in the interiors of s_i and s_k then
        if ver(s_i) = ∅ & ver(s_k) = ∅ then
            D̂(v) = CSD(v, P);
        else if ver(s_i) ≠ ∅ then
            D̂(v) ← Subr 2 (D̂(p), p, v, s_j, s_k);     ▷ p = ver(s_i), s_j = cross(ver(s_i))
        else
            D̂(v) ← Subr 2 (D̂(p), p, v, s_j, s_i);     ▷ p = ver(s_k), s_j = cross(ver(s_k))
        end if
        if D̂(v) > max then
            max ← D̂(v);
            median ← v;
        end if
        ver(s_i) ← v, ver(s_k) ← v, cross(ver(s_i)) ← s_k, cross(ver(s_k)) ← s_i;
    end if
    Push any new common right endpoints of the edges onto I;
end while                                       ▷ End of the topological sweep.
return (median, max).
```

Acknowledgments

This research was partially supported by an NSERC Discovery Grant to T. Stephen and by an SFU Graduate Fellowships to O. Zasenko. We thank A. Deza for comments on the presentation.

References

1. Adiprasito, K., Brinkmann, P., Padrol, A., Paták, P., Patáková, Z., Sanyal, R.: Colorful simplicial depth, Minkowski sums, and generalized Gale transforms (2016), preprint. ArXiv:1607.00347

2. Aloupis, G.: Geometric measures of data depth. In: Data depth: robust multivariate analysis, computational geometry and applications, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., vol. 72, pp. 147–158. Amer. Math. Soc., Providence (2006)
3. Aloupis, G., Langerman, S., Soss, M., Toussaint, G.: Algorithms for bivariate medians and a Fermat-Torricelli problem for lines. Comput. Geom. 26(1), 69–79 (2003)
4. Bárány, I., Onn, S.: Colourful linear programming and its relatives. Math. Oper. Res. 22(3), 550–567 (1997)
5. Cheng, A.Y., Ouyang, M.: On algorithms for simplicial depth. In: Proceedings of the 13th Canadian Conference on Computational Geometry. pp. 53–56 (2001)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. The MIT Press and McGraw-Hill Book Company (1989)
7. Deza, A., Huang, S., Stephen, T., Terlaky, T.: Colourful simplicial depth. Discrete Comput. Geom. 35(4), 597–615 (2006)
8. Deza, A., Huang, S., Stephen, T., Terlaky, T.: The colourful feasibility problem. Discrete Appl. Math. 156(11), 2166–2177 (2008)
9. Edelsbrunner, H., Guibas, L.J.: Topologically sweeping an arrangement. J. Comput. System Sci. 38(1), 165–194 (1989), 18th Annual ACM Symposium on Theory of Computing (Berkeley, CA, 1986)
10. Edelsbrunner, H., Mücke, E.P.: Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. In: Proceedings of the Fourth Annual Symposium on Computational Geometry (Urbana, IL, 1988). pp. 118–133. ACM, New York (1988)
11. Fukuda, K., Rosta, V.: Data depth and maximum feasible subsystems. In: Graph theory and combinatorial optimization, GERAD 25th Anniv. Ser., vol. 8, pp. 37–67. Springer, New York (2005)
12. Geffert, V., Gajdoš, J.: Multiway in-place merging. Theoret. Comput. Sci. 411(16-18), 1793–1808 (2010)
13. Gil, J., Steiger, W., Wigderson, A.: Geometric medians. Discrete Math. 108(1-3), 37–51 (1992)
14. Khuller, S., Mitchell, J.S.B.: On a triangle counting problem. Inform. Process. Lett. 33(6), 319–321 (1990)
15. Lee, D.T., Ching, Y.T.: The power of geometric duality revisited. Inform. Process. Lett. 21(3), 117–122 (1985)
16. Liu, R.Y.: On a notion of data depth based on random simplices. Ann. Statist. 18(1), 405–414 (1990)
17. Matousek, J., Wagner, U.: On Gromov's method of selecting heavily covered points 52(1), 1–33 (2014)
18. Pach, J., Radoičić, R., Tardos, G., Tóth, G.: Improving the crossing lemma by finding more crossings in sparse graphs. Discrete Comput. Geom. 36(4), 527–552 (2006)
19. Rousseeuw, P.J., Ruts, I.: Bivariate location depth. Applied Statistics: Journal of the Royal Statistical Society Series C 45(4), 516–526 (1996)
20. Sarrabezolles, P.: The colourful simplicial depth conjecture. J. Combin. Theory Ser. A 130, 119–128 (2015)
21. Tukey, J.W.: Mathematics and the picturing of data. In: Proceedings of the International Congress of Mathematicians (Vancouver, B. C., 1974), Vol. 2. pp. 523–531. Canad. Math. Congress, Montreal, Que. (1975)
22. Zasenko, O.: Colourful simplicial depth in the plane (2016), `https://github.com/olgazasenko/ColourfulSimplicialDepthInThePlane`, accessed July 13, 2016
23. Zuo, Y., Serfling, R.: General notions of statistical depth function. Ann. Statist. 28(2), 461–482 (2000)