Computing hypervolume contributions in low dimensions: asymptotically optimal algorithm and complexity results

Michael T. M. Emmerich[1,2] and Carlos M. Fonseca[3,2,1]

[1] DEEI, Faculty of Science and Technology, University of Algarve
Campus de Gambelas, 8005-139 FARO, Portugal
mtemmerich@ualg.pt
[2] CEG-IST, Instituto Superior Técnico, Technical University of Lisbon
Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal
[3] Department of Informatics Engineering, University of Coimbra,
Pólo II, 3030-290 Coimbra, Portugal
cmfonsec@dei.uc.pt

Abstract. Given a finite set $\mathrm{Y} \subset \mathbb{R}^d$ of n mutually non-dominated vectors in $d \geq 2$ dimensions, the hypervolume contribution of a point $\mathbf{y} \in \mathrm{Y}$ is the difference between the hypervolume indicator of Y and the hypervolume indicator of $\mathrm{Y} \setminus \{\mathbf{y}\}$. In multi-objective metaheuristics, hypervolume contributions are computed in several selection and bounded-size archiving procedures.
This paper presents new results on the (time) complexity of computing all hypervolume contributions. It is proved that for $d = 2, 3$ the problem has time complexity $\Theta(n \log n)$, and, for $d > 3$, the time complexity is bounded below by $\Omega(n \log n)$. Moreover, complexity bounds are derived for computing a single hypervolume contribution.
A dimension sweep algorithm with time complexity $\mathcal{O}(n \log n)$ and space complexity $\mathcal{O}(n)$ is proposed for computing all hypervolume contributions in three dimensions. It improves the complexity of the best known algorithm for $d = 3$ by a factor of $\sqrt{n}$. Theoretical results are complemented by performance tests on randomly generated test-problems.

Keywords: multiobjective selection, complexity, hypervolume indicator

1 Introduction

The *hypervolume indicator* (or S-metric, Lebesgue measure) was introduced by Zitzler and Thiele [26] to measure the quality of Pareto front approximations. Given a finite set Y of mutually non-dominated vectors in $\mathbb{R}^d$, the hypervolume indicator measures the volume (Lebesgue measure) of the subspace simultaneously dominated by Y and bounded below (in case of maximization) by a reference point.

Besides being frequently used as a performance metric, the hypervolume indicator is also used in guiding selection in indicator-based metaheuristics [10, 11, 13, 14, 19, 25] and archivers [16]. In this context, the problem of computing hypervolume contributions and/or the minimal hypervolume contributor of a set of points often arises [10, 13, 14, 16, 20, 25]. The hypervolume contribution of a point $\mathbf{y} \in \mathrm{Y}$ is defined as the difference between the hypervolume indicator of Y and the hypervolume indicator of $\mathrm{Y} \setminus \{\mathbf{y}\}$. The problem of finding the minimal hypervolume contributor is #P-hard in the number of dimensions d [8].

Many applications of multiobjective optimization involve a small number of objectives, say 2–4. In these cases, polynomial time algorithms for computing hypervolume contributions are known (e.g. [9, 20]), but the extent to which they are efficient is so far unknown. When objective functions can be computed fast, hypervolume computations can limit the performance of multiobjective optimization algorithms also in low dimensions.

Hence, this paper focuses on computing hypervolume contributions in low dimensions and analyzes complexity in the cardinality, n, of the input set, Y. One aim is to derive sharper complexity bounds for the computation of all (or single) hypervolume contributions. Moreover, it is shown that the dimension sweep paradigm, that has yielded asymptotically optimal algorithms for computing the hypervolume indicator [4] and maximal set [18] in low dimensions, can also allow hypervolume contributions to be computed efficiently.

The paper is organized as follows: In Section 2, several problems related to hypervolume contributions are introduced. A summary of known results and related work on these problems is given. In Section 3, lower bounds on the complexity of computing hypervolume contributions are established. In Section 4, an algorithm that efficiently computes the hypervolume contributions given a set in three dimensions is introduced and analyzed. In Section 5, performance tests on randomly generated test data are conducted. Finally, in Section 6, contributions of the paper are summarized and suggestions for future research are given.

2 Preliminaries and related work

The following conventions will be used throughout the paper. Complexity, unless stated otherwise, refers to *time complexity in the worst case*. The algebraic decision tree model of computation is considered [17]. Asymptotic lower, upper, and sharp bounds are denoted by $\Omega()$, $\mathcal{O}()$, and $\Theta()$, respectively. Vectors are represented in bold font. Sets are denoted by roman capitals, e.g. X or Y, and problems and algorithm names are typeset in small capitals, e.g. allContributions. When applied to vectors, operators $\leq$, $\geq$ and $=$ indicate componentwise comparison. Throughout this paper, maximization is the goal.

The following concepts will be relevant in the problem definition:

Definition 1. *(dominance) A vector* $\mathbf{y} \in \mathbb{R}^d$ dominates $\mathbf{y}' \in \mathbb{R}^d$, *iff* $\mathbf{y} \geq \mathbf{y}'$ *and* $\mathbf{y} \neq \mathbf{y}'$. *In symbols:* $\mathbf{y} \succ \mathbf{y}'$.

Dominance is often defined with minimization of vectors as the goal. In this case $\geq$ must be replaced by $\leq$ in the above definition.

Definition 2. *(approximation set) A d-dimensional approximation set [27] is a finite set* $\mathrm{Y} \subset \mathbb{R}^d$ *such that* $\forall \mathbf{y}, \mathbf{y}' \in \mathrm{Y} : \mathbf{y} \geq \mathbf{y}' \Rightarrow \mathbf{y} = \mathbf{y}'$. *The set of all d-dimensional approximation sets is denoted by* $\mathbb{A}^d$.

Definition 3. *(hypervolume indicator ($\mathcal{S}$)) Given a set* $\mathrm{Y} \subset \mathbb{R}^d$ *and a reference point* $\mathbf{y}^r$ *that satisfies* $\forall \mathbf{y} \in \mathrm{Y} : \mathbf{y} \geq \mathbf{y}^r$, *the hypervolume indicator of* Y *with respect to reference point* $\mathbf{y}^r$ *is defined as [26, 24]:*

$$\mathcal{S}(\mathrm{Y}) = \mathrm{Vol}\left(\bigcup_{\mathbf{y} \in \mathrm{Y}} [\mathbf{y}^r, \mathbf{y}]\right). \tag{1}$$

Here $\mathrm{Vol}()$ *denotes the Lebesgue measure in d dimensions and* $[\mathbf{l}, \mathbf{u}] \subset \mathbb{R}^d$ *represents a closed axis-parallel box that is bounded below by* $\mathbf{l} \in \mathbb{R}^d$ *and bounded above by* $\mathbf{u} \in \mathbb{R}^d$.

Whenever the reference point is not explicitly mentioned, by convention, $\mathbf{y}^r = \mathbf{0}$ will be assumed.

Definition 4. *(hypervolume contribution of a point) Given an approximation set* $\mathrm{Y} \in \mathbb{A}^d$, *the hypervolume contribution of a point* $\mathbf{y} \in \mathrm{Y}$ *is defined as*

$$\Delta\mathcal{S}(\mathbf{y}, \mathrm{Y}) = \mathcal{S}(\mathrm{Y}) - \mathcal{S}(\mathrm{Y} \setminus \{\mathbf{y}\})$$

This paper is mainly concerned with the following problem:

*Problem 1 (*AllContributions*).* Given $\mathrm{Y} \in \mathbb{A}^d$ as an input, compute the hypervolume contributions of all points $\mathbf{y} \in \mathrm{Y}$.

In addition new results on closely related problems, that can all be reduced in at most linear time to AllContributions, will be derived:

*Problem 2 (*OneContribution*).* Given $\mathrm{Y} \in \mathbb{A}^d$ and $\mathbf{y} \in \mathrm{Y}$ as an input, compute the hypervolume contribution $\Delta\mathcal{S}(\mathbf{y}, \mathrm{Y})$.

*Problem 3 (*MinimalContribution*).* Given $\mathrm{Y} \in \mathbb{A}^d$ as an input, compute the minimal hypervolume contribution, i.e., $\min_{\mathbf{y} \in \mathrm{Y}} \Delta\mathcal{S}(\mathbf{y}, \mathrm{Y})$.

*Problem 4 (*MinimalContributor*).* Given $\mathrm{Y} \in \mathbb{A}^d$ as an input, find a point with minimal hypervolume contribution, i.e., $\mathbf{y}^* \in \arg\min_{\mathbf{y} \in \mathrm{Y}} \Delta\mathcal{S}(\mathbf{y}, \mathrm{Y})$.

A straightforward algorithm to solve AllContributions consists of enumerating all subsets of size $n - 1$, computing their hypervolumes, and subtracting each of these in turn from the hypervolume of the total set. Bringmann and Friedrichs [7] show that computing the hypervolume is #P-hard in the number of dimensions d. To compute the hypervolume, for $d = 3$, the dimension sweep algorithm by Fonseca et al.[12] is asymptotically optimal (Beume et al.[4]). For general d, computing the hypervolume indicator can be considered a special case of computing the measure of a union of rectangles [3]. The best known algorithm has a complexity $\mathcal{O}(n^{d/2} \log n)$ (cf. [21]) and has been simplified for the special case of computing hypervolume by Beume [3], though the complexity stays the same. A dimension sweep algorithm with complexity $\mathcal{O}(n^{d-2} \log n)$ is described by Fonseca et al. [12]. It has lower complexity for $d = 3$ and the same complexity for $d = 4$. Algorithms with a higher worst case complexity proposed in While et al. [23] have proved to be competitive or faster on problems of moderate size and dimensions.

A specialized algorithm for computing hypervolume contributions that guarantees a better worst-case performance has been proposed by Bringmann and Friedrichs [9] based on Overmars and Yap's algorithm, [21]. Its time complexity is $\mathcal{O}(n^{d/2} \log n)$ for $d > 2$. Note, that their algorithm also solves the more general problem of finding the hypervolume contributions of a finite set $\mathrm{Y} \in \mathbb{R}^d$, i.e. it is not required that points in the set are mutually non-dominated. For $d = 2$, the problem reduces simply to sorting a set of n points ($\mathcal{O}(n \log n)$) (e.g. [16]). Another algorithm for computing *incremental hypervolumes* has been proposed by Bradstreet et al. [6], who also describe an algorithm for updating all contributions in [5], the worst case complexity of which is $\mathcal{O}(n^{d-1})$. Empirical studies show a high performance of these schemes in case of many objectives.

For high dimensions, fast heuristics and approximation algorithms have been proposed. For instance, Bader and Zitzler [2] propose Monte Carlo algorithms that work also for many objectives. Fast approximation algorithms with accuracy guarantees are proposed by Bringmann and Friedrichs [8]. Approximate integration based on scalarization is suggested in [15].

Despite this progress, sharp complexity bounds for AllContributions, OneContribution, MinimalContribution, and MinimalContributor have remained unavailable to date.

3 Complexity bounds

This section will start with a theorem on a lower bound on AllContributions and proceed with the discussion of lower bounds on computing single contributions. To obtain a lower bound on the complexity of AllContributions a reduction to UniformGap, a well-known problem from computational geometry, is used.

*Problem 5 (*UniformGap). The problem of deciding whether a set of n points on the real line is equidistant is called UniformGap.

Lemma 1. *(Preparata and Shamos [22], p. 260) The complexity of* UniformGap *is $\Omega(n \log n)$ in the algebraic decision tree model of computation.*

Now, the theorem on the complexity of AllContributions reads:

Theorem 1. *Any algorithm that solves* AllContributions *in $d > 1$ dimensions requires $\Omega(n \log n)$ time in the algebraic decision tree model of computation.*

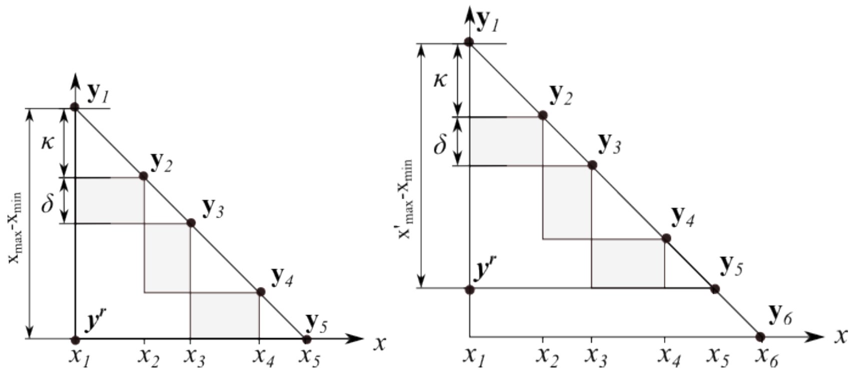

Fig. 1. Problem transformation. Odd cardinality (left) and even cardinality (right)

Proof of theorem 1: It will be shown that UniformGap reduces in linear time to AllContributions. (For the general idea of a reduction proof see, e.g., [4].) Given a set X of n values x_i on the real line as input to UniformGap, each coordinate is augmented with a second coordinate, which yields $\mathrm{Y} = \{(x_i, x_{max} - x_i) : i \in \{1, \dots, n\}$, where $x_{max} = \max\{x_i : i = 1, \dots, n\}$ (see Fig. 1).

Case 1 (n is odd): Compute all hypervolume contributions of points using $\mathbf{y}^r = (x_{min}, x_{min})$, $x_{min} = \min(X)$ as reference point, see Fig. 1. Next, all hypervolume contributions are computed and compared to $((x_{max} - x_{min})/(n-1))^2$. If they are all equal to this value the answer to UniformGap is positive, otherwise it is negative. This algorithm is correct for odd n, since all hypervolume contributions of inner points are equal to $((x_{max} - x_{min})/(n-1))^2$ if and only if gaps are uniform. This will be shown next: Consider a line segment from $\mathbf{y}_1 = (x_{min}, x_{max})$ to $\mathbf{y}_n = (x_{max}, x_{min})$. As all intermediate points in Y lie on the line segment, the hypervolume contribution equality condition enforces a certain alternating pattern of rectangles bounding the contributions (see Fig. 1). The rectangles are congruent, but rotate by 90° in each step. The side-length of the rectangles is δ for the shorter side and κ for the longer side. Because the number of points on the line segment is odd

$$x_{max} - x_{min} = \frac{n-1}{2}(\delta + \kappa) \text{ and thus } \kappa = \frac{2(x_{max} - x_{min})}{(n-1)} - \delta. \qquad (2)$$

The contribution of each inner point x is given by $c(x) = \delta\kappa$. To maximize c over all δ and κ eliminate κ in $c(x) = \delta\kappa$, which leads to the problem:

$$\text{maximize } c(\delta) = \delta\left(\frac{2(x_{max} - x_{min})}{n-1} - \delta\right) \qquad (3)$$

The equation describes a concave parabola, the unique maximizer δ^* of which is the point with zero derivative. Solving $2\frac{(x_{max} - x_{min})}{n-1} - 2\delta^* = 0$ yields

$$\delta^* = \frac{x_{max} - x_{min}}{n-1} \qquad (4)$$

Inserting δ^* in Equation 3 yields $\kappa^* = \delta^*$. This solution is the only solution with (maximal) hypervolume contribution $c = (\delta^*)^2 = ((x_{max} - x_{min})/(n-1))^2$ for all points $\mathrm{Y} \setminus \{\mathbf{y}_1, \mathbf{y}_n\}$.

Case 2 (n is even): Let $\mathrm{X}' = \mathrm{X} \setminus \{\max(\mathrm{X})\}$. Let $x'_{max} = \max(\mathrm{X}')$ and $x'_{min} = x_{min}$. Given that the uniform gap condition is fulfilled for X' (odd size) and $x_{max} - x'_{max} = (x'_{max} - x'_{min})/(n-2)$ the answer is positive, otherwise not. The complexity of the extra steps in this algorithm is $\mathcal{O}(N)$, excepting perhaps the time for solving AllContributions. Thus, if AllContributions could be solved faster than $\mathcal{O}(n \log n)$, then also UniformGap could be solved faster than $\mathcal{O}(n \log n)$, which would contradict lemma 1.

The result generalizes to $d > 2$ as otherwise the problem AllContributions in two dimensions could be solved by a linear time reduction to AllContributions in higher dimensions. It would suffice to set all $d-2$ additional coordinates of $\mathbf{y}$ to 1 and the reference point to $\mathbf{0}$ and solve the problem. □

A similar result can be obtained for computing single contributions:

Theorem 2. *The time complexity of* OneContribution *is* $\Theta(n)$ *for* $d = 2$*,* $\Theta(n \log n)$ *for* $d = 3$*, and for* $d > 3$ *it is bounded below by* $\Omega(n \log n)$*.*

Proof: Case $d = 2$: All points need to be considered, yielding $\Omega(n)$ as lower bound; computing a contribution requires $\mathcal{O}(n)$ time (nearest neighbor search). Case $d = 3$: The problem of computing the hypervolume of a set in two dimensions (Hypervolume2d) can be reduced in linear time to OneContribution in three dimensions. The complexity of Hypervolume2D is bounded below by $\Omega(n \log n)$ (cf. [4]). To compute the solution of Hypervolume2D for an $\mathrm{Y} \in \mathbb{A}^2$ using OneContribution in 3-D, represent Y in three dimensions by augmenting all points Y with a z-coordinate of 2, resulting in $\mathrm{Z} \in \mathbb{A}^3$. Create a point $(x_{max}, y_{max}, 1)$ in $\mathcal{O}(n)$ time with x_{max} being the maximum x-coordinate and y_{max} the maximum y-coordinate in Y. The solution of Hypervolume2D is $x_{max}y_{max} - \Delta\mathcal{S}((x_{max}, y_{max}, 1), (x_{max}, y_{max}, 1) \cup \mathrm{Z})$. The same principle can be used for proving a lower bound of $\Omega(n \log n)$ for $n > 3$. The upper bound of $\mathcal{O}(n \log n)$ for $n = 3$ is due to the fact that the computation of a single contribution of a point $\mathbf{y}$ in Y reduces to computing the difference $\mathcal{S}(\mathrm{Y}) - \mathcal{S}(\mathrm{Y} \setminus \{\mathbf{y}\})$, and the computation of each hypervolume takes $\mathcal{O}(n \log n)$ (see [12]). □

4 Dimension sweep algorithm

A dimension sweep algorithm that computes the hypervolume contributions for a set $\mathrm{Y} \in \mathbb{A}^3$ in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space is introduced and discussed. The dimension sweep paradigm seems to be promising for constructing the algorithm, as it has already yielded asymptotically optimal algorithms for the maximal set problem [18] and the computation of the total hypervolume in 3-D [12].

The algorithm processes points in the order of descending z-coordinates. For the i-th such level, the z-coordinate of the i-th point defines the height of a sweeping plane that stretches out in x- and y-direction (see figure 2).

The construction of the algorithm is based on the observation that the region above the sweeping plane that is exclusively dominated by a point when considering only the subset of points above the sweeping plane, will be also exclusively dominated – by the same point – when the entire set of points is considered. Hence hypervolume contributions can be accumulated level by level. This can be done by partitioning the exclusively dominated hypervolume into axis-parallel boxes. At each level, it is made sure that each exclusively dominated region above the sweeping plane is covered by exactly one such box. As the sweeping plane reaches lower levels new boxes may be *created*, existing boxes may grow (in their z-coordinate), and boxes may be *completed*, i.e., they will no longer grow. Each box is associated with the point that exclusively dominates the hypervolume it encloses. After the sweeping plane has reached the level of the reference point, the exclusive hypervolume contribution of a point is the sum of the volumes of the completed boxes that have been associated with this point.

To achieve efficiency, the algorithm maintains lists of boxes in such a way that each box that needs to be updated can be efficiently accessed. This can be achieved by maintaining boxes in doubly linked lists associated with the points. Only *active* boxes are maintained, i.e. boxes that may still grow. Boxes that are completed are removed from these lists, and their hypervolume is added to the point to which they belong. Moreover, efficient access to points, the box lists of which need to be updated, is achieved by maintaining a tree that comprises a subset of points that have already been visited [18]. Accordingly, the following non-elementary *data-structures* are used in the algorithm:

– A height balanced search tree (e.g. AVL tree [1]), named T, the nodes of which refer to the non-dominated subset of input points that have been processed so far by the algorithm. It is sorted in ascending order of the x-coordinate and features insertion of a point (T.insert), deletion of a point (T.delete) (both in time $\mathcal{O}(\log n)$), and range selection in $\mathcal{O}(\log(n) + k)$, where n is the number of points and k the number of qualified items.
– An axis-parallel box $\mathbf{b}$ is defined by its lower corner ($\mathbf{b}$.lx, $\mathbf{b}$.ly, $\mathbf{b}$.lz), and its upper corner ($\mathbf{b}$.ux, $\mathbf{b}$.uy, $\mathbf{b}$.uz). *Active* boxes have an undefined lower bound $\mathbf{b}$.lz=`NaN` whereas all other coordinates are set to their final value after a box has been created.
– An array of doubly linked lists of active boxes: Each point is associated with one box list. List are sorted in ascending x-coordinate. A box list, say L, has these methods: L.push_front($\mathbf{b}$) adds a box item to the list's head, and L.push_back($\mathbf{b}$) to its tail. Moreover, $\mathbf{b}$=L.pop_front() retrieves a box from the head of a list and $\mathbf{b}$=L.pop_back() from the tail, removing the corresponding box from the list.

A detailed outline of the algorithm in pseudocode is given in algorithm 1. The array of points is sorted by the z-coordinate. Sorting can be achieved in $\mathcal{O}(n \log n)$ and therefore, due to theorem 1, it does not impose significant additional costs onto the algorithm.

Algorithm 1 can be subdivided in an initialization phase and the main loop. *Initialization:* The algorithm intializes an empty AVL tree (T) and inserts the

Algorithm 1 Algorithm HYCON3D

```
    Input: $(\mathbf{p}[1], ..., \mathbf{p}[n])$: mutually non-dominated $\mathbb{R}^3$-points sorted by $z$-coordinate
    in descending order
    (1) $\mathbf{p}$[n+1]=$(\infty, \infty, 0)$;
    (2) Initialize AVL tree T for 3-D points
        Insert $\mathbf{p}$[1], $(\infty, 0, \infty)$, $(0, \infty, \infty)$ into T;
    (3) Initialize doubly linked lists L[1]=empty(); ... L[n+1]=empty();
        b= ((0,0,NaN), (p[1].x, p[1].y, p[1].z)); L[1].push_front(b);
    (4) Initialize hypervolume contributions $c[1] = 0$; ...; $c[n] = 0$
    for i= 2 to n+1 do {Main Loop}
    (a) Retrieve the following information from tree T:
            r: index of the successor of p[i] in x-coordinate (right neighbor)
            t: index of the successor of p[i] in y-coordinate (left neighbor)
            d[1], ..., d[s]: indices of points dominated by p[i] in xy-plane,
            sorted ascending in $x$-coordinate (region B, Figure 3).
    (b) while not L[r].empty()    {Process right neighbor, region R}
            b= L[r].pop_front()
            if (b.ux $\leq$ p[i].x)
                b.lz=p[i].z; c[r]=c[r]+VOL(b);
            else if (b.lx $<$ p[i].x)
                b.lz=p[i].z; c[r]=c[r]+VOL(b);
                b.lx=p[i].x; b.uz=p[i].z; b.lz=NaN; {Add box $b^r$ in region R}
                L[r].push_front(b); break;
            else L[r].push_front(b); break
    (c) xleft = p[t].x    {Process dominated points, region M}
        for j=1 to s
            jdom=d[s]; d=p[jdom];
            while (not L[jdom].empty())
                b=L[jdom].pop_front();
                b.lz=p[i].z; c[jdom]=c[jdom]+VOL(b);
            b=[(xleft, d.y, NaN),(d.x, p[i].y, p[i].z)];
            L[i].push_back(b);
            xleft = b.ux;
            remove p[jdom] from AVL tree
        b=[(xleft, p[r].y, NaN), (p[i].x, p[i].y, p[i].z)]; {Add box $b^+$ in region R}
        L[i].push_back(b);
    (d) xleft=p[t].x;    {Process left neighbor, region L}
        while not L[t].empty()
            b=L[t].pop_back();
            if (b.ly $<$ p[i].y)
                b.lz = p[i].z; c[t]=c[t]+VOL(b);
                xleft = b.lx
            else L[t].push_back(b); break;
        if (xleft $<$ p[t].x)
            b=[(xleft, p[i].y, NaN),(p[t].x, p[t].y, p[i].z)]; {Add box $b^l$ in region R}
            L[t].push_back(b);
    (e) Insert p[i] in AVL tree T;
(5) output c[1], ..., c[n] {Exclusive contributions}
```

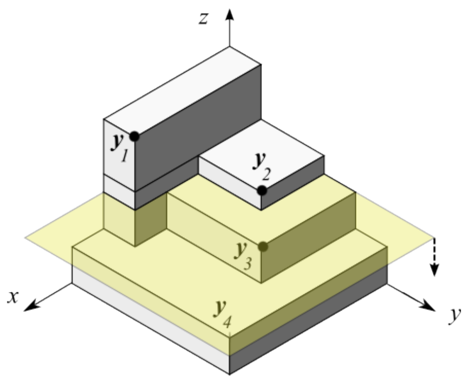

Fig. 2. Sweeping plane

highest point $\mathbf{p}[1]$ into T. Also, two auxiliary points $(\infty, 0, \infty)$ and $(0, \infty, \infty)$ that bound the region from the left and from the right are inserted. They have either an x or a y coordinate of 0, respectively, such that they never dominate points with positive hypervolume contribution. Here, the value of 0 stems from the reference point $\mathbf{c} = \mathbf{0}$. Their z-coordinate is infinite, such that they are never dominated themselves by input points and remain in the tree. Throughout the algorithm they serve to make sure that every point in the xy-plane has a well defined neighbor in the x- and in the y-coordinate.

Additionally, an auxiliary point $\mathbf{p}[n+1]$=(∞, ∞, 0) is created. It receives the index $n+1$. It will be processed in the last iteration of the main loop. Again, the value of 0 stems from the reference point. Its projection onto the xy-plane is (∞, ∞) and dominates all projections of input points to the xy-plane. This point will force the completion of boxes that are still open in the last iteration. The array L of box lists is then initialized with empty lists. A point is inserted into the box list of point $\mathbf{p}[1]$ with ($\mathbf{p}[1].x$, $\mathbf{p}[1].y$, $\mathbf{p}[1]$.z) and the x-and y-coordinate of the reference point as a lower corner. The lower limit of the box in the z-coordinate is determined later in the algorithm. This box delimits the region that is dominated exclusively by the first point above the second level of the sweeping plane. Finally, the array c[1], ..., c[n] that is used to accumulate hypervolume contributions of completed boxes is initialized to zero.

The *main loop* of the algorithm processes in ascending order of the z-coordinate all points $\mathbf{p}[2]$, ... $\mathbf{p}[n+1]$. It maintains the following invariant properties:

- At the end of each iteration, the volume of all active boxes in the box lists, if they were completed at the current z-level, plus the accumulated volumes of previously completed boxes, is the total exclusively dominated hypervolume above the sweeping plane at level i. This property is essential to guarantee the correctness of the algorithm.
- The box lists of each point contains active boxes sorted by ascending x coordinates. This property allows efficient updates of box lists.

– To find relevant information for the update of box lists, in the i-th iteration, T stores points that are non-dominated among $\mathbf{p}[1]$, ..., $\mathbf{p}[i-1]$ in the xy-plane. These are the only relevant points for determining increments of hypervolume contributions from level i onwards. T is organized as a balanced search tree sorted by x-coordinate.

After the introduction of a new point $\mathbf{p}[i]$ some boxes simply grow in z-direction and no access by the algorithm is required for them, whereas others need to be created or completed and require access. From Figure 3, it becomes clear that for the creation and completion of boxes only box lists of particular points are of interest. These are $\mathbf{p}[r]$, the upper neighbor in T of $\mathbf{p}[i]$ in the x-direction, $\mathbf{p}[t]$, the upper neighbor in T of $\mathbf{p}[i]$ in the y-direction, and the sequence of all dominated points $\mathbf{p}[\mathrm{d}[1]]$, ..., $\mathbf{p}[\mathrm{d}[s]]$, being in ascending order of the x-coordinate. The algorithm determines the indices of these points using the AVL tree and making use of the fact that a sorting in the x-coordinate implies a reverse order in the y-coordinate (points in the AVL tree are mutually non-dominated in the xy-plane.)

Firstly, the right hand side of $\mathbf{p}[i]$ is considered (region R in Figure 3). The point $\mathbf{p}[r]$ may dominate regions exclusively until level i that from this level onwards are no longer dominated exclusively by $\mathbf{p}[r]$. They can be obtained step-by-step by traversing the box list $\mathrm{L}[r]$ in the direction of ascending x-coordinates (from the head of the list). Each dominated box is completed and the yet undefined z-coordinate is set to $\mathbf{p}[i].z$. The volume of the box is added to the hypervolume contribution $\mathrm{c}[r]$ of $\mathbf{p}[r]$. The algorithm stops removing boxes from the list after the lower bound of a box retrieved from L[r] exceeds $\mathbf{p}[i].x$. The right part of the last box removed may still be exclusively dominated by $\mathbf{p}[r]$. Hence, a new box, (b^r, in Figure 3), may be inserted and attached to the front of the list L[r].

Region M (see Figure 2) comprises the points dominated by $\mathbf{p}$[i] in the xy-plane, namely $\mathbf{p}[\mathrm{d}[1]]$, ..., $\mathbf{p}[\mathrm{d}[s]]$. They are processed in ascending order of x. For each such point, a new box is created and pushed to the back of the list of $\mathbf{p}[i]$. Additionally, a box in region R is created and pushed to the back of the same list with $\mathbf{p}[i]$ as an upper corner (b^+ in Figure 3), if $\mathbf{p}[d].x < \mathbf{p}[i].x$. Its lower y-coordinate is $\mathbf{p}[r].y$. After adding these boxes, the area that is, from now on, exclusively dominated by $\mathbf{p}[i]$ is partitioned into boxes. Moreover, all boxes associated with dominated points (projections) are completed and discarded from the box lists, as they are from now on also dominated by $\mathbf{p}$[i] in the xy-plane. The dominated points themselves are discarded from the tree T for the same reason.

Boxes to the left of $\mathbf{p}$[t] (in region L in Figure 3) are only dominated by $\mathbf{p}[t]$ up to the i-th level, and from level i onwards, additionally dominated partly by $\mathbf{p}[i]$. The boxes in this region need to be updated in their lower y bound. Above $\mathbf{p}[i].y$ they will still be exclusively dominated by $\mathbf{p}$[t]. The update is achieved by completing all boxes with lower y-bound smaller than $\mathbf{p}[i].y$. The corresponding regions exclusively dominated by $\mathbf{p}$[t] can be merged into one single box (b^l in

Figure 3). Therefore only one new box is added to $\mathbf{p}$[t]. This box has $\mathbf{p}[i].y$ as its lower bound in the y-direction, and $\mathbf{p}[t].y$ as its upper bound.

In the final iteration of the algorithm, $i = n + 1$ and all previously processed points are dominated by the auxiliary point $\mathbf{p}[n+1]$. Volumes are updated for all points that still have boxes in their lists. These boxes are lower bounded in the z-coordinate by the reference point. After this operation has finished, the array c[1], ..., c[n] contains the exclusive contributions of the points $\mathbf{p}$[1], ..., $\mathbf{p}[n]$.

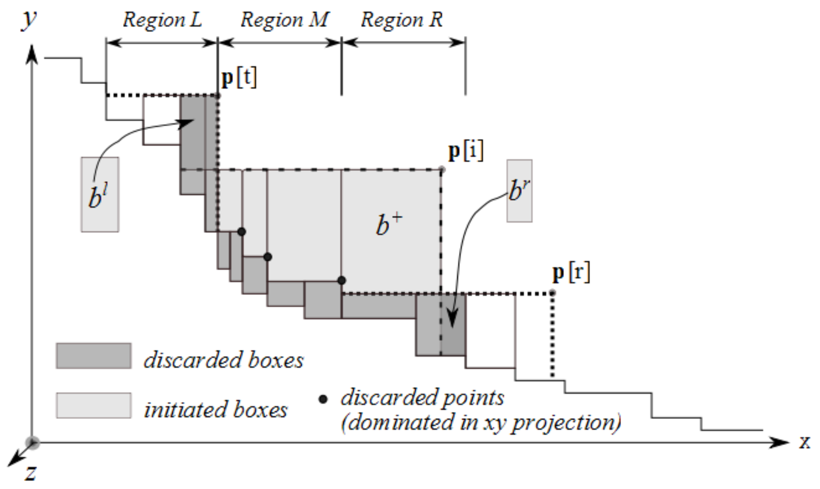

Fig. 3. Single level of the sweeping plain

The runtime of algorithm HyCon3D is estimated by the following theorem:

Theorem 3. *Algorithm* HyCon3D *has a time complexity of* $\mathcal{O}(n \log n)$.

Proof: The time for the initialization phase, including sorting the input, is bounded by $\mathcal{O}(n \log n)$. Time critical operations within the main loop are (1) updating and retrieving points from the AVL tree, and (2) creation and completion of boxes.

The algorithm needs to retrieve all dominated points, and the neighbor point indices r and t from the AVL tree. To identify the neighbor to the right takes time $\mathcal{O}(\log n)$. As the xy-projection is mutually non-dominated the points are sorted also by their y coordinate, and the tree can then be traversed in constant time per additional dominated point. Each point is inserted and discarded only once. The total cost of these operations amortizes to $\mathcal{O}(n \log n)$. Excepting the two boundary points $\mathbf{p}$[r] and $\mathbf{p}$[t], each point that is retrieved from the tree is discarded. Hence, the total cost of tree operations also amortizes to $\mathcal{O}(n \log n)$.

Furthermore, the total number of boxes that are created and completed can be bounded. Each box is created and completed only once. Four different events can cause the creation of a box. Each point $\mathbf{p}$ that is processed in the algorithm causes the creation of (at most) two boxes: One box might be created when the $\mathbf{p}$ is inserted, of which $\mathbf{p}$ is the upper corner. Moreover a box is created when $\mathbf{p}$ gets discarded. It is associated with the point that at that time dominates $\mathbf{p}$ in the xy-plane. In total, at most $2n$ boxes are created this way.

At most two additional boxes may be created per level i, one box for $\mathbf{p}[\mathrm{t}]$ and one box for $\mathbf{p}[\mathrm{r}]$. Accumulating over all n levels, at most $2n$ such boundary boxes are created. Hence, at most $4n$ boxes are created in total.

For all box creations and completions, any relevant point and any box in that point's list of boxes can be located in constant time, because boxes that need to be inserted and deleted are always at the head or at the tail of a doubly linked box list. This results in an overall cost for box list updates of $\mathcal{O}(n)$, and as volumes are only updated when a box is deleted, this is also the complexity of all hypervolume contribution updates. As no other time critical operations are executed, the complexity is $\mathcal{O}(n \log n)$. □

Theorem 3 implies sharper bounds on the complexity of related problems:

Theorem 4. *The following statements hold for $d = 3$ and an input set $\mathrm{Y} \in \mathbb{A}^d$ of size n:*

1. AllContributions *has time complexity $\Theta(n \log n)$.*
2. *The time complexity of* MinimalContribution *is bounded by $\mathcal{O}(n \log n)$.*
3. *The time complexity* MinimalContributor *is bounded by $\mathcal{O}(n \log n)$.*

Proof: (1) Theorem 3 establishes an upper bound for AllContributions of $\mathcal{O}(n \log n)$. This matches the lower bound of $\Omega(n \log n)$ (see theorem 1). (2)+(3): Finding a minimal contribution and contributor can be accomplished in a single scan (linear time) of the results of AllContributions. □

5 Numerical experiments

Experiments are conducted in order to find out the CPU-time needed to compute all hypervolume contributions for input sets of different size $n \in [100, 1000]$. For each value of n, the time needed to process $m = 100$ randomly generated Pareto fronts is measured. Performance is studied on three test problems:

*Problem 6 (*convexSpherical*).* Find all contributions of $\mathrm{Y} \in \mathbb{A}^d$, where $\mathbf{y}$ are generated independently, and $y_i = 10|v_i|/\|\mathbf{v}\|$, $v_i \sim \mathrm{Normal}(0, 1), i = 1, \ldots, d$. (cf. Figure 4, upper left)

*Problem 7 (*concaveSpherical*).* Find all contributions of $\mathrm{Y} \in \mathbb{A}^d$, where $\mathbf{y} \in \mathrm{Y}$ are generated independently, and $y_i = 10 - 10|v_i|/\|\mathbf{v}\|$, $v_i \sim \mathrm{Normal}(0, 1), i = 1, \ldots, d$. (cf. Figure 4, upper middle)

*Problem 8 (*cliff3D*).* Find all contributions for $\mathrm{Y} \subset \mathbb{A}^d$, where $\mathbf{y} \in \mathrm{Y}$ are generated independently, and $y_i = 10|v_i|/\|\mathbf{v}\|$, $v_i \sim \mathrm{Normal}(0, 1)$, $i = 1, 2$, $y_3 \sim \mathrm{Uniform}(0, 10)$. (cf. Figure 4, upper right)

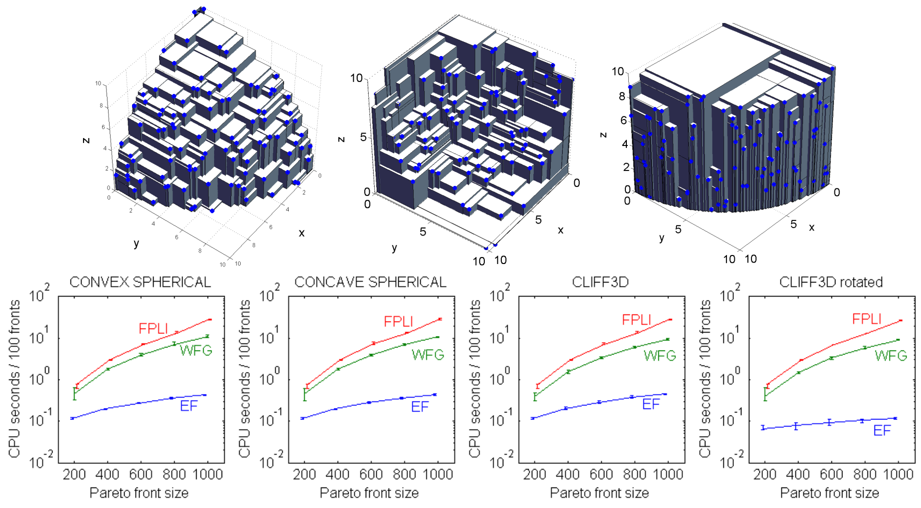

Fig. 4. Randomly generated fronts for the problems convexSpherical, concaveSpherical, and cliff3D with $n = 100$ (above, left to right) and speed tests on these fronts (below). For each type of front, 50 sets were generated. Error bars indicate mean, maximum, and minimum time consumption per algorithm and Pareto-front type and size.

convexSpherical and concaveSpherical consider uniformly distributed points on a convex and, respectively, concave spherical surface. cliff3D considers the third coordinate according to a uniform distribution. It is constructed in a way that, during the dimension sweep, all points remain in the tree until the final layer is reached. Therefore the time for processing this set is supposed to be high. The function `Clock()` is used to measure time in CPU seconds (measured using clock_t and time.h in MinGW/MS Windows). Compiler options are `g++ -O3` on a Toshiba Satellite PRO, Genuine Intel(R) CPU 1.66 GHz, T2300, with 1 GByte RAM. The tested algorithm implementations are:

EF: The dimension sweep algorithm discussed in this paper.[4]

FPLI: iterated application of total hypervolume computation with dimension sweep algorithm by Fonseca, Paquete and López-Ibáñez[5], cf. [12].

WFG: IHSO Algorithm[6] by Walking Fish Group (WFG) e.g. [6].

Figure 4 shows the results on the test-sets. It confirms the considerable speed gain of the new algorithm (EF) as compared to the other algorithms. Results are similar for different shapes of non-dominated fronts (Figure 4, concaveSpherical, convexSpherical, cliff3D). Swapping the $y-$ and $z-$ coordinates of the cliff3D problem (Figure 4, right, below) yields a further speed gain.

[4] C++ source code is available from the authors on request.

[5] http://iridia.ulb.ac.be/∼manuel/hypervolume

[6] (http://www.wfg.csse.uwa.edu.au/toolkit/

6 Conclusion and outlook

The complexity of computing all contributions to the hypervolume given an approximation set $\mathrm{Y} \in \mathbb{A}^d$ has shown to be $\Theta(n \log n)$ when $d = 2$ and $d = 3$, and a lower bound of $\Omega(n \log n)$ has been established for $d > 3$. Also, the problem of computing the hypervolume contribution of a single point has been considered, and has been shown to have complexity $\Theta(n)$ for $d = 2$, $\Theta(n \log n)$ for $d = 3$, and to be bounded by $\Omega(n \log n)$ for $d > 3$. An interesting aspect is that computing a single contribution has the same complexity in the 3-D case as computing all contributions, while in 2-D computing one contribution is less complex than computing all contributions.

A novel dimension sweep algorithm with asymptotically optimal time complexity $\mathcal{O}(n \log n)$ and space complexity $\mathcal{O}(n)$ for computing all hypervolume contributions in 3-D has been introduced. It improves existing algorithms [9] for this problem by a factor of $\sqrt{n}$. Empirical performance tests on randomly generated 3-D non-dominated fronts indicate that the new algorithm is considerably faster than existing algorithm implementations.

It is promising to apply the new algorithm in hypervolume-based archivers (e.g. [16]) and evolutionary algorithms (e.g. [20]), as it will make larger population/archive sizes and a higher number of iterations affordable. Interesting directions for future research could be the extension of the approach to problems in higher dimensions or to more general hypervolume-based subset selection problems, and the analysis of incremental update schemes.

Acknowledgments Michael Emmerich gratefully acknowledges support by Fundação para a Ciência e a Technologia (FCT), Portugal. Grant: Set-Indicator Based Multiobjective Optimization (SIMO), SFRH/BPD/65330/2009, with funds from POPH of QREN Portugal 2007-2013 and the MCTES state budget. The authors would like to acknowledge Andreia Guerreiro and the anonymous reviewers for their insightful feedback on the manuscript.

References

1. Adelson-Velskij, G., Landis, E.: An algorithm for the organization of information. Doklady Akad. Nauk SSSR 146, 263–266 (1962)
2. Bader, J., Zitzler, E.: HypE: An algorithm for fast hypervolume-based many-objective optimization. Evolutionary Computation (2010), (in press)
3. Beume, N.: S-metric calculation by considering dominated hypervolume as Klee's measure problem. Evolutionary Computation 17(4), 477–492 (2009)
4. Beume, N., Fonseca, C.M., López-Ibáñez, M., Paquete, L., Vahrenhold, J.: On the complexity of computing the hypervolume indicator. IEEE Transact. Evolutionary Computation 13(5), 1075–1082 (2009)
5. Bradstreet, L., Barone, L., While, L.: Updating exclusive hypervolume contributions cheaply. In: Conf. on Evolutionary Computation. pp. 538 –544. IEEE Press (2009)
6. Bradstreet, L., While, L., Barone, L.: A fast incremental hypervolume algorithm. IEEE Transact. on Evolutionary Computation 12(6), 714–723 (2008)

7. Bringmann, K., Friedrich, T.: Approximating the volume of unions and intersections of high-dimensional geometric objects. In: Algorithms and Computation, LNCS, vol. 5369, pp. 436–447. Springer (2008)
8. Bringmann, K., Friedrich, T.: Approximating the least hypervolume contributor: NP-hard in general, but fast in practice. In: Evolutionary Multicriterion Optimization. LNCS, vol. 5467, pp. 6–20. Springer (2009)
9. Bringmann, K., Friedrich, T.: An efficient algorithm for computing hypervolume contributions. Evolutionary Computation 18(3), 383–402 (2010)
10. Emmerich, M., Beume, N., Naujoks, B.: An EMO algorithm using the hypervolume measure as selection criterion. In: Evolutionary Multicriterion Optimization. LNCS, vol. 3410, pp. 62–76. Springer (2005)
11. Fleischer, M.: The measure of pareto optima applications to multi-objective metaheuristics. In: Evolutionary Multicriterion Optimization. LNCS, vol. 2632, pp. 519–533. Springer (2003)
12. Fonseca, C.M., Paquete, L., López-Ibáñez, M.: An improved dimension-sweep algorithm for the hypervolume indicator. In: Conf. on Evolutionary Computation. pp. 1157 – 1163. IEEE Press (2006)
13. Huband, S., Hingston, P., While, L., Barone, L.: An evolution strategy with probabilistic mutation for multi-objective optimisation. In: Conf. on Evolutionary Computation. vol. 4, pp. 2284 – 2291. IEEE Press (2003)
14. Igel, C., Hansen, N., Roth, S.: Covariance matrix adaptation for multi-objective optimization. Evolutionary Computation 15(1), 1–28 (2007)
15. Ishibuchi, H., Tsukamoto, N., Sakane, Y., Nojima, Y.: Indicator-based evolutionary algorithm with hypervolume approximation by achievement scalarizing functions. In: GECCO '10. pp. 527–534. ACM, NY, USA (2010)
16. Knowles, J., Corne, D., Fleischer, M.: Bounded archiving using the Lebesgue measure. In: Conf. on Evolutionary Computation. pp. 2490–2497. IEEE Press (2003)
17. Knuth, D.: The Art of Computer Programming Vol.3. Addison-Wesley (1998)
18. Kung, H.T., Luccio, F., Preparata, F.P.: On finding the maxima of a set of vectors. JACM 22(4), 469–476 (1975)
19. Mostaghim, S., Branke, J., Schmeck, H.: Multi-objective particle swarm optimization on computer grids. In: GECCO '07. pp. 869–875. ACM, NY, USA (2007)
20. Naujoks, B., Beume, N., Emmerich, M.: Multi-objective optimisation using S-metric selection: Application to three-dimensional solution spaces. vol. 2 (2005)
21. Overmars, M.H., Yap, C.K.: New upper bounds in klee's measure problem. SIAM J. Comput. 20(6), 1034–1045 (1991)
22. Preparata, F.P., Shamos, M.I.: Computational Geometry. Springer (1985)
23. While, L., Hingston, P., Barone, L., Huband, S.: A faster algorithm for calculating hypervolume. IEEE Transact. Evolutionary Computation 10(1), 29 – 38 (2006)
24. Zitzler, E.: Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. Ph.D. thesis, ETH Zurich, Switzerland (1999)
25. Zitzler, E., Künzli, S.: Indicator-based selection in multiobjective search. In: Parallel Problem Solving from Nature, LNCS, vol. 3242, pp. 832–842. Springer (2004)
26. Zitzler, E., Thiele, L.: Multiobjective optimization using evolutionary algorithms - a comparative case study. In: Parallel Problem Solving from Nature. LNCS, vol. 1498, pp. 292–301. Springer, Amsterdam (1998)
27. Zitzler, E., Thiele, L., Laumanns, M., Foneseca, C.M., Grunert da Fonseca, V.: Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transact. on Evolutionary Computation 7(2), 117–132 (2003)