# A Fast Dimension-Sweep Algorithm for the Hypervolume Indicator in Four Dimensions

Andreia P. Guerreiro*† Carlos M. Fonseca† Michael T. M. Emmerich‡

**Abstract**

The Hypervolume Indicator is one of the most widely used quality indicators in Evolutionary Multiobjective Optimization. Its computation is a special case of Klee's Measure Problem (KMP) where the upper end of all rectangular ranges coincides with a given reference point (assuming minimization, without loss of generality). Although the time complexity of the hypervolume indicator in two and three dimensions is known to be $\Theta(n \log n)$, improving upon the $O(n^{d/2} \log n)$ complexity of Overmars and Yap's algorithm for the general KMP in higher dimensions has been a challenge. In this paper, a new dimension-sweep algorithm to compute the hypervolume indicator in four dimensions is proposed, and its complexity is shown to be $O(n^2)$.

## 1 Introduction

In multiobjective optimization, solutions may be seen as points in a decision space, $\mathcal{S}$, which are mapped onto a multi-dimensional objective space, $\mathbb{R}^d$, by means of a vector-valued objective function, $f : \mathcal{S} \rightarrow \mathbb{R}^d$. Minimization of all objective function components is assumed throughout this work without loss of generality.

In this context, a solution $x \in \mathcal{S}$ is said to dominate another solution $z \in \mathcal{S}$ iff $f(x) \leq f(z)$ and $f(x) \neq f(z)$, where the inequality $\leq$ applies componentwise. A solution $x \in \mathcal{S}$ is said to be Pareto-optimal iff $\forall z \in \mathcal{S}, f(z) \leq f(x) \Rightarrow f(z) = f(x)$. The set of all Pareto-optimal solutions in decision space is called the *Pareto-optimal set*, and the corresponding set of points in objective space is called the *Pareto-optimal front*.

Since enumerating the whole Pareto-optimal set, or even the Pareto-optimal front, is usually infeasible, multiobjective optimization typically aims at finding a good, discrete approximation to the Pareto-optimal front. In comparative studies, the quality of such approximations is often assessed in a quantitative manner by means of quality indicators, which map a set of points in objective space to a real value. Such quality indicators have also been integrated into some evolutionary multiobjective optimizers [3, 14], which leads to the quality indicator being evaluated many times in the course of an optimization run, and imposes a need for efficient algorithms to compute it.

One of the most widely used quality indicators is the hypervolume indicator [18]. Given a set of points $\mathrm{X} \subset \mathbb{R}^d$ and a reference point $r \in \mathbb{R}^d$, the hypervolume indicator $H(\mathrm{X})$ is the Lebesgue measure, $\lambda(\cdot)$, of the region dominated by X and bounded above by $r$, i.e. $H(\mathrm{X}) = \lambda(\{q \in \mathbb{R}^d \mid \exists p \in \mathrm{X} : p \leq q \wedge q \leq r\})$. Alternatively, the hypervolume indicator may be written as the measure of the union of $n$ isothetic hyperrectangles in $d$ dimensions:

$$H(\mathrm{X}) = \lambda\left(\bigcup_{\substack{p \in \mathrm{X} \\ p \leq r}} [p, r]\right)$$

where $[p, r] = \{q \in \mathbb{R}^d \mid p \leq q \wedge q \leq r\}$, which highlights its connection to Klee's Measure Problem (KMP) [13]. Furthermore, given a point $p \in \mathrm{X}$ and a reference point $r \in \mathbb{R}^d$, the individual contribution of $p$ is the Lebesgue measure of the region exclusively dominated by $p$. It can be obtained by subtracting $H(\mathrm{X} - \{p\})$ from $H(\mathrm{X})$.

It is known that the hypervolume indicator and, thus, Klee's measure problem cannot be computed exactly in time polynomial in the number of dimensions unless P = NP [6]. While asymptotically optimal, $O(n \log n)$-time algorithms for the hypervolume indicator in two and three dimensions are known [2], no tight complexity bounds are available for $d \geq 4$. For a long time, the best upper bounds for $d \geq 4$ stemmed from algorithms for the more general KMP. Chan's algorithm [8], with time bound $O(n^{d/2}\, 2^{O(\log^* n)})$, where $\log^*$ denotes the iterated logarithm function, provides the best general upper bound to date. It slightly improves upon Overmars and Yap's algorithm, which has time complexity $O(n^{d/2} \log n)$ [13]. Beume [1] developed a simplified version of Overmars and Yap's algorithm for the hypervolume indicator, but with the same complexity.[1]

*INESC-ID, Instituto Superior Técnico, Technical University of Lisbon, Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal apg@dei.uc.pt

†CISUC, Department of Informatics Engineering, University of Coimbra, Pólo II, 3030-290 Coimbra, Portugal cmfonsec@dei.uc.pt

‡LIACS, Faculty of Science, Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands emmerich@liacs.nl

[1]A gap in the analysis presented in N. Beume and G. Rudolph, *Faster S-Metric Calculation by Considering Dominated Hypervolume as Klee's Measure Problem*, in Proc. 2nd IASTED Conf. on Comp. Intelligence, 231–236, 2006, is acknowledged in [1].

Better bounds have been obtained for the KMP on unit cubes [7] and on fat boxes [5], for example, but reducing the hypervolume indicator to such problems is not possible in general. Only very recently has a tighter, general upper bound of $O(n^{(d-1)/2} \log n)$ on the time complexity of the hypervolume indicator been obtained [17].

In this paper, a fast dimension-sweep algorithm for the hypervolume indicator in four dimensions is proposed. Although its quadratic time complexity exceeds Yıldız and Suri's new upper bound by a factor of $n^{1/2}/\log n$, it can be easily implemented based on simpler data structures, and runs much faster than the currently available implementations of alternative algorithms on standard benchmark instances [11].

The paper is structured as follows: the next section reviews some of the existing algorithms for the hypervolume indicator and their time complexities. Section 3 describes the proposed algorithm to compute the hypervolume indicator in four dimensions, and states its complexity. Concluding remarks are drawn in Section 4.

## 2 Related work

Several algorithms for the hypervolume indicator have been proposed in the literature in addition to the aforementioned simplified algorithm by Beume, known as HOY [1]. Fonseca *et al.*'s algorithm [10] is a recursive dimension-sweep algorithm which improves upon a previous algorithm known as HSO (Hypervolume by Slicing Objectives) [16] by caching intermediate results without sacrificing linear space complexity, and using the asymptotically optimal $O(n \log n)$ algorithm later detailed by Beume *et al.* [2] as its three-dimensional base case. Its $O(n^{d-2} \log n)$ time complexity for $d > 2$ matches HOY's for $d = 4$, but is worse for greater values of $d$.

Two other algorithms, IIHSO (Iterated Incremental HSO) [4] and WFG (Walking Fish Group) [15], have been reported to be the fastest known algorithms in practice, the former for $d = 4$ and the latter for $d > 4$, based on experimental results on a set of benchmark instances [15]. However, IIHSO has $O(n^{d-1})$ time complexity and WFG is reported to be exponential in the number of points in the worst case [15]. Finally, Yıldız and Suri's new algorithm [17] is asymptotically the fastest to date, and it will be interesting to see how well it performs in practice.

## 3 New dimension-sweep algorithm for the four-dimensional case

Like WFG [15], the new algorithm proposed here follows a dimension-sweep approach, and implements an iterated incremental computation of the hypervolume indicator. However, its time complexity can be shown to be at most quadratic.

The next subsection presents the main ideas behind the proposed algorithm. An algorithm to compute the individual contribution of a point in three dimensions, on which the main algorithm relies, is described in Subsection 3.2. The data structures used and the operation of the overall algorithm are described in Subsections 3.3 and 3.4, respectively. Subsection 3.5 discusses in detail how the $O(n^2)$ time complexity is achieved.

### 3.1 General description

An asymptotically optimal, $O(n \log n)$-time algorithm to compute the hypervolume indicator in three-dimensions is described by Beume *et al.* [2], and is a natural extension of Kung *et al.*'s algorithm for maxima in three dimensions [12]. In a minimization setting, the algorithm operates by sweeping input points in ascending order of $z$-coordinate values. While sweeping, the set S of the points seen so far whose (orthogonal) projections on the $(x,y)$-plane are not dominated by the projection of any other points already seen is efficiently maintained, using a balanced search tree with either the $x$ or $y$ coordinate as the key. At each step, the volume of a slice bounded below by the current point and bounded above by the next point is computed.

Algorithm 1 details this approach, and generalizes it to any number of dimensions. For each new point, $p$, the volume of a slice is computed by determining the measure, $v$, of the region dominated by the projection of $\mathrm{S} \cup \{p\}$ onto $(d-1)$-dimensional space, denoted $\mathrm{S}^* \cup \{p^*\}$, and multiplying it by the difference between the last coordinate of the next point, $q$, and that of $p$. Note that the asterisk is used to denote projection onto $(d-1)$-dimensional space.

The currently dominated $(d-1)$-dimensional hypervolume, $v$, is updated by finding the points in $\mathrm{S}^*$ that are dominated by $p^*$ and subtracting their individual contributions from $v$ in turn as they are removed from $\mathrm{S}^*$, before adding the individual contribution of $p^*$ to $v$ and inserting $p^*$ into $\mathrm{S}^*$. Once $v$ has been updated, the height, and thus the hypervolume, of the current slice can be easily computed by fetching the next point.

In Algorithm 1, $\mathsf{contribution}(p^*, \mathrm{S}^*, r^*)$ denotes the individual contribution of a point $p^*$ to a non-dominated point set $\mathrm{S}^*$, given a reference point $r^*$. Each point is swept and removed at most once, resulting in a total of $O(n)$ computed contributions. In the 3-dimensional case, individual 2-dimensional contributions can be computed in constant time, and the complexity of the algorithm is dominated by the time needed to find each dominated projection, $s^*$. In the 4-dimensional case, the contribution of each point (in three dimensions) to a non-dominated point set can be computed in (amortized) $O(n)$ time as it will be seen next. Since all dominated projections, $s^*$, can also be found in linear time, the 4-dimensional hypervolume indicator may be com-

**Algorithm 1** General algorithm

```
Input: X // a set of $n$ points in $\mathbb{R}^d$
Input: $r$ // $r \in \mathbb{R}^d$ is the reference point
Output: $h$ // Total hypervolume
 1: $s \leftarrow (-\infty, ..., -\infty, r^d) \in [-\infty, +\infty]^d$
 2: Q is a queue containing $X \cup s$ sorted in ascending
    order of dimension $d$
 3: $S^* \leftarrow \varnothing$
 4: $v \leftarrow 0$
 5: $h \leftarrow 0$
 6: $p \leftarrow$ dequeue(Q)
 7: while $Q \neq \varnothing$ do
 8:     for all $s^* \in S^* : p^* \leq s^*$ do
 9:         $S^* \leftarrow S^* - \{s^*\}$
10:         $v \leftarrow v -$ contribution$(s^*, S^*, r^*)$
11:     $v \leftarrow v +$ contribution$(p^*, S^*, r^*)$
12:     $S^* \leftarrow S^* \cup \{p^*\}$
13:     $q \leftarrow$ dequeue(Q)
14:     $h \leftarrow h + (q^d - p^d) \cdot v$
15:     $p \leftarrow q$
16: return $h$
```

puted in $O(n^2)$ time.

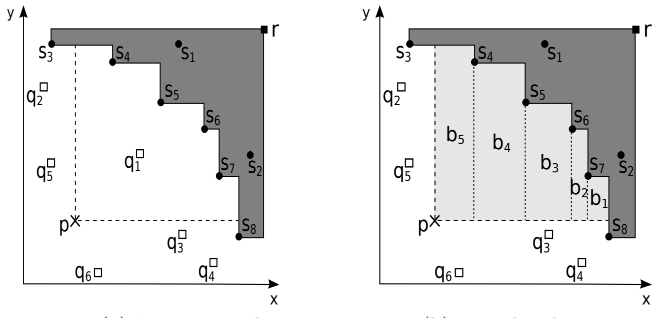
### 3.2 Individual contribution of a point in three dimensions

In order to achieve $O(n)$ time complexity in the computation of a single point contribution to a non-dominated point set $S^*$ in the conditions imposed by Algorithm 1 (i.e., $p^* \in \mathbb{R}^3$, $S^* \subset \mathbb{R}^3$ and $\nexists q^* \in S^* : p^* \leq q^*$), a method inspired in Emmerich and Fonseca's algorithm [9] is proposed.

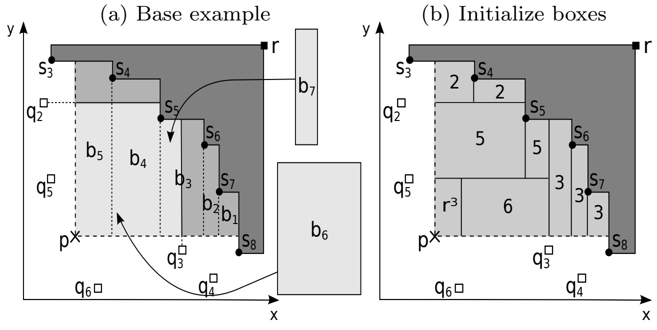
Given a non-dominated point set $S \subset \mathbb{R}^3$ and a reference point $r \in \mathbb{R}^3$, the individual contribution of each point $p$ in $S$ may be efficiently determined using the method proposed by Emmerich and Fonseca [9]. The volume dominated exclusively by each point is divided into cuboids (or boxes), and the sum of their volumes is computed. To this end, S is swept in ascending order of the $z$-coordinate[2] and the region of the ($x$,$y$)-plane exclusively dominated by each point $p$ at $z = p^z$ is partitioned into smaller non-overlapping rectangular areas. This partitioning can be obtained by sweeping those points that have coordinate $z$ lower than $p^z$ along one of the dimensions $x$ or $y$, and is used in the algorithm proposed here. Unlike in [9], where all contributions are computed, in Algorithm 2 only the contribution of a single point needs to be updated.

The example in Figure 1 will be used throughout the remainder of this Section to illustrate the computation of single-point contributions. The base problem is depicted in Figure 1a. Ignoring the presence of $q_1$ in the example of Figure 1a, as it would have been removed in a previous step, the contribution of $p$ would be computed as the sum of the volumes of the boxes depicted in Figure 1d, where the numbers indicate the corresponding heights.

[2]Note that, in [9], maximization is assumed. For clarity and consistency, the description here considers minimization instead.



(a) Base example (b) Initialize boxes



(c) Simulate closeBoxesLeft and closeBoxesRight (d) Expected result

Figure 1: Example of a problem in 3 dimensions, where the goal is to determine the contribution of $p$ to S ($S = \{q_1, ..., q_6\} \cup \{s_1, ..., s_8\}$), given the reference point $r$. In this problem, $s_t^z \leq p^z$ ($t = 1, ..., 8$) and $q_i^z > p^z$ ($i = 1, ..., 6$). It is assumed that $p^z = 0$ and $q_i^z = i$.

**Algorithm 2** HV4D - contribution

```
Input: $p \in \mathbb{R}^3$
Input: $S \subset \mathbb{R}^3$
Input: $r \in \mathbb{R}^3$ // The reference point
Output: $c$ // contribution
 1: $S_1, S_2 \leftarrow$ split$(S, p^z)$ // $S_1 = \{q \mid q \in S : q^z \leq p^z\}$,
 2:                        // $S_2 = \{q \mid q \in S : q^z > p^z\}$
 3: B $\leftarrow$ initializeBoxes$(p, S_1)$
 4: $c \leftarrow$ determineContrib$(p, S_2, B, r)$
 5: return $c$
```

The main steps of the computation of the contribution of a point $p \in \mathbb{R}^3$ to a set $S \subset \mathbb{R}^3$, as described above, are detailed in Algorithm 2. All of them can be implemented in $O(n)$ time, as long as S is a non-dominated point set and there are no points in S which are dominated by $p$, which is guaranteed to happen by Algorithm 1. Furthermore, points must be kept sorted with respect to dimension two, in order to delimit the base of the boxes (see Figure 1b), and to dimension three, to allow their heights to be determined.

## 3.3 Data structures

Algorithm 1 receives a non-dominated point set $X \subset \mathbb{R}^4$ as input, and sets up a queue Q containing all points in X in ascending order of the fourth coordinate. A sentinel is added to Q in order to ensure that point $q$, which is used to determine the height of the slice, always exists in line 13. The set $S^*$ is stored in a data structure that maintains all points sorted in ascending order of coordinates $y$ and $z$, using two doubly-linked lists. Such sorted lists are used also for the two subsets $S_1$ and $S_2$ of S in Algorithm 2, and support the following operations:

**next**$^y(p, S)$ The point following $p$ in S with respect to coordinate $y$, for $p \in S$.

**higher**$^y(p, S)$ The point $q \in S$ with the least $q^y > p^y$, for $p \notin S$.

**getXRightBelow**$(p, S)$ The point $q \in S$ with the least $q^x \geq p^x$ such that $q^y \leq p^y$

Operations next$^z$ and higher$^z$, analogous to next$^y$ and higher$^y$, are available as well. Operation next is performed in constant time as long as $p$ itself is in $S$, while the remaining operations are performed in linear time.

In Algorithm 2, the volume exclusively dominated by a point is partitioned into cuboids, here referred to as boxes. Each box $b$ is defined by its lower corner $(l^x, l^y, l^z)$ and its upper corner $(u^x, u^y, u^z)$. Boxes are kept in a doubly-linked list, B, so that boxes that need to be updated or removed may be accessed easily (in constant time). Since there is no overlap between boxes in the list, it is possible to keep the list of boxes sorted in ascending order of coordinate $x$. When a box is created, only $(l^x, l^y, l^z)$ and $(u^x, u^y)$ are known. Boxes are kept in the list as long as the corresponding value of $u^z$ is not known. Once this value is determined, the box is closed, i.e., its volume is computed, and the box is removed from the list. Then, the volume is added to the accumulated volume $c$.
In order to manage the list of boxes, the following operations are implemented:

**pushLeft**$(B, b)$ Add box $b$ to the left of the box list B

**closeAllBoxes**$(B, z)$ Close all boxes in list B, setting the corresponding value of $u^z$ to $z$, and return the sum of the volumes of those boxes.

**closeBoxesLeft**$(B, y, z)$ From left to right, close all boxes in list B for which $u^y > y$, setting the corresponding value of $u^z$ to $z$ and $l^y$ to $y$. After closing those boxes, push to the left of B a new box whose lower corner coincides with $p$, and has $u^y = y$ and $u^x$ equal to the $u^x$ of the last box removed. Finally, return the total volume of the closed boxes.

**Algorithm 3** HV4D - contribution - initializeBoxes

```
Input: $p \in \mathbb{R}^3$
Input: $S_1 \subset \mathbb{R}^3$ // $\forall q \in S \Rightarrow q^z \leq p^z$
Input: $r \in \mathbb{R}^3$ // The reference point
Output: B // Box list
 1: $S_1 \leftarrow S_1 \cup \{(r^x, -\infty, -\infty), (-\infty, r^y, -\infty)\}$
 2: $B \leftarrow \varnothing$
 3: $q \leftarrow \mathsf{higher}^y(p, S_1)$
 4: $m \leftarrow \mathsf{getXRightBelow}(p, S_1)$
 5: while $q^x > p^x$ do
 6:     if $q^x < m^x$ then
 7:         $b \leftarrow ((q^x, p^y, p^z), (m^x, q^y, p^z))$
 8:         $\mathsf{pushLeft}(B, b)$
 9:         $m \leftarrow q$
10:     $q \leftarrow \mathsf{next}^y(q, S_1)$
11: $b \leftarrow ((p^x, p^y, p^z), (m^x, q^y, p^z))$
12: $\mathsf{pushLeft}(B, b)$
13: return B
```

**closeBoxesRight**$(B, x, z)$ From right to left, close all boxes in list B for which $u^x > x$, setting their $u^z$ to $z$. If the last removed box is such that $l^x < x$, $l^x$ is updated to $x$ before closing it, and a new box is pushed to the right of B with the same corners, but with $u^x$ set to $x$. Return the total volume of the closed boxes.

Operation pushLeft is performed in constant time. Operation closeAllBoxes is performed in $k$ steps, and the remaining operations in $k + 1$ steps. Therefore, all have a cost of $O(k)$, where $k \leq n$ represents the number of boxes removed.

## 3.4 Detailed description

Algorithm 1 sweeps through every point $p$ in Q and determines the contribution of its projection on $(x,y,z)$-space, $p^*$, to the volume dominated by $S^*$. This may cause the removal of points in $S^*$ that are dominated by $p^*$. Point removal can be performed in constant time, but requires the computation of the corresponding contributions, as well. After computing its individual contribution, $p^*$ is added to $S^*$ while keeping the lists used to maintain $S^*$ sorted in ascending order of both $y$ and $z$ coordinates, which can be implemented in linear time. Furthermore, Algorithm 1 guarantees that, when calculating the contribution of any point $p^*$, all points in $S^*$ are kept sorted in ascending order of coordinates $y$ and $z$, and that no point in $S^*$ is dominated by any other point in $S^*$ or by $p^*$ itself. As explained before, these constraints allow linear-time computation of a point's contribution.

Algorithm 2 computes the 3-dimensional contribution of $p$ to a set of points S. The computation consists of two parts: the bases of an initial set of boxes are

**Algorithm 4** HV4D - contribution - determineContrib

```
Input: $p \in \mathbb{R}^3$
Input: $S_2 \subset \mathbb{R}^3$ // $\forall q \in S \Rightarrow q^z > p^z$
Input: B is a list of boxes
Output: $c$ // contribution
1: $S_2 \leftarrow S_2 \cup \{(-\infty, -\infty, r^z)\}$
2: $q \leftarrow \mathsf{higher}^z(p, S_2)$
3: while not $empty(B)$ do
4:   if $q^x \leq p^x$ then
5:     if $q^y \leq p^y$ then
6:       $c \leftarrow c + \mathsf{closeAllBoxes}(B, q^z)$ // Case 3
7:     else
8:       $c \leftarrow c + \mathsf{closeBoxesLeft}(B, q^y, q^z)$ // Case 1
9:   else
10:    $c \leftarrow c + \mathsf{closeBoxesRight}(B, q^x, q^z)$ // Case 2
11:  $q \leftarrow \mathsf{next}^z(q, S_2)$
12: return $c$
```

determined first (Algorithm 3), and then box heights are found (Algorithm 4).

To determine the bases of the boxes, the points in S whose $z$ coordinate is lower than or equal to $p^z$ ($S_1$) and which are dominated by $p$ with respect to the $x$ and $y$ coordinates, but not by any other point in $S_1$, are swept (points $s_4, ..., s_7$ in Figure 1a). Boxes are created from right to left by sweeping through points in $S_1$ in ascending order of coordinate $y$, starting from point $\mathsf{higher}^y(p, S_1)$ ($s_7$), which is the lowest point in $S_1$ higher than $p^y$, and stopping when a point to the left of $p$ is found ($s_3$). Note that such points always exist because of the presence of the sentinel $(-\infty, r^y, -\infty)$, although this is not shown in Figure 1. All points between the starting point ($s_7$) and the end point ($s_3$) that do not fulfill all the above conditions are skipped ($s_2$). Each of the points that satisfy the above conditions ($s_7$, $s_6$, $s_5$, $s_4$) defines $l^x$ and $u^y$ of a box as well as $u^x$ of the next box. For example, in Figure 1b, point $s_5$ defines $l^x$ and $u^y$ of box $b_3$ and $u^x$ of box $b_4$. The value of $u^x$ for the first and rightmost box created is determined by getXRightBelow($p$) ($s_8$), and is guaranteed to exist due of the sentinel $(r^x, -\infty, -\infty)$. Finally, the end point $s_3$ only defines $u^y$ of the last and leftmost box. In the example of Figure 1a, after executing the first part of the algorithm, B contains $b_1, ..., b_5$ as depicted in Figure 1b.

The next step, determineContrib, consists of determining the height of boxes and closing them and, in some cases, initializing a new box (Algorithm 4). The total area covered by boxes in the $(x, y)$-plane shrinks after each step. Only points with coordinate $z$ higher than $p^z$ ($S_2$) need to be considered ($q^2, ..., q^6$). Therefore, points in $S_2$ are swept in ascending order of coordinate $z$ as long as there are still boxes to be closed. While processing each point $q$, three cases are considered, depending on the projection of $q$ on the $(x, y)$-plane:

Case 1: $q$ is to the left of and above $p$ (e.g. $q_2, q_5$)

Case 2: $q$ is to the right of and below $p$ (e.g. $q_3, q_4, q_6$)

Case 3: $q$ dominates $p$ (e.g. the sentinel $(-\infty, -\infty, r^z)$, which is not represented in Figure 1)

Note that $q$ is never dominated by $p$ on the ($x$,$y$)-plane, because it would also be dominated in $(x, y, z)$-space in that case, but those points were previously removed in Algorithm 1.

Cases 1, 2 and 3 cause the algorithm to call functions closeBoxesLeft, closeBoxesRight and closeAllBoxes, respectively. Figure 1c shows an example of what happens when cases 1 or 2 occur. The darker regions of boxes $b_5$ and $b_4$ (respectively, $b_1$, $b_2$ and $b_3$) represent the boxes that are shrunk and closed when case 1 (case 2) occurs while processing $q_2$ ($q_3$). After closing those boxes, box $b_6$ ($b_7$) is inserted to the left (right) of the box list to account for the area left uncovered due to the shrinking of the boxes before they are closed. Every function returns the total volume of the closed boxes. When case 3 occurs, all boxes are closed and the algorithm can terminate, as there are no more box heights to be determined.

## 3.5 Complexity

It is not difficult to see that the complexity of Algorithm 2 is $O(n)$. Splitting S into two subsets ($S_1$ and $S_2$), and each of the two remaining stages of the algorithm can be performed in $O(n)$ time. Regarding the first stage (initializeBoxes), note that for each point in $S_1$ with coordinate $y$ greater than $p^y$, of which there are at most $n$ points, at most one box is created (in constant time). Therefore, $O(n)$ complexity is achieved. The second stage processes all points with coordinate $z$ greater than $p^z$, which are also at most $n$ points. For each of these points, $k$ boxes are closed, $k \in [0, n]$. Moreover, at most one box is created, which can happen only if at least one box is closed. Note that if there are $t$ points with third coordinate lower or equal to $p^z$, then the first stage can create up to $t$ boxes, while the second stage can create at most $n - t$ boxes, which gives a total of up to $n$ boxes created. Therefore, the maximum number of closed boxes is also $n$. Independently of which function is used to close boxes (closeBoxesRight, closeBoxesLeft or closeAllBoxes) $k$ steps are performed if boxes are only closed, or $k + 1$ steps, if a box is also created, leading to $O(k)$ cost either way. Therefore, the total cost of the operations of Algorithm 2 amortizes to $O(n)$.

Algorithm 1 sweeps through $n$ points and, for each point $p$, it determines the points in $S^*$ with greatest $y$ and $z$ coordinates lower than $p^y$ and $p^z$, respectively, in order to keep the lists of points associated with $S^*$ sorted, at a cost of $O(n)$. Moreover, for each point swept, the individual contributions of $k$ dominated points and the contribution of the current point are computed. Since each point in the original set X is added to $S^*$ and removed from it at most once, the algorithm computes at most $2n$ contributions, each at a cost of $O(n)$ using Algorithm 2, leading to $O(n^2)$ amortized time complexity.

## 4 Concluding remarks

A C-language implementation of the algorithm proposed here confirms that it can be implemented efficiently, and that no large constants hide in the $O$ notation [11]. Following the same ideas, it may be possible to obtain a dimension-sweep algorithm for $d = 5$ with complexity $O(n^2 \log n)$, since at least a 4-dimensional analogue of initializeBoxes with $O(n \log n)$ complexity would be easily constructed. This would match Yıldız and Suri's upper bound [17] for the 5-dimensional case.

**Acknowledgement** Andreia P. Guerreiro gratefully acknowledges funding from Fundação para a Ciência e Tecnologia (FCT), Portugal, through Ph.D. grant SFRH/BD/77725/2011.

## References

[1] N. Beume. S-metric calculation by considering dominated hypervolume as Klee's measure problem. *Evol. Comput.*, 17:477–492, Dec. 2009.

[2] N. Beume, C. M. Fonseca, M. López-Ibáñez, L. Paquete, and J. Vahrenhold. On the complexity of computing the hypervolume indicator. *IEEE Trans. Evol. Comput.*, 13(5):1075–1082, 2009.

[3] N. Beume, B. Naujoks, and M. Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *Eur. J. Oper. Res.*, 181(3):1653–1669, 2007.

[4] L. Bradstreet, L. While, and L. Barone. A fast many-objective hypervolume algorithm using iterated incremental calculations. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 179–186, July 2010.

[5] K. Bringmann. Klee's measure problem on fat boxes in time $O(n^{(d+2)/3})$. In *26th Symposium on Computational geometry (SoCG)*, pages 222–229, New York, NY, USA, 2010. ACM.

[6] K. Bringmann and T. Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. In S.-H. Hong et al., editors, *Algorithms and Computation*, volume 5369 of *LNCS*, pages 436–447. Springer Berlin / Heidelberg, 2008.

[7] T. M. Chan. Semi-online maintenance of geometric optima and measures. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '02, pages 474–483, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[8] T. M. Chan. A (slightly) faster algorithm for Klee's measure problem. *Computational Geometry*, 43:243–250, 2010.

[9] M. Emmerich and C. M. Fonseca. Computing hypervolume contributions in low dimensions: Asymptotically optimal algorithm and complexity results. In R. H. C. Takahashi et al., editors, *EMO 2011*, volume 6576 of *LNCS*, pages 121–135. Springer Berlin / Heidelberg, 2011.

[10] C. M. Fonseca, L. Paquete, and M. López-Ibáñez. An improved dimension-sweep algorithm for the hypervolume indicator. In *IEEE Congress on Evolutionary Computation (CEC 2006)*, pages 1157–1163, Piscataway, NJ, July 2006. IEEE Press.

[11] A. P. Guerreiro. Efficient algorithms for the assessment of stochastic multiobjective optimizers. Master's thesis, IST, Technical University of Lisbon, Portugal, 2011.

[12] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22(4):469–476, 1975.

[13] M. H. Overmars and C.-K. Yap. New upper bounds in Klee's measure problem. *SIAM J. Comput.*, 20(6):1034–1045, 1991.

[14] T. Wagner, B. Nicola, and B. Naujoks. Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In S. Obayashi et al., editors, *EMO 2007*, volume 4403 of *LNCS*, pages 742–756, Berlin, Heidelberg, 2007. Springer-Verlag.

[15] L. While, L. Bradstreet, and L. Barone. A fast way of calculating exact hypervolumes. *IEEE Trans. Evol. Comput.*, 16(1):86–95, 2012.

[16] L. While, P. Hingston, L. Barone, and S. Huband. A faster algorithm for calculating hypervolume. *IEEE Trans. Evol. Comput.*, 10(1):29–38, Feb. 2006.

[17] H. Yıldız and S. Suri. On Klee's measure problem on grounded boxes. In *28th Symposium on Computational Geometry (SoCG)*, Chapel Hill, North Carolina, USA, June 2012.

[18] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms – A comparative case study. In A. E. Eiben et al., editors, *Parallel Problem Solving from Nature, PPSN V*, volume 1498 of *LNCS*, pages 292–301. Springer, Heidelberg, 1998.