Computing and Updating Hypervolume Contributions in Up to Four Dimensions

Andreia P. Guerreiro and Carlos M. Fonseca

***Abstract*—Arising in connection with multiobjective selection and archiving, the Hypervolume Subset Selection Problem (HSSP) consists in finding a subset of size $k \leq n$ of a set $\mathrm{X} \subset \mathbb{R}^d$ of n nondominated points in d-dimensional space that maximizes the hypervolume indicator. The incremental greedy approximation to the HSSP has an approximation guarantee of $1-1/e$, and is polynomial in n and k, while no polynomial exact algorithms are known for $d \geq 3$. The decremental greedy counterpart has no known approximation guarantee, but is potentially faster for large k, and still leads to good approximations in practice. The computation and update of individual hypervolume contributions are at the core of the implementation of this greedy strategy.**

In this paper, new algorithms for the computation and update of hypervolume contributions are developed. In three dimensions, updating the total hypervolume and all individual contributions under single-point changes is performed in linear time, while in the four-dimensional case all contributions are computed in $O(n^2)$ time. As a consequence, the decremental greedy approximation to the HSSP can now be obtained in $O(n(n-k)+n\log n)$ and $O(n^2(n-k))$ time for $d=3$ and $d=4$, respectively. Experimental results show that the proposed algorithms significantly outperform existing ones.

***Index Terms*—Hypervolume Indicator, Hypervolume Contributions, HSSP, Incremental Algorithms**

I. Introduction

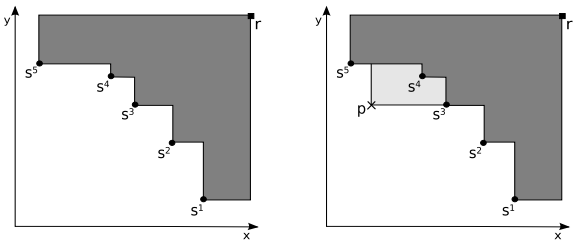
In multiobjective optimization, solutions in a decision space $\mathcal{S}$ are evaluated by means of d objective functions and mapped onto corresponding points in objective space $\mathbb{R}^d$. Formally, a vector function $f : \mathcal{S} \rightarrow \mathbb{R}^d$ where $f(x) = (f_1(x), \ldots, f_d(x))$ is considered. Assuming minimization without loss of generality, the ideal solution would have the lowest possible function value in every objective. Since such a Utopian solution seldom exists, the concept of Pareto-dominance [1] is often used to define what an optimal solution is in this context.

Given two solutions $a, b \in \mathcal{S}$ and their corresponding images in objective space, $u = f(a)$ and $v = f(b)$, solution a is said to weakly dominate solution b in the Pareto sense iff $u_i \leq v_i$ for all $i = 1, \ldots, d$. In this case, vector u is also said to weakly dominate v, which is represented by $u \leq v$. If neither $u \leq v$ or $v \leq u$ are true, solutions a and b are said to be incomparable, and so are u and v. When $u = v$, a and b are said to be indifferent [1].

A solution $a \in \mathcal{S}$ is said to be a *Pareto-optimal*, or *efficient*, solution iff $f(b) \leq f(a) \Rightarrow f(b) = f(a),\ \forall\, b \in \mathcal{S}$. Since weak dominance is generally a partial order on $\mathbb{R}^d$ (and a partial pre-order on $\mathcal{S}$), there may be multiple incomparable Pareto-optimal solutions. The set of all Pareto-optimal solutions of a given problem instance is called the Pareto-optimal set of that instance, while the corresponding set of points in objective space is called the Pareto-optimal front [1], [2].

CISUC, Department of Informatics Engineering, University of Coimbra, Polo II, P-3030 290 Coimbra, Portugal, e-mail: {apg,cmfonsec}@dei.uc.pt.

Manuscript received Month ??, ????; revised month ??, ????.



(a) Hypervolume Indicator (b) Hypervolume Contribution

Fig. 1. Example of the hypervolume indicator of a point set $\mathrm{X} = \{s^1, \ldots, s^5\} \subset \mathbb{R}^2$ and of the contribution of a point $p \in \mathbb{R}^2$ to X.

In practice, the Pareto-optimal front may be a very large or even infinite set, and determining it exactly may therefore be infeasible or, at least, impractical. For that reason, the problem of finding good finite approximations to the Pareto front is usually considered instead. The evaluation of the quality of such approximation sets currently relies mainly on unary quality indicators, such as the *hypervolume indicator* [3], [4], which map a point set to a real value. The hypervolume indicator measures the size of the region that is weakly dominated by any of the points in a set and is bounded above by a reference point, as illustrated in Figure 1 (a). The hypervolume *contribution* of a point to a point set is the size of the region exclusively dominated by that point. Such a region is depicted in light gray in Figure 1 (b).

The hypervolume indicator is used as part of the selection and/or archiving processes in a number of Evolutionary Multiobjective Optimization (EMO) algorithms [3], [5]–[7]. A typical scenario consists in obtaining the next parental population by discarding a number of the weakest individuals in the current population according to some hypervolume-based figure of merit, such as the hypervolume contribution.

Selecting k out of n points in a set $\mathrm{X} \subset \mathbb{R}^d$ so as to maximize the hypervolume indicator is known as the Hypervolume Subset Selection Problem (HSSP) [8]. Although efficient exact algorithms for the HSSP in two dimensions have recently become available [9], [10], solving the HSSP in more than two dimensions still requires time that grows exponentially in $n-k$ [11]. To side-step these difficulties, greedy algorithms to approximate the HSSP have been proposed [12]. The incremental greedy strategy, whereby k points are selected from X one at a time so as to maximize the hypervolume gained at each step, is polynomial in n and k, and guarantees a $(1-1/e)$ approximation factor [13], [14].

The decremental greedy counterpart, whereby $n-k$ points

are discarded one at a time so as to maximize the hypervolume retained at each step, is also polynomial in n and k, but has no known approximation guarantee. It is also potentially faster than the incremental strategy when k is large, and still leads to good approximations in practice [12]. In particular, it is clearly optimal when $k = n - 1$, a case commonly arising in steady-state EMO algorithms such as LAHC [3] and SMS-EMOA [15]. At each generation of these algorithms, a new solution/point is generated and inserted in the population, and the population is subsequently truncated to its original size by removing a point with the least hypervolume contribution (or least hypervolume *contributor*).

The repeated truncation of the population in LAHC and SMS-EMOA and the decremental greedy approach to the approximation of the HSSP both require the identification of a least hypervolume contributor in sets that differ from each other in a single point. Therefore, rather than computing a least contributor for each set in isolation, the similarity between such consecutive point sets should be exploited.

Although computing hypervolume contributions is not strictly required, a least contributor can be easily determined by inspection after computing the contributions of all points in a set. Given an n-point set in $\mathbb{R}^d$, this can be easily achieved by performing $n + 1$ computations of the hypervolume indicator, currently leading to an overall time complexity bound of $O(n^{(1+d/3)} \operatorname{polylog} n)$ at best [16]. Dedicated algorithms for the computation of all contributions in up to 6 dimensions are asymptotically faster, however, with complexities of $\Theta(n \log n)$ for $d = 2, 3$ [17], and $O(n^{d/2} \log n)$ for $d > 3$ [11]. On the other hand, in two dimensions, the contributions that are changed by the removal of a point from, or addition of a point to, a set can be updated in $O(\log n)$ time, provided that the contributions of the points in the original set are known in advance [18], [19]. For $d > 2$, existing algorithms [20], [21] speed up contribution updates by computing the size of the regions that are simultaneously and exclusively dominated by the point being updated and each of the remaining points, but no time-complexity results are reported.

In the following, new algorithms to compute and/or update all hypervolume contributions in three and four dimensions are proposed, matching or improving upon the currently known upper bounds in each case. Alternative algorithms for the computation and update of the hypervolume indicator are presented first, and are then extended to the all-contributions case, leading to the computation of the decremental greedy approximation to the HSSP in $O(n(n - k) + n \log n)$ and $O(n^2(n - k))$ time for $d = 3$ and $d = 4$, respectively. All algorithms are explained and experimentally compared against the alternative algorithms available.

This paper is organized as follows. Section II is dedicated to the state-of-the-art where the most relevant algorithms for this work are described. Section III first describes the new algorithms for $d = 3, 4$ applied only to the computation and update of hypervolume indicator, and are then extended in Section IV to compute contributions. The algorithms proposed in Sections III and IV are experimentally evaluated in Section V. Finally, concluding remarks are drawn in Section VI.

II. Background

A. Notation

In the following, points are represented by lower case letters, and sets are represented by capital letters. Letters x, y, z and w in subscript denote the coordinates of a point in (x, y, z, w)-space. Numbers in superscript are used to enumerate points in a set. Projection onto $(d - 1)$-space by omission of the last coordinate is denoted by an asterisk. For example, given a set $\mathrm{X} \subset \mathbb{R}^3$ and a point $p \in \mathrm{X}$, X^* and p^* denote the projection on the (x, y)-plane of X and p, respectively.

B. Definitions

The hypervolume indicator [3], [4] is formally defined as follows:

Definition 1 (Hypervolume Indicator): Given a point set $\mathrm{X} \subset \mathbb{R}^d$ and a reference point $r \in \mathbb{R}^d$, the hypervolume indicator is:

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

where $[p, r] = \{q \in \mathbb{R}^d \mid p \leq q \wedge q \leq r\}$ and $\lambda(\cdot)$ denotes the Lebesgue measure.

The hypervolume contribution is formally defined based on the Hypervolume Indicator [22]:

Definition 2 (Hypervolume Contribution): The hypervolume contribution of a point $p \in \mathbb{R}^d$ to a set $\mathrm{X} \subset \mathbb{R}^d$ is:

$$H(p, \mathrm{X}) = H(\mathrm{X} \cup \{p\}) - H(\mathrm{X} \setminus \{p\})$$

As pointed out in [22], this definition is consistent with the case where $p \in \mathrm{X}$, and the contribution is the hypervolume lost when p is removed from X, as well as with the case where $p \notin \mathrm{X}$, and the contribution of p is the hypervolume gained when adding p to X. While this is certainly convenient, it does not reflect the fact that the hypervolume gained by "adding" a point p to a set already including it is zero. However, this last situation can be handled easily as a special case by checking whether X includes p before applying the definition.

In some cases, it is useful to consider the region dominated simultaneously and exclusively by two points $p, q \in \mathbb{R}^d$:

Definition 3 (Joint Hypervolume Contribution): The joint contribution of points $p, q \in \mathbb{R}^d$ to $\mathrm{X} \subset \mathbb{R}^d$ is:

$$H(p, q, \mathrm{X}) = H((\mathrm{X} \setminus \{p, q\}) \cup \{p \vee q\}) - H(\mathrm{X} \setminus \{p, q\})$$

where $\vee$ denotes the component-wise maximum.

Moreover, the contribution of a point p to a set X is bounded above by certain points $q \in \mathrm{X}$ that shall be referred to as *delimiters*, and are defined as follows [14]:

Definition 4 (Delimiter): Given a point set $\mathrm{X} \subset \mathbb{R}^d$ and a point $p \in \mathbb{R}^d$, let $\mathrm{J} = \mathsf{nondominated}(\{(p \vee q) \mid q \in \mathrm{X} \setminus \{p\}\})$, where $\mathsf{nondominated}(\mathrm{S}) = \{s \in \mathrm{S} \mid t \leq s \Rightarrow s \leq t, \forall t \in \mathrm{S}\}$ denotes the set of nondominated points in S. Then, $q \in \mathrm{X}$ is called a *(weak) delimiter* of the contribution of p to X iff $(p \vee q) \in \mathrm{J}$. If, in addition, $H(p, q, \mathrm{X}) > 0$, then q is also a *strong delimiter* of the contribution of p to X.

The following extension to the notion of delimiter will also be needed in this work:

Definition 5 (Outer Delimiter): Given a point set $\mathrm{X} \subset \mathbb{R}^d$ and a point $p \in \mathbb{R}^d$, $q \in \mathrm{X}$ is called an *outer delimiter* of the contribution of p to X if it is a delimiter of the contribution of p to $\{s \in \mathrm{X} \mid p \not\leq s\}$. A delimiter, q, of the contribution of p to X is called an *inner delimiter* if it is not an outer delimiter, i.e., if $p \leq q$.

Non-strong delimiters can only exist when X contains points with repeated coordinates. Similarly, in general, outer delimiters may not be actual, or *proper*, delimiters in the sense of Definition 4. In the example of Figure 1 (b), points $s^3, \ldots, s^5$ are the (proper) delimiters of the contribution of p to X, of which s^3 and s^4 are inner delimiters. There are two outer delimiters, s^2 and s^5. Point s^2 is not a proper delimiter of the contribution of p to X because $(p \vee s^3) = s^3 \leq (p \vee s^2)$.

C. Related Problems

Many computational problems related to the Hypervolume Indicator can be found in the literature. The following problems are needed in the context of this work, and are based on a previous set of definitions by Emmerich and Fonseca [17]. Note that the reference point is usually considered to be a constant.

Problem 1. *(*HYPERVOLUME*).* Given an n-point set $\mathrm{X} \subset \mathbb{R}^d$ and a reference point $r \in \mathbb{R}^d$, compute the hypervolume indicator of X, i.e., $H(\mathrm{X})$.

Problem 2. *(*ONECONTRIBUTION*).* Given an n-point set $\mathrm{X} \subset \mathbb{R}^d$, a reference point $r \in \mathbb{R}^d$ and a point $p \in \mathbb{R}^d$, compute the hypervolume contribution of p to X, i.e., $H(p, \mathrm{X})$.

Problem 3. *(*ALLCONTRIBUTIONS*).* Given an n-point set $\mathrm{X} \subset \mathbb{R}^d$ and a reference point $r \in \mathbb{R}^d$, compute the hypervolume contributions of all points $p \in \mathrm{X}$ to X.

Problem 4. *(*LEASTCONTRIBUTOR*).* Given an n-point set $\mathrm{X} \subset \mathbb{R}^d$ and a reference point $r \in \mathbb{R}^d$, find a point $p \in \mathrm{X}$ with minimal hypervolume contribution to X.

Sometimes, the above problems are computed for a sequence of sets that differ from the previous one in a single point, either by adding a point to (Incremental case), or by removing a point from (Decremental case), the previous set.

Problem 5. *(*HYPERVOLUMEUPDATE*).* Given an n-point set $\mathrm{X} \subset \mathbb{R}^d$, the reference point $r \in \mathbb{R}^d$, the value of $H(\mathrm{X})$, and a point $p \in \mathbb{R}^d$, compute:

Incremental: $H(\mathrm{X} \cup \{p\}) = H(\mathrm{X}) + H(p, \mathrm{X})$, where $p \notin \mathrm{X}$.
Decremental: $H(\mathrm{X} \setminus \{p\}) = H(\mathrm{X}) - H(p, \mathrm{X})$, where $p \in \mathrm{X}$.

Problem 6. *(*ALLCONTRIBUTIONSUPDATE*).* Given an n-point set $\mathrm{X} \subset \mathbb{R}^d$, a reference point $r \in \mathbb{R}^d$, the value of $H(q, \mathrm{X})$ for every $q \in \mathrm{X}$, and a point $p \in \mathbb{R}^d$:

Incremental Compute $H(q, \mathrm{X} \cup \{p\}) = H(q, \mathrm{X}) - H(p, q, \mathrm{X})$ for all $q \in \mathrm{X}$, and also $H(p, \mathrm{X})$, where $p \notin \mathrm{X}$.
Decremental Compute $H(q, \mathrm{X} \setminus \{p\}) = H(q, \mathrm{X}) + H(p, q, \mathrm{X})$, where $p \in \mathrm{X}$.

Finally, the HSSP problem [8] is formally defined here as:

Problem 7. *(Hypervolume Subset Selection Problem).* Given an n-point set $\mathrm{X} \subset \mathbb{R}^d$ and an integer $k \in \{0, 1, \ldots, n\}$, find a subset $\mathrm{A} \subseteq \mathrm{X}$, such that $|\mathrm{A}| \leq k$ and $H(\mathrm{A}) \geq H(\mathrm{B})$ for all $\mathrm{B} \subseteq \mathrm{X}, |\mathrm{B}| \leq k$.

Note that X is usually a nondominated point set, even though this is not mandatory. Any dominated point $q \in \mathrm{X}$ will have zero contribution to X. However, if q is dominated by a single point $p \in \mathrm{X}$, then the contribution of p to X will be lower than what it would be if $q \notin \mathrm{X}$. Moreover, the incremental scenarios of Problems 5 and 6 explicitly require that $p \notin \mathrm{X}$ because the adopted definition of hypervolume contribution does not handle adding a point to a set in which it is already included, as discussed before, nor does it consider the multiset that would result from such an operation. If such cases become relevant, the hypervolume contribution of repeated points in a multiset should be considered to be zero.

Problem 1 is the most well-studied problem, and several algorithms to compute the hypervolume indicator have been proposed. In the scope of this paper, the asymptotically optimal, $\Theta(n \log n)$-time algorithm for $d = 3$ by Beume *et al.* [23], here referred to as HV3D, and an $O(n^2)$-time algorithm for $d = 4$ known as HV4D [24], are the most relevant. Together with the WFG [25] and QHV [26] algorithms for $d > 4$, they are considered to be the fastest in practice. However, Chan's algorithm [16] still exhibits the best time-complexity upper bound of $O(n^{d/3} \operatorname{polylog} n)$ for $d \geq 4$. Other important algorithms include [27]–[29]. A recent HV4DX algorithm [30] is claimed to improve upon HV4D, but the experimental results presented in Section V do not support those claims.

It is clear from Definition 2 that any algorithm that computes HYPERVOLUME can also be used to compute ONECONTRIBUTION (Problem 2), and vice-versa. Moreover, HYPERVOLUMEUPDATE (Problem 5) can be solved by computing either HYPERVOLUME given $\mathrm{X} \cup \{p\}$ or ONECONTRIBUTION given X and p. IHSO [31] is an algorithm for ONECONTRIBUTION that is used in IIHSO [32] to compute HYPERVOLUME by solving a sequence of HYPERVOLUMEUPDATE problems as new points are added to a set. Similarly, in HV3D and HV4D, HYPERVOLUME is computed in $d = 3$ and $d = 4$ dimensions, respectively, by iteratively updating the indicator of a $(d-1)$-dimensional projection of a subset of the input set as points are added to it. Such update operations are performed in (amortized) time complexity $O(\log n)$ and $O(n)$ per point, respectively, which is faster than recomputing HYPERVOLUME in $d - 1$ dimensions at each iteration. The same techniques can be used in the decremental case, as well.

A rather different approach to HYPERVOLUME computation has been proposed by Lacour *et al.* [33]. HBDA computes the hypervolume indicator by partitioning the dominated region into $O(n^{\lfloor \frac{d}{2} \rfloor})$ axis-parallel boxes and adding up the corresponding hypervolumes. The incremental version of the algorithm (HBDA-I) runs in $O(n^{\lfloor \frac{d}{2} \rfloor + 1})$ time, and allows input points to be processed in any order. Since the current box decomposition must be stored across iterations, $O(n^{\lfloor \frac{d}{2} \rfloor})$ space is required. By processing input points in ascending order of any given coordinate, the memory requirements are

reduced to $O(n^{\lfloor \frac{d-1}{2} \rfloor})$, and the time complexity is improved to $O(n^{\lfloor \frac{d-1}{2} \rfloor + 1})$. HBDA-NI has been shown to be competitive in $d \geq 4$ dimensions, but its memory requirements are a limiting factor for large d.

It should also be clear that any algorithm that computes ONECONTRIBUTION can be used to compute ALLCONTRIBUTIONS (Problem 3), and vice-versa. Dedicated algorithms for ALLCONTRIBUTIONS include Bringmann and Friedrich's [11] algorithm for $d \geq 2$, with $O(n^{d/2} \log n)$ time complexity, and Emmerich and Fonseca's [17] algorithm for $d = 3$, referred to as EF, which runs in optimal $\Theta(n \log n)$ time. These algorithms can also be used to solve Problem 4 (LEASTCONTRIBUTOR), by computing all contributions and then selecting a point with minimal contribution, although this is not strictly required. Algorithms such as IHSO* [31] and IWFG [34] identify a least contributor while trying to avoid computing all contributions exactly. Stochastic approximation algorithms [35], where a point with at most $1 + \epsilon$ times the minimal contribution is determined with probability $1 - \delta$ for any $\epsilon > 0$ and $\delta > 0$, are another alternative, particularly in high dimensions [36]. Problem 6 (ALLCONTRIBUTIONSUPDATE) can be solved in $O(\log n)$ time when $d = 2$ [18], [19], but otherwise all contributions must be recomputed in the absence of dedicated algorithms.

Finally, and as pointed out earlier in Section I, efficient algorithms for Problem 7 (HSSP) are known only for two dimensions, with time-complexity bounds of $O(nk+n \log n)$ [9] and $O((n-k)k + n \log n)$ [10]. For $d \geq 3$, Bringmann and Friedrich's algorithm [11] takes exponential time in $n-k$, and the problem is claimed to be NP-hard [37]. Incremental greedy approaches provide a $(1 - 1/e)$-approximation to HSSP, i.e. $H(\mathrm{S}')/H(\mathrm{S}^{\mathrm{opt}}) \geq 1-1/e$, where $\mathrm{S}' \subset \mathrm{X}$ and $\mathrm{S}^{\mathrm{opt}} \subset \mathrm{X}$ denote the greedy and the optimal solutions, respectively. For $d = 3$, such an approximation can be computed in $O(nk + n \log n)$ time [14]. Decremental greedy approaches to HSSP have also been proposed [38], and are potentially faster when k is large. Currently, time complexities of $O(n(n-k) \log n)$ and $O(n^2(n-k) \log n)$ can be achieved in 3 and 4 dimensions, respectively, by iterating over the corresponding ALLCONTRIBUTIONS algorithms [11], [17]. However, they offer no approximation guarantee. In particular, for $d \geq 3$, the decremental greedy approximation may be very far from optimal with respect to the hypervolume lost when $1 < n-k \leq d$ points are discarded, i.e., $(H(\mathrm{X}) - H(\mathrm{S}'))/(H(\mathrm{X}) - H(\mathrm{S}^{\mathrm{opt}}))$ may be arbitrarily large [11]. However, as the following example shows, a similar result applies to the incremental case. Consider $\mathrm{X} = \{p, q^1, q^2\}$, where $p = (-\varepsilon - 1, -\varepsilon - 1)$, $q^1 = (-\varepsilon - 2, -\varepsilon)$, and $q^2 = (-\varepsilon, -\varepsilon - 2)$, arbitrarily large $\varepsilon > 1$, the reference point $(0, 0)$, and $k = 2$. Then, $\mathrm{S}^{\mathrm{opt}} = \{q^1, q^2\}$ and $\mathrm{S}' = \{p, q^1\}$ (or $\mathrm{S}' = \{p, q^2\}$), and $(H(\mathrm{X}) - H(\mathrm{S}'))/(H(\mathrm{X}) - H(\mathrm{S}^{\mathrm{opt}})) = \varepsilon$.

In summary, solving any of the above problems more efficiently should allow at least some of the other problems to be solved more efficiently, as well.

D. Dimension-Sweep Algorithms

Dimension sweep is a paradigm which has been widely used in the development of algorithms for hypervolume-related problems. In a dimension-sweep algorithm, a problem involving n points in $\mathbb{R}^d$ is solved by visiting all points in ascending (or descending) order of one of the coordinates, solving a $(d-1)$-dimensional subproblem for each point visited, and combining the solutions of those subproblems. The subproblems themselves can often be solved using dimension sweep as well, until a sufficiently low-dimensional base case that can be solved by a dedicated algorithm is reached. However, the time complexity of the resulting algorithms typically increases by an $O(n)$ factor per dimension, which may or may not be competitive with other approaches.

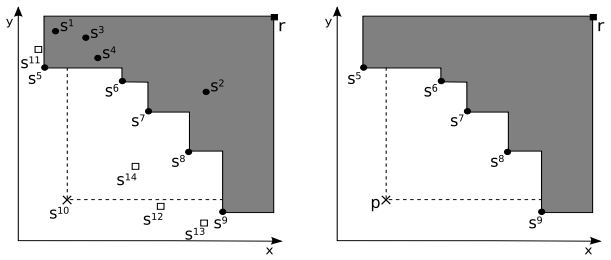


Fig. 2. An example in three-dimensions where $s_z^1 < \cdots < s_z^{14}$ and $p = s^{10}$.

A typical dimension-sweep algorithm for Problem 1 [27], [29] works as follows. Input points are sorted and visited in ascending order of the last coordinate. The d-dimensional dominated region is partitioned into n slices by axis-parallel cut hyperplanes defined by the last coordinate value of each input point and the reference point. The desired hypervolume indicator value is the sum of the hypervolumes of all slices, and the hypervolume of a slice is the hypervolume of its $(d-1)$-dimensional base multiplied by its height. The base of a slice is the $(d-1)$-dimensional region dominated by the projection of the points below it according to dimension d onto the corresponding cut hyperplane. The height of a slice is the difference between the values of the last coordinate of two consecutive points.

1) HYPERVOLUME *in Three Dimensions:* HV3D, by Beume *et al.* [23], is a dimension-sweep algorithm to compute HYPERVOLUME in three dimensions that operates by solving a sequence of two-dimensional HYPERVOLUMEUPDATE problems. Given an n-point set $\mathrm{X} \subset \mathbb{R}^3$, points in X are sorted and visited in ascending z-coordinate order. Each point $p \in \mathrm{X}$ marks the beginning of a new slice, the base area of which is computed by updating the area of the base of the previous slice (if it exists). This is illustrated in Figure 2 (a), where the shaded region represents the base of the previous slice to be updated when $p = s^{10}$ is visited. To that end, the points visited so far whose projections on the (x, y)-plane are mutually nondominated, depicted in Figure 2 (b), are kept sorted in ascending order of the y coordinate using a height-balanced binary tree, T. For each $p \in \mathrm{X}$, the point $q \in \mathrm{T}$ with the least $q_x > p_x$ such that $q_y < p_y$ (s^9 in the example) is determined in $O(\log n)$ steps by searching T. Then, the contribution of p^* to T^* is computed by visiting the successors of q in T in ascending order of y until a point is found whose projection is not dominated by p^* (in the example, s^5).

Let the set of points in T that are weakly dominated by p

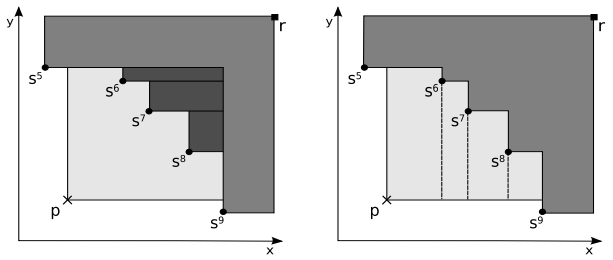


Fig. 3. Computing the contribution of p^* to T^*.

on the (x, y)-plane be denoted by $\mathrm{Q} = \{q \in \mathrm{T} \mid p^* \leq q^*\}$. The contribution of p^* to T^* can be computed in two different ways. In HV3D, for each point $q \in \mathrm{Q} \subseteq \mathrm{T}$, the contribution of q^* to T^* is computed and subtracted from the area of the base of the previous slice in constant time, and q is removed from T in $O(\log n)$ time (see Figure 3 (a)). Then, the area of the region exclusively dominated by p^*, $(s^9_x - p_x) \times (s^5_y - p_y)$, is determined and added to the current base area. Alternatively, the contribution of p^* can be computed by partitioning it into boxes while iterating over Q and adding up the corresponding areas (see Figure 3 (b)). At each iteration, the current point in Q is removed from T, as before. In both cases, p is added to T in $O(\log n)$ time once the base area has been updated.

In the above, each point in X is visited twice: once when it is added to T and again when it is removed from T. Since all of the corresponding operations are performed in $O(\log n)$ time, the algorithm has $O(n \log n)$ time complexity.

2) ALLCONTRIBUTIONS *in Three Dimensions:* The EF algorithm [17] extends HV3D to the computation of the contributions of all points in a nondominated set $\mathrm{X} \subset \mathbb{R}^3$. For each point $p \in \mathrm{X}$, a box partition of the contribution of p^* to T^*, as previously seen in Figure 3 (b), is stored in a doubly-linked list, and the box partitions of the corresponding delimiters in T^* are updated. Individual boxes are characterized by their lower and upper corners in all three dimensions, although they are initially unbounded above in the z dimension.

Boxes are stored until another point in X that (partially) dominates them on the (x, y)-plane is visited. In particular, for each $p \in \mathrm{X}$, all boxes associated to the inner delimiters of p^* are closed (in the example, these would be all boxes associated with points s^6, s^7 and s^8). Closing a box means setting the z coordinate of its top corner to p_z, computing its volume and adding it to the current value of the contribution of the associated point, and discarding that box. Boxes partially dominated by p^* are also closed, and are replaced by a new box accounting for the base area that remains not dominated by p^*. In the example, this would be the case of some of the boxes associated with s^5 or s^9.

All box operations are performed in $O(1)$ time, and since at most $O(n)$ boxes are created and closed, the algorithm retains the $O(n \log n)$ time complexity of HV3D.

3) HYPERVOLUME *in Four Dimensions:* HV4D [24] is an extension of HV3D to four dimensions where a sequence of three-dimensional HYPERVOLUMEUPDATE problems is solved via the corresponding ONECONTRIBUTION problems using similar techniques to those in EF. Points in the input set $\mathrm{X} \in \mathbb{R}^4$ are visited in ascending order of the last coordinate,

Algorithm 1 genericSweep$_y(p, \mathrm{S}, s)$

Require: $s = \mathsf{outerDelimiter}_x(p, \mathrm{S})$
1: $e \leftarrow \mathsf{next}_y(s, \mathrm{S})$
2: procedureA(S, p, s, e)
3: **while** $p_x \leq e_x$ **do**
4: $\quad q \leftarrow e$
5: $\quad e \leftarrow \mathsf{next}_y(e)$
6: $\quad$ procedureB(S, p, q, e)
7: procedureC(S, p, q, e)
8: **return** *info*

partitioning the dominated region into four-dimensional slices. For each $p \in \mathrm{X}$, the base volume of the new slice is computed by updating the volume of the base of the previous slice with the contribution of p^* to the projection on (x, y, z)-space of the points visited so far.

For that purpose, the points visited so far whose projections are nondominated are stored in a data structure, L, consisting of two doubly-linked lists sorted in ascending order of the y and z coordinates, respectively. Base-volume updates are performed in two steps. First, for each $q \in \mathrm{L}$ such that $p^* \leq q^*$, the contribution of q^* to L^* is computed and subtracted from the current base volume, and q is removed from L. Then, the contribution of p^* to L^* is determined and added to the current base volume, and p is added to L.

Under the above conditions, the contribution of p^* to L^* is computed in linear time. As in EF, the two-dimensional base of the contribution is partitioned into boxes by sweeping the points in L in ascending order of y. Then, all points $q \in \mathrm{L}$ such that $q_z > p_z$ are visited in ascending order of z, and for each of these points, the boxes that are (partially) dominated by q^* are updated. Since a three-dimensional contribution is computed at most twice for each input point, once when it is added to L and once in case it is removed from L, the time complexity of HV4D amortizes to $O(n^2)$.

III. HYPERVOLUME INDICATOR

In this section, HV3D [23] is modified in order to allow for incremental and decremental updates in linear time, resulting in a new algorithm that will be called HV3D$^+$. This is achieved by preprocessing the input points and setting up a data structure to support the subsequent hypervolume calculation. Hypervolume updates are performed by updating the data structure to reflect the insertion or the removal of a point and either recomputing the new hypervolume as a whole or computing the corresponding contribution. By iterating over such updates in three dimensions, a new $O(n^2)$-time algorithm for four dimensions is obtained as an alternative to HV4D [24].

A. Three Dimensions

1) Data Structures: Maintaining the set of points visited so far whose projections on the (x, y)-plane are nondominated and being able to access them in ascending order of the y coordinate are key aspects of both HV3D and HV3D$^+$. Let S represent a data structure for that purpose, which can be either a balanced binary tree, or a linked list. Note that, since

S^* contains nondominated points only, ascending order of coordinate y is equivalent to descending order of coordinate x.

Consider the following operations on S, as well as the corresponding operations obtained by switching the roles of the x and y coordinates. It is assumed that $s \in \mathrm{S}$, $p \in \mathbb{R}^3$, and $q <_L p$ for all $q \in \mathrm{S}$, where $q <_L p$ denotes that q is lexicographic less than p in dimensions z, y, x. Thus, $q_z \leq p_z$.

head$_y(\mathrm{S})$ Return the point $q \in \mathrm{S}$ with the smallest q_y.

next$_y(s, \mathrm{S})$ Return $q \in \mathrm{S}$ with the smallest $q_y > s_y$.

remove(s, S) Remove s from S.

outerDelimiter$_y(p, \mathrm{S})$ Return the point $q \in \mathrm{S}$ with the smallest $q_y > p_y$ such that $q_x < p_x$.

removeDominated$_y(p, \mathrm{S}, s)$ If $s =$ outerDelimiter$_x(p, \mathrm{S})$, remove all points $q \in \mathrm{S}$ such that $p^* \leq q^*$ from S, and return them sorted in ascending order of q_y.

computeArea$_y(p, \mathrm{S}, s)$ If $s =$ outerDelimiter$_x(p, \mathrm{S})$, compute and return the area of the region exclusively dominated by p^* with respect to S^*.

add$_y(p, \mathrm{S}, s)$ Insert p into S if $s_y < p_y <$ next$_y(s, \mathrm{S})$ or $p_y <$ head$_y(\mathrm{S})$. In the latter case, s should be NULL.

Operation outerDelimiter requires time in $O(\log n)$ if S is a binary tree and in $O(n)$ if it is a linked list. Operations head, next, add and remove take $O(1)$ time on a linked list (because $s \in \mathrm{S}$) and, in general, $O(\log n)$ time on a balanced binary tree. On a tree, head can also be implemented in constant time just by caching a pointer to the head node.

Operations removeDominated and computeArea both follow the template presented in Algorithm 1. Points $q \in \mathrm{S}$ whose projections are dominated by p^* (inner delimiters) are visited in ascending order of q_y by starting at $s =$ outerDelimiter$_x(p, \mathrm{S})$, which must be passed as an input argument, and stopping at the first subsequent point e such that $p^* \not\leq e^*$. Routines procedureA, procedureB and procedureC, respectively, represent the pre-processing, processing and post-processing operations associated with the sequence of points visited. In operation removeDominated, an empty point list is initialized in procedureA. In procedureB, the visited points q are added to that list, and are removed from S by invoking remove(q, S), whereas procedureC does nothing. The list is returned as *info*. In operation computeArea, the area of the rectangle $[(p_x, p_y), (s_x, e_y)]$ is computed, and is stored in *info* in procedureA. Similarly, the area of $[(p_x, q_y), (q_x, e_y)]$ is computed and added to *info* in procedureB. As before, procedureC does nothing, but this routine will become important later, in Section IV.

Returning to the example of Figure 2 (b), given p, $\mathrm{S} = \{s^5, \ldots, s^9\}$, and $s = s^9$, the delimiters of p^* are visited starting at s^9 and stopping at s^5. Operation removeDominated removes points s^8, s^7 and s^6 from S, and returns them in this order in a list, whereas computeArea returns the area exclusively dominated by p^*, and leaves S unmodified. Since the processing of each point is dominated by the complexity of add, next and remove, the resulting time complexity upper bounds are $O(t)$ on lists and $O(t \log |\mathrm{S}|)$ on binary trees, where t denotes the number of inner delimiters of p^*.

2) Preprocessing: Algorithm 2 reproduces the sequence of binary-tree operations performed in HV3D. Input points are stored and visited in ascending lexicographic order of

Algorithm 2 HV3D$^+$ – Preprocessing

Require: $\mathrm{X} \subset \mathbb{R}^3$ // a set of n nondominated points
Require: $r \in \mathbb{R}^3$ // the reference point
1: $\mathrm{Q} \leftarrow \mathrm{X}$ // linked list sorted in *ascending* lexicographic order of coordinates z, y, x
2: $\mathrm{T} \leftarrow \{(r_x, -\infty, -\infty), (-\infty, r_y, -\infty)\}$ // binary tree sorted in *ascending* order of dimension y
3: **for each** $p \in \mathrm{Q}$ **do**
4: $\quad s \leftarrow$ outerDelimiter$_x(p, \mathrm{T})$
5: $\quad$ removeDominated$_y(p, \mathrm{T}, s)$
6: $\quad p.cx \leftarrow s$
7: $\quad p.cy \leftarrow$ next$_y(s, \mathrm{T})$ // same as outerDelimiter$_y(p, \mathrm{T})$
8: $\quad$ add$_y(p, \mathrm{T}, s)$
9: **return** Q

coordinates z, y and x to ensure data-structure consistency and well-defined operations in the presence of repeated z coordinates. For each $p \in \mathrm{Q}$, its rightmost outer delimiter, s, is looked up in binary tree T (line 4). Then, the points in T whose projections are dominated by p^* are removed (line 5), and p is added to T (line 8). Pointers to the outer delimiters of p^* in T^* are saved as attributes of p ($p.cx$ and $p.cy$ in lines 6 and 7) for future use. Sentinel nodes in T guarantee that such outer delimiters always exist.

As in HV3D, each input point is visited at most twice, once when it is added to T, and again if it has to be removed from T, at a cost of $O(\log n)$ time in both cases. Determining and saving the outer delimiters of each point (lines 4, 6 and 7) also takes $O(\log n)$ time per input point. Therefore, this preprocessing is performed in $O(n \log n)$ time.

Note that, although the input set X is required to be a nondominated point set, dominated points in Q can be easily detected and discarded at no extra cost by checking immediately after line 4 whether next$_y(s, \mathrm{S})$ dominates p, and skipping to the next input point if it does.

3) Hypervolume Computation: Algorithm 3 can be seen as a reimplementation of HV3D using a linked list, L, instead of a binary tree, T, and follows the same structure as Algorithm 2. However, the outer delimiters of each input point are now required to be known in advance as a result of preprocessing. This allows all next, add and remove operations to be implemented in constant time, as explained in Subsection III-A1.

The hypervolume indicator is computed in a similar way to HV3D. Variables $area$ and vol are used to store, respectively, the area dominated by the points in L^* and the volume of the region dominated by the points visited so far up to the current point, p. The volume is accumulated in vol at the beginning of the loop by multiplying the current (base) area by the height of the current slice (line 4). The area dominated by the points in L^* is updated by adding the contribution $H(p^*, \mathrm{L}^*)$ to the current area (line 6). Finally, the total volume is updated with the volume of the last slice (line 10), and returned.

Since all inner delimiters visited in computeArea are immediately removed in removeDominated, and the complexity of both of these functions is now linear in the number of such delimiters, the complexity of Algorithm 3 amortizes to $O(n)$.

Algorithm 3 HV3D$^+$ – HYPERVOLUME computation

Require: $\mathrm{X} \subset \mathbb{R}^3$ // a set of n nondominated points
Require: $r \in \mathbb{R}^3$ // the reference point
Require: Q // a linked list containing X sorted in *ascending* order of dimension z with $p.cx$ and $p.cy$ set for all $p \in \mathrm{Q}$
1: $\mathrm{L} \leftarrow \{(r_x, -\infty, -\infty), (-\infty, r_y, -\infty)\}$ // linked list sorted in *ascending* order of dimension y
2: $vol, area, z \leftarrow 0$
3: **for each** $p \in \mathrm{Q}$ **do**
4: $\quad vol \leftarrow vol + area \cdot (p_z - z)$
5: $\quad s \leftarrow p.cx$
6: $\quad area \leftarrow area + \mathsf{computeArea}_y(p, \mathrm{L}, s)$
7: $\quad \mathsf{removeDominated}_y(p, \mathrm{L}, s)$
8: $\quad \mathsf{add}_y(p, \mathrm{L}, s)$
9: $\quad z \leftarrow p_z$
10: $vol \leftarrow vol + area \cdot (r_z - z)$
11: **return** vol // $H(\mathrm{X})$

4) Data Structure Updates: Adding a new point u to the data structure maintained by HV3D$^+$ requires setting attributes $u.cx$ and $u.cy$, updating the corresponding attributes of the remaining points in the lexicographically sorted list Q, and inserting u into Q. These operations are performed in linear time in a single sweep of Q, as follows (cy attributes are updated in a similar way, but with the roles of the x and y coordinates switched):

- Set $u.cx$ to the point $q \in \mathrm{Q}$ with the smallest $q_x > u_x$ such that $q_y < u_y$ and $q <_L u$. If such a point is not unique, the alternative with the smallest q_y is preferred.
- For $q \in \mathrm{Q}$, set $q.cx$ to u iff $u_y < q_y$ and $u <_L q$, and either $q_x < u_x < (q.cx)_x$ or $u_x = (q.cx)_x$ and $u_y \leq (q.cx)_y$.
- Insert u into Q immediately before the point $q \in Q$ with the lexicographically smallest q such that $u <_L q$.

As an example, let s^{10} in Figure 2 (a) be the new point u to be inserted, and Q contain all of the remaining points. Then, $s^{10}.cx$ is set to s^9 and $s^{10}.cy$ is set to s^5. Also, $s^{12}.cy$, which is s^7 before s^{10} is inserted, is set to s^{10}.

Although one may require that $\mathrm{Q} \cup \{u\}$ be a nondominated point set, handling dominated points arising from the insertion of u is simple. If u is dominated by points in Q, which can be checked in constant time per point while sweeping Q, then u is simply discarded. If some points in Q are dominated by u, they will not be referenced as delimiters (cx or cy) of any nondominated point in $\mathrm{Q} \cup \{u\}$. This is because any references to points dominated by u will either be made by other points dominated by u or have been updated to refer to u itself. Such dominated points can either be simply removed from Q or be marked as such and remain in Q as it will be the case in Section IV.

Removing a point $u \in \mathrm{Q}$ also requires updating the cx and cy attributes of the remaining points, as follows:

- For every $p \in \mathrm{Q} \setminus \{u\}$ such that $p.cx = u$, set $p.cx$ to the point $q \in \mathrm{Q} \setminus \{u\}$ with the smallest $q_x > p_x$ such that $q_y < p_y$ and $q <_L p$ (and analogously for $p.cy$). If such a point is not unique, the alternative with the smallest q_y (respectively, q_x) is preferred.

Assuming that Q does not contain any dominated points, this is also achieved in linear time by performing essentially the same sequence of operations in Algorithm 2, but using a linked list, L, instead of binary tree, T, and replacing the call to $\mathsf{outerDelimiter}_x$ by $p.cx$. In addition, if $p.cx = u$, variable s should be set to $p.cy$ instead, and the roles of dimensions x and y should be reversed in lines 5 to 8, for that iteration.

In the example of Figure 2 (a), let s^{10} be the point u to be removed from Q, and $p = s^{12}$ be the current point. Hence, $s^{12}.cx = s^9$, $s^{12}.cy = s^{10}$, and $\mathrm{L} = \{s^{11}, s^5, s^6, s^7, s^8, s^9\}$. Then, the points in L whose projections p^* dominates (s_8) are removed from L, and $s^{12}.cy$ is set to $\mathsf{next}_y(s^{12}.cx, \mathrm{L}) = s^7$.

5) Hypervolume Updates: Having established how to update the HV3D$^+$ data structure in linear time, it is clear that HYPERVOLUMEUPDATE can also be computed in linear time by using Algorithm 3 to recompute HYPERVOLUME after updating the data structure. Alternatively, the value of the hypervolume indicator can be updated with the contribution of the point u to be added to, or removed from, Q. Although the contribution of u can be computed while the data structure is being updated in both cases, for simplicity only the case where the point has not yet been added to or has already been removed from Q is explained here.

The ONECONTRIBUTION computation begins with the construction of a list containing all points in $\{p \in \mathrm{Q} \mid p_z \leq u_z\}$ whose projections are inner or outer delimiters of the contribution of u^* to the projection of that set. This list, L, can be set up in linear time as in Algorithm 3, by sweeping Q while $p_z \leq u_z$. Having constructed L, the contribution of u^* to L^* is computed and stored in a variable, $area$, and points $p \in \mathrm{Q}$ such that $p_z > u_z$ are visited in ascending order of p_z until a point such that $p^* \leq u^*$ is found.

For each visited point, p, the contribution of u is updated by accumulating the product of $area$ by the height of the current slice in another variable, vol, and $area$ is updated by subtracting the joint contribution $H(p^*, u^*, \mathrm{L}^*)$ from it. $H(p^*, u^*, \mathrm{L}^*)$ is computed by calling either $\mathsf{computeArea}_y(p \vee u, \mathrm{L}, p.cx)$ or $\mathsf{computeArea}_x(p \vee u, \mathrm{L}, p.cy)$, depending on the relative position of p with respect to u. Then, all points that are no longer outer or inner delimiters of the contribution of u^* on the plane $z = p_z$ are removed from L, and p is added to L if it made $area$ decrease. When u does not dominate p, computing the joint contribution of p^* and u^* and removing the required points from L always entails starting at one end of L, which requires linear time in the number of points removed. On the other hand, when $u \leq p$, such a linear time operation is made possible by the availability of attributes $p.cx$ and $p.cy$, as before. Therefore, the time complexity of the whole ONECONTRIBUTION computation amortizes to $O(n)$.

The procedures used to compute a three-dimensional contribution in HV3D$^+$ and in HV4D are rather alike, and although the latter is tightly integrated in the main algorithm, it also considers both the incremental and the decremental scenarios, and could easily be made available standalone. The main differences lie in the data structures used (a list of points versus a list of boxes) and, more crucially, in how dominated points are handled. In HV4D, computing the three-dimensional

contribution of a point u to the current set of points requires removing any points dominated by u from the current set first and computing their contributions, as well. Although this is done in linear time per point when there are no dominated points, and amortizes to linear time per point over a complete four-dimensional HYPERVOLUME computation, the complexity of computing a single contribution in HV4D is not linear in general, whereas it is always linear in HV3D$^+$.

B. Four Dimensions

The hypervolume indicator in four dimensions can be computed in $O(n^2)$ time by performing a sequence of HV3D$^+$ updates, leading to a new algorithm, HV4D$^+$. As in HV4D, points in $\mathrm{X} \subset \mathbb{R}^4$ are visited in ascending w-coordinate order, dividing the dominated region into n slices. For each visited point, p, the volume of the three-dimensional base of the current slice is computed by adding the contribution of p^* to the volume of the previous slice. Because p^* may dominate points in X^* visited previously, handling dominated points is a must. Not having to remove points whose projections are dominated is an important feature in extending the approach to the computation of all contributions in four dimensions (see Section IV-B).

IV. HYPERVOLUME CONTRIBUTIONS

HV3D$^+$ lends itself to further extension to the computation and update of all hypervolume contributions. A new ALLCONTRIBUTIONS algorithm for three dimensions, named HVC3D, supports contribution updates in linear time, and is proposed next. Two update scenarios are considered:

Nondominated sets Any dominated points are ignored and/or removed as described in subsections III-A2 and III-A4, and do not affect the value of the individual contributions of nondominated points. This is sufficient to implement SMS-EMOA, for example.

Dominated points A new point may dominate existing ones (incremental scenario), in which case its individual contribution is also delimited by the points it dominates. This allows a new $O(n^2)$ algorithm to be constructed for ALLCONTRIBUTIONS in four dimensions.

More general scenarios involving dominated points can in principle be addressed using similar techniques, but their relevance is not clear at present.

A. Three Dimensions

1) Data Structures: To support the computation of individual contributions, the HV3D$^+$ data structure is extended with additional point attributes. For each point $p \in \mathrm{Q}$, the area of the current base of its contribution is stored in $p.a$, the current volume of this contribution is stored in $p.v$, and the z-coordinate value up to which the volume $p.v$ has been computed is stored in $p.z$. Moreover, to support the handling of dominated points, the number of points that dominate p and a pointer to one of those points are stored in $p.nd$ and $p.dom$, respectively. Since the initial set of points, X, is required to contain only nondominated points, $p.nd$ is initialized to zero and $p.dom$ is set to NULL. Whenever a point that dominates p is added to Q, $p.nd$ is incremented, and $p.dom$ is set to that point. As discussed later in Subsection IV-A2, points which become dominated by more than one point will be discarded.

In HVC3D, the set of points visited so far whose projections on the (x, y)-plane are nondominated is maintained in a doubly-linked list, L, sorted in ascending y-coordinate order (and, therefore, also in descending order of the x coordinate). This list plays exactly the same role as L in HV3D$^+$. Moreover, each point $q \in \mathrm{L}$ maintains in $q.\mathrm{L}$ a list of the points visited so far whose projections are outer or inner delimiters of the current exclusive contribution of q^*. In the example of Figure 4 (a), before p is processed, L contains $s^5, \ldots, s^9$. List $s^5.\mathrm{L}$ contains points s^6, s^4, s^3, s^1, and a sentinel, in this order.

Finally, a new operation based on Algorithm 1 is defined:

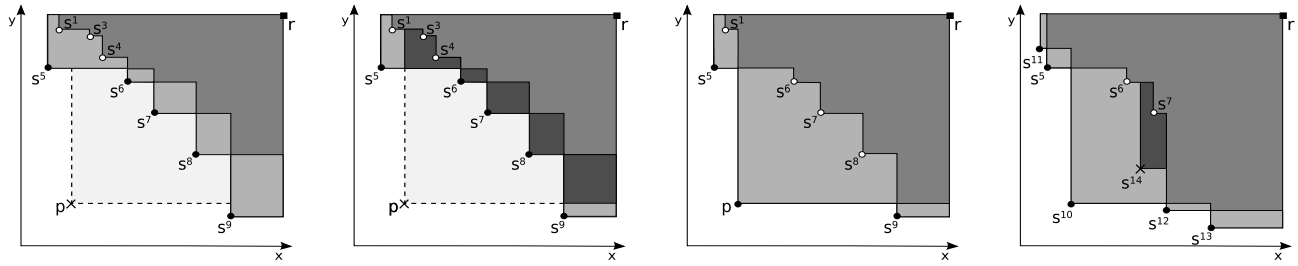
updateVolumes$_y(p, \mathrm{S}, s)$ If $s = \mathsf{outerDelimiter}_x(p, \mathrm{S})$, for each point $q \in \mathrm{S}$ whose projection q^* is an outer or inner delimiter of the contribution of p^* to S^*, add the volume of the current contribution slice, $q.a \cdot (p_z - q.z)$, to $q.v$, and set $q.z = p_z$.

In this case, procedureA, procedureB and procedureC in Algorithm 1 all perform the same operations, but on different points, respectively $s = \mathsf{outerDelimiter}_x(p, \mathrm{S})$, all $q \in \mathrm{S}$ such that $p^* \leq q^*$, and $e = \mathsf{outerDelimiter}_y(p, \mathrm{S})$. Just like the other operations based on Algorithm 1, updateVolumes requires $O(t)$ time, where t is the number of points dominated by p^* in S^*. In the example of Figure 4 (a), given p, $\mathrm{L} = \{s_5, \ldots, s_9\}$ and $s = s_9$, $\mathsf{updateVolumes}(p, \mathrm{L}, s)$ updates the values of attributes v and z of points $s_5, \ldots, s_9$.

2) Computing All Contributions: As an extension of HV3D$^+$ to the computation of ALLCONTRIBUTIONS, HVC3D consists of an $O(n \log n)$-time preprocessing step, identical to Algorithm 2, followed by an actual computation step, which is detailed in Algorithm 4. Points $p \in \mathrm{Q}$ are visited in ascending z-coordinate order, as in Algorithm 3. Considering that all points are nondominated (first scenario), each point is processed in lines 6 to 21 of Algorithm 4. In the example of Figure 4 (a), the regions exclusively dominated by the projections of the points in $\mathrm{L} = \{s^5, \ldots, s^9\}$ before p is processed are depicted in medium gray.

The processing of nondominated points is divided into three main parts. In the first part (lines 6 to 11), the volumes associated with the outer and inner delimiters of p^* in L^* are updated, and the base area of the contribution of p, depicted in light gray in Figure 4 (a), is computed. Then, the points whose projections are dominated by p^* (s^8, s^7, s^6) are moved from L to $p.\mathrm{L}$, as their contribution is zero above the plane $z = p_z$, and copies of $p.cx$ and $p.cy$ (s^9 and s^5), corresponding to the outer delimiters of p^*, are added at each end of $p.\mathrm{L}$, so that the contribution of p can be updated efficiently in subsequent iterations. Finally, p is inserted into L.

In the second part (lines 12 to 16), the base area of the contribution of point $q = p.cy$ is updated, as p^* dominates part of the region dominated by q^*. To that end, the joint contribution of q^* and p^* to $q.\mathrm{L}^*$ is subtracted from $q.a$, the points whose projections are dominated by p^* are removed from $q.\mathrm{L}$, and the head of this list is replaced by p. Now referring to Figure 4 (b), $q = s^5$ and $q.\mathrm{L}$ contains $\{s^6, s^4, s^3, s^1\}$ plus the



(a) Before adding p (b) Reduce areas dominated by p^* (c) After adding p (d) Dominated point s^{14}

Fig. 4. Example of the contribution of each point, before adding $p = s^{10}$, when p reduces contributions, after adding p, and when adding a point it dominates.

Algorithm 4 HVC3D – ALLCONTRIBUTIONS Computation

Require: $\mathrm{X} \subset \mathbb{R}^3$ // a set of n points
Require: $r \in \mathbb{R}^3$ // the reference point
Require: Q // a linked list sorted in *ascending* order of dimension z containing $\mathrm{X} \cup \{(-\mathrm{REALMAX}, -\mathrm{REALMAX}, r_z)\}$ with $p.cx$, $p.cy$, $p.nd$ and $p.dom$ set for all $p \in \mathrm{Q}$

1: $\mathrm{L} \leftarrow \{(r_x, -\infty, -\infty), (-\infty, r_y, -\infty)\}$ // linked list sorted in *ascending* order of dimension y
2: **for each** $p \in \mathrm{Q}$ **do**
3: $p.v \leftarrow 0$
4: $p.z \leftarrow p_z$
5: **if** $p.nd = 0$ **then**
6: $\mathsf{updateVolumes}_y(p, \mathrm{L}, p.cx)$
7: $p.a \leftarrow \mathsf{computeArea}_y(p, \mathrm{L}, p.cx)$
8: $p.\mathrm{L} \leftarrow \mathsf{removeDominated}_y(p, \mathrm{L}, p.cx)$
9: $\mathsf{add}_y(p.cx, p.\mathrm{L}, \mathrm{NULL})$
10: $\mathsf{add}_x(p.cy, p.\mathrm{L}, \mathrm{NULL})$
11: $\mathsf{add}_y(p, \mathrm{L}, p.cx)$
12: $q \leftarrow p.cy$
13: $q.a \leftarrow q.a - \mathsf{computeArea}_y(p \vee q, q.\mathrm{L}, \mathsf{head}_y(q.\mathrm{L}))$
14: $\mathsf{removeDominated}_y(p \vee q, q.\mathrm{L}, \mathsf{head}_y(q.\mathrm{L}))$
15: $\mathsf{remove}(\mathsf{head}_y(q.\mathrm{L}), q.\mathrm{L})$
16: $\mathsf{add}_y(p, q.\mathrm{L}, \mathrm{NULL})$
17: $q \leftarrow p.cx$
18: $q.a \leftarrow q.a - \mathsf{computeArea}_x(p \vee q, q.\mathrm{L}, \mathsf{head}_x(q.\mathrm{L}))$
19: $\mathsf{removeDominated}_x(p \vee q, q.\mathrm{L}, \mathsf{head}_x(q.\mathrm{L}))$
20: $\mathsf{remove}(\mathsf{head}_x(q.\mathrm{L}), q.\mathrm{L})$
21: $\mathsf{add}_x(p, q.\mathrm{L}, \mathrm{NULL})$
22: **if** $p.nd = 1$ **then**
23: $q \leftarrow p.dom$
24: $q.v \leftarrow q.v + q.a \cdot (p_z - q.z)$
25: $q.z \leftarrow p_z$
26: $q.a \leftarrow q.a - \mathsf{computeArea}_y(p, q.\mathrm{L}, p.cx)$
27: $\mathsf{removeDominated}_y(p, q.\mathrm{L}, p.cx)$
28: $\mathsf{add}_y(p, q.\mathrm{L}, p.cx)$

sentinel. The joint contribution of p^* and q^* to $q.\mathrm{L}^*$ at $z = p_z$ is the area of the darker region dominated by both points, and its computation (line 13) involves visiting the points from s^6 to s^1. Then, s^4 and s^3 are discarded (line 14) and s^6 is replaced by p at the head of $q.\mathrm{L}$ (lines 15 and 16). At this point, $q.\mathrm{L}$ contains the outer and inner delimiters of the exclusive contribution of q^* at $z = p_z$, namely points p, s^1 and the sentinel, as Figure 4 (c) illustrates.

The base area of the contribution of $p.cx$ is updated analogously in the third part (lines 17 to 21). The last point in Q to be visited is the sentinel $(-\mathrm{REALMAX}, -\mathrm{REALMAX}, r_z)$, which forces the computation of the contributions of all input points to complete and the results for all points $p \in \mathrm{X} \subset \mathrm{Q}$ to be made available in $p.v$ at the end of the run. Here, $-$REALMAX denotes a finite value less than any coordinate of any input point.

In the second scenario, Q may contain dominated points due to incremental updates to the data structure. Such updates are performed exactly as described in Subsection III-A4 with respect to point attributes cx and cy, but any points $p \in \mathrm{Q}$ which are dominated by a new point u to be inserted are also marked as such by incrementing $p.nd$ and setting $p.dom = u$. Note that, although the hypervolume contribution of a dominated point is zero by definition, if it is dominated by a single point, it also decreases the contribution of that point, which is why such dominated points must remain in Q. On the other hand, points dominated by two or more points do not affect the exclusive contribution of any of those points, and can be discarded. The insertion of new dominated points is not considered in this scenario.

Points $p \in \mathrm{Q}$ which are dominated by a single point ($p.nd = 1$) are processed in lines 23 to 28. After updating the volume $q.v$ associated with $q = p.dom$, the contribution of p^* to $q.\mathrm{L}^*$ is computed and subtracted from $q.a$. Then, any points in $q.\mathrm{L}$ whose projections are simultaneously dominated by p^* and q^* are discarded, and p is inserted into $q.\mathrm{L}$. In the example of Figure 4 (d), $p = s^{14}$ is dominated (only) by s^{10}. Therefore, $p.nd = 1$ and $q = p.dom = s^{10}$. Also, $p.cx = s^{12}$, $p.cy = s^6$ and $q.\mathrm{L} = \{s^{12}, s^7, s^6, s^5\}$. The contribution of p^* to $q.\mathrm{L}^*$ (line 26) is computed by visiting points s^{12}, s^7 and s^6. Then, s^7 is discarded (line 27) and p is added to $q.\mathrm{L}$ (line 28). At the end of the iteration, $s^{10}.\mathrm{L} = \{s^{12}, s^{14}, s^6, s^5\}$ contains the points whose projections are delimiters of s^{10^*} at $z = s^{14}_z$.

In Algorithm 4, each point in $p \in \mathrm{Q}$ is added to L (line 11) once. Moreover, each p other than the sentinel is added at most once to a list associated with a point in L as an inner delimiter (lines 8 or 28). Although the same point may be an outer delimiter (cx or cy) of several other points in Q, at most four outer delimiters are added to lists associated with points in L in each iteration ($p.cx$ and $p.cy$ are added to $p.\mathrm{L}$ in lines 9 and 10, and p is added to two lists in lines 16 and 21). Therefore, at most $(n+1)+n+4(n+1) = 6n+5$ points are added to lists, which is also an upper bound on the number of

points removed. In each iteration of the algorithm, points other than p are visited only to update the corresponding volumes or to update some area. This involves $O(1)$-time operations on each of those points through calls to updateVolumes and computeArea. Every point visited in those calls is either discarded right after (in removeDominated) or is an outer delimiter, and the number of outer delimiters visited in each call is a constant. Consequently, the time-complexity of Algorithm 4 amortizes to $O(n)$.

3) Updating All Contributions: By updating the HVC3D data structure and recomputing all contributions with Algorithm 4, the ALLCONTRIBUTIONSUPDATE problem can now be solved in $O(n)$ time. Although HVC3D extends the HV3D$^+$ data structure with additional point attributes, attributes cx and cy are still updated in linear time exactly as described in Subsection III-A4. When new points may dominate existing ones, attributes nd and dom also need to be updated as explained in Subsection IV-A2, which is also carried out in linear time. All other point attributes are for Algorithm 4's own use, and do not need to be updated on single point insertion or removal.

A potentially faster alternative is suggested by the very definition of the ALLCONTRIBUTIONSUPDATE problem. It consists in computing the contribution of the point u to be added to Q as described in Subsection III-A5 (which is not needed when removing a point), as well as the joint contributions of u and p to Q, for each *strong* delimiter $p \in \mathrm{Q}$ of the contribution of u, thus avoiding unnecessary computations. Going back to the example depicted in Figure 2 (a), if the contributions of the points in $\mathrm{X} = \{s^1, \ldots, s^9, s^{11}, \ldots, s^{14}\}$ are known and $u = p$ is the new point to be added to X, the contributions of $s^1, \ldots, s^4, s^{11}$ and s^{13} remain unchanged, and there is no need to recompute them.

Since it may be difficult to know in advance which delimiters are strong and whether an outer delimiter is a proper delimiter or not, let D^1 denote the set of all inner and outer delimiters of u in Q. Points in D^1 are used to compute the contribution of u, but they are also the points whose exclusive contribution may be decreased by the addition of u to Q. Therefore, in order to update their joint contribution with u, their own inner and outer delimiters must be considered. The set of points that are inner and outer delimiters of the joint contribution between u and individual points in D^1, but not of the contribution of u, will be denoted by $\mathrm{D}^2 \subseteq \mathrm{Q} \setminus \mathrm{D}^1$. The remaining points in $\mathrm{X} \setminus (\mathrm{D}^1 \cup \mathrm{D}^2)$ can be ignored. In Figure 2 (a), $\mathrm{D}^1 = \{s^5, \ldots, s^9, s^{12}, s^{14}\}$, $\mathrm{D}^2 = \{s^1, s^3, s^4, s^{11}, s^{13}\}$, and s^2 can be ignored. Note that points s^1, s^3, s^4 and s^{11} are in D^2 because they delimit the contribution of $(u \vee s^5)$, but not that of u.

Recall the computation of the contribution of u to Q as described in Subsection III-A5. After saving the current contributions and setting $p.v = 0$ for all $p \in \mathrm{Q}$, the points in $\{p \in \mathrm{Q} \mid p_z \leq u_z\}$ whose projections are inner or outer delimiters of the contribution of u^* to the projection of that set are added to list $\mathrm{L} \subseteq \mathrm{D}^1$. Then, for each $q \in \mathrm{L}$, a list $q.\mathrm{L}$ is constructed containing the outer and inner delimiters of the joint contribution between u^* and q^* at $z = u_z$. All of these lists can be set up in linear time as in Algorithm 4, by sweeping Q while $p_z \leq u_z$. In Figure 4 (a), considering $u = p$, $\mathrm{L} = \{s^5, \ldots, s^9\}$. Then, for each point $q^* \in \mathrm{L}^*$, the area of the region previously dominated exclusively by q^* that is also dominated by u^* at $z = u_z$ is computed in linear time, and stored in $q.a$. These regions are depicted in dark gray in Figure 4 (b).

After these initialization steps, the points $p \in \mathrm{Q}$ such that $p_z > u_z$ are visited in ascending order of p_z. These points are skipped unless they decrease the area associated with u or with any of the points in L. The joint contribution between p^* and u^* is computed and stored in $p.a$. Moreover, for each point $q \in \mathrm{L}$, the associated volume is updated as usual, and the area of the region jointly dominated by q^*, u^* and p^* is computed and subtracted from $q.a$.

The algorithm ends once all points in Q have been visited or after two points dominating u^* are encountered, as from that point onwards the joint contribution of u and any other point must be zero. Finally, if u is to be added to Q, the computed joint contributions are subtracted from the corresponding original values, and the HVC3D data structure is updated. If u was removed from Q, then the data structure has already been updated, and it suffices to add the computed joint contributions to their corresponding previous values.

B. Four Dimensions

Similarly to HV4D$^+$, a new algorithm named HVC4D for the ALLCONTRIBUTIONS problem in four dimensions is obtained by performing a sequence of HVC3D updates, as follows. Given a nondominated point set $\mathrm{X} \subset \mathbb{R}^4$, points in $\mathrm{X} \cup \{(-\infty, -\infty, -\infty, r_w)\}$ are sorted in ascending w-coordinate order, stored in a linked list Q, and visited in that order. For each visited point, p, the contribution of p^* to the projection, S^*, of the (initially empty) set, S, of points visited before p is computed, and the contributions of its inner and outer delimiters in $\mathrm{S}^* \cup \{p^*\}$ are updated as described in Subsection IV-A3. Then, p is added to S, and the contribution of each point in S^* is multiplied by the difference between the w-coordinate of the next point in Q and p_w to obtain the hypervolume of the current four-dimensional slice of each individual contribution. Slice hypervolumes are accumulated separately for each point to obtain the corresponding contributions in four dimensions. Since the algorithm performs n linear-time HVC3D updates, the time complexity of this algorithm is $O(n^2)$.

Handling points which are dominated in three dimensions is required to correctly compute ALLCONTRIBUTIONS in four dimensions because a point $q^* \in \mathrm{S}^*$ which is dominated by a single point $p^* \in \mathrm{S}^*$ decreases the contribution of p^*. Since dominated points are not considered in the EF algorithm, iterating over EF without modification to compute all contributions in four dimensions would not work.

C. Practical Implications

The HVC3D algorithm improves the current upper bound on the time complexity of ALLCONTRIBUTIONSUPDATE in three dimensions from $O(n \log n)$ to $O(n)$. Using this algorithm in HVC4D improves the current upper bound on the complexity

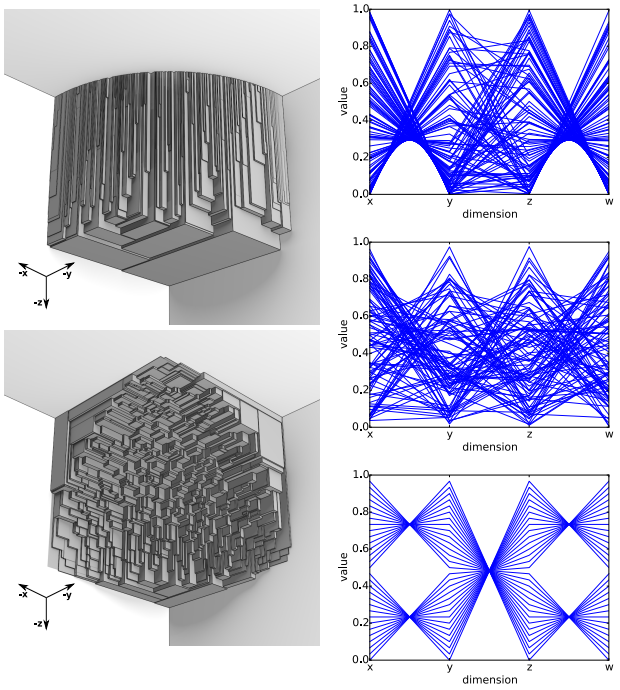


Fig. 5. Three-dimensional data sets: cliff (top left) and spherical (bottom left). Four-dimensional data sets: cliff (top right), spherical (middle right) and hard (bottom right).

of AllContributions in four dimensions from $O(n^2 \log n)$ to $O(n^2)$ time. Practical implications of these algorithms include at least the following:

- Faster implementation of the decremental greedy approximation to the HSSP in three and four dimensions, which can now be computed in at most $O(n(n-k)+n\log n)$ and $O(n^2(n-k))$ time, respectively.
- Faster implementation of archive-based EMO algorithms such as SMS-EMOA, also in three and four dimensions, even when the reference point is adjusted as the population moves towards the Pareto Front.

In addition, the proposed data structures, preprocessing and computation techniques are likely to be useful in the development of more efficient algorithms for related problems, namely for the computation and update of the Expected Hypervolume Improvement (EHVI) [39].

V. Experimental Results

To evaluate the potential impact of the proposed algorithms in practice, they were evaluated experimentally and compared to their most direct competitors in the literature, considering a number of relevant scenarios and concrete data sets.

A. Experimental Setup

The proposed algorithms were implemented in C. All codes used in the experiments[1] were compiled with gcc 5.3.1 and flags -march=corei7 -O3. Tests were run on an Intel Core i7-3612QM 2.10GHz CPU with 6 MB cache and 8 GB of RAM.

[1]The source code for the proposed algorithms (HVC package), as well as for HV4D (v1.2), gHSS (v1.1) and HBDA is available through https://eden.dei.uc.pt/~cmfonsec/software.html. HV4DX, WFG (1.11) and IWFG (1.01) implementations are made available by the respective authors at http://www.wfg.csse.uwa.edu.au/hypervolume/.

To evaluate the performance of the algorithms, *cliff* and (concave) *spherical* data sets [17] containing 10^5 points each were generated at random. Spherical data sets were generated as sets of points $p \in \mathbb{R}^d$ such that $p_i = |X_i|/\|X\|$, where $X_i \sim \mathcal{N}(0,1)$ for all $i = 1,\ldots,d$. The cliff data sets were such that $p_i = 1 - |X_i|/\|X\|$, where $X_i \sim \mathcal{N}(0,1)$ for $i = 1,2$, and $p_3 \sim \mathcal{U}(0,1)$ if $d = 3$, whereas $p_{j+2} = 1 - |Y_j|/\|Y\|$, $Y_j \sim \mathcal{N}(0,1)$, $j = 1,2$ if $d = 4$. All smaller sets of points were generated by sampling the initial sets of 10^5 points at random. Additionally, *hard* data sets for $d = 4$, as proposed by Lacour *et al.* [33], were generated for every set size $n \leq 10^5$ considered. Considering an even set size, n, the *hard* data set is the set of points $p \in \mathbb{R}^4$ such that $p^j = (\frac{n+2j}{2n}, \frac{n-j-1}{n}, \frac{j}{n}, \frac{n-2j-2}{2n})$ for $j = 0,...,\frac{n}{2}-1$ and $p^j = (p^l_w, p^l_z, p^l_y, p^l_x)$ for $j = \frac{n}{2},...,n-1$ where $l = j - \frac{n}{2}$. Figure 5 illustrates the various data sets. The reference point used was $(1,\ldots,1)$.

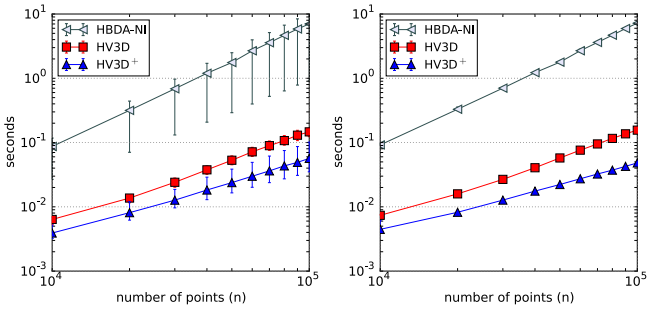
The plots presented next show runtimes for growing numbers of points on the above types of data sets. Because many algorithms for the hypervolume indicator are sensitive to objective reordering [32], each data point and the corresponding error bar on a plot represent the average, minimum and maximum runtimes over all permutations of the objectives for a single set instance (6 permutations for $d = 3$, and 24 for $d = 4$). Due to the computational effort required, smaller sets of up to 10^4 points were considered in some experiments. In general, greater variability was observed on cliff data sets than on spherical data sets.

B. Hypervolume Indicator

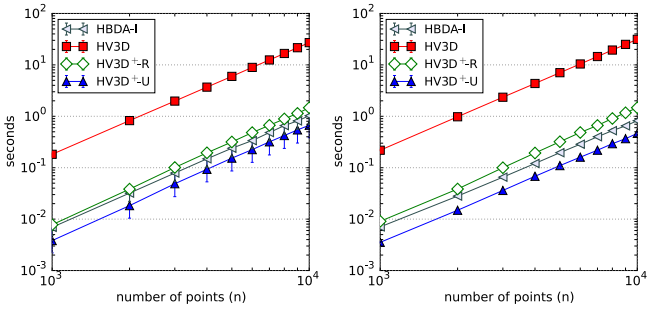
Figure 6 (a) shows that HV3D$^+$ is generally faster than the original HV3D at computing Hypervolume in three dimensions, despite sweeping input sets twice. The runtime of HBDA-NI appears to grow quadratically, as expected.

Results for the case of an initially empty unbounded archive whose hypervolume indicator value was updated each time a single point from a given test set was added to it are presented in Figure 6 (b). Since all test sets contained only nondominated points, the size of the archive increased with every new point. Concerning HV3D, the hypervolume indicator was computed n times for growing archive size $k = 1,\ldots,n$, resulting in an $O(n^2 \log n)$-time algorithm for this scenario. This is compared to HV3D$^+$ linear-time updates, where recomputing the hypervolume indicator for each new point is denoted by HV3D$^+$-R, and computing and adding the contribution of the new point to the current value of the indicator is denoted by HV3D$^+$-U. Both update approaches take $O(n^2)$ time on the whole problem. It can be seen that both HV3D$^+$-R and HV3D$^+$-U clearly outperform HV3D, with speed-ups of up to 23 and 47 times on cliff data sets, and up to 25 and 68 times on spherical data sets, respectively. HV3D$^+$-U was up to 3 times faster than HV3D$^+$-R. HBDA-I is slightly faster that HV3D$^+$-R, but is still outperformed by HV3D$^+$-U, which was up to twice as fast as HBDA-I.

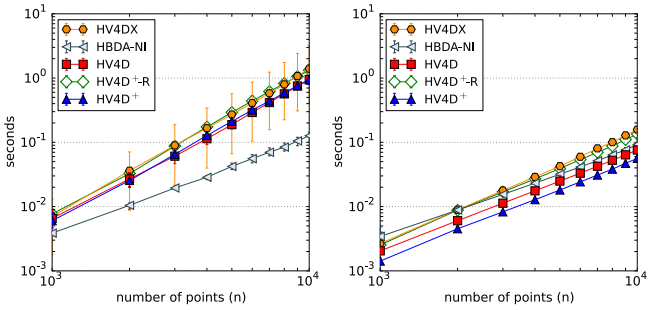
Finally, Figures 6 (c) and 7 show that the default HV4D$^+$ (based on single contribution updates) and a variant HV4D$^+$-R based on full recalculation updates are competitive with HV4D



(a) HYPERVOLUME computation for $d = 3$



(b) Sequential incremental HYPERVOLUMEUPDATE for $d = 3$



(c) HYPERVOLUME computation for $d = 4$

Fig. 6. Runtime performance of algorithms on different hypervolume indicator problems and data sets: cliff (left) and spherical (right).

for HYPERVOLUME computation in four dimensions, with HV4D$^+$ generally matching or exceeding the performance of HV4D. Although the quadratic time complexity of HBDA-NI for $d = 4$ can be observed on the hard data set (Figure 7, left), it exhibited sub-quadratic behavior on the cliff data set, where it clearly outperformed the other algorithms. Nevertheless, it was up to 3 times slower than HV4D$^+$ on the other data sets.

Finally, the claimed [30] performance improvement of HV4DX over the $O(n^2)$-time HV4D algorithm could not be observed. Not only was it up to twice as slow on average as the original HV4D implementation on the cliff and spherical data sets (Figure 6 (c)), it also exhibited cubic runtime growth on the hard data set (Figure 7, right).

C. Hypervolume Contributions

Regarding the computation of ALLCONTRIBUTIONS in three dimensions, it can be observed in Figure 8 (a) that HVC3D remains competitive with EF, as expected. In comparison to a dedicated adaptation of WFG[2], here referred to as WFG-c, they were 34 to 728 times faster in the tests performed. Since computing ALLCONTRIBUTIONS can also be used to identify a least contributor, IWFG was included in the comparison, but it was nevertheless 21 to 456 times slower than HVC3D or EF.

[2]Version 1.11 of WFG was adapted to iterate over the function used to compute the contribution of a single point, which is faster than iterating over WFG as such.

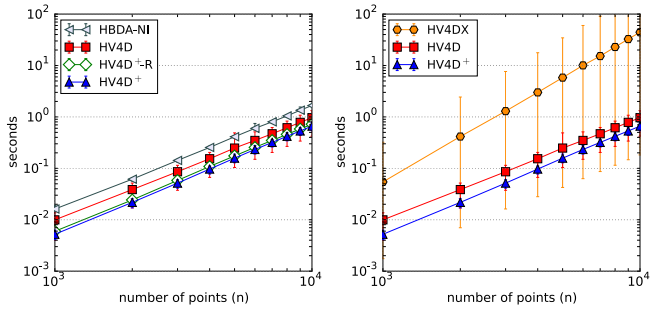
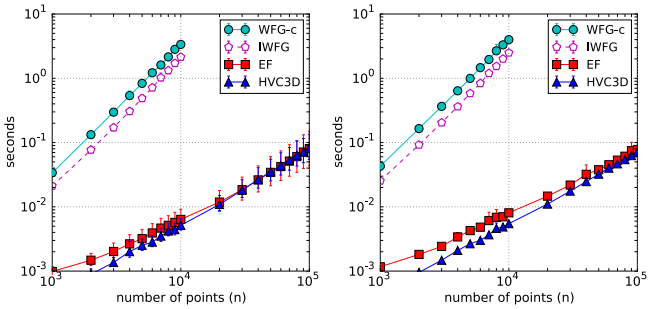


Fig. 7. Runtime performance of algorithms for HYPERVOLUME computation for $d = 4$ on the hard data set.

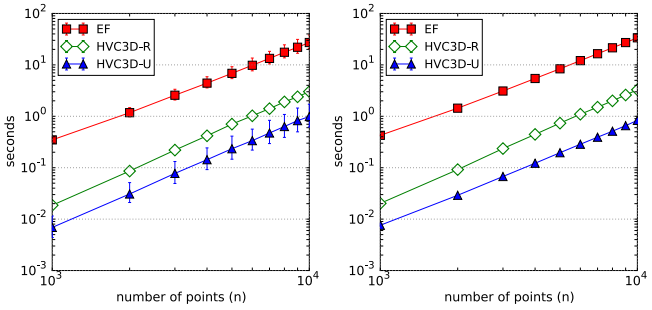
Figure 8 (b) shows the runtimes for the sequential update of all contributions in three dimensions on the unbounded archive setup described earlier in connection with Figure 6 (b). Similarly to HV3D in that case, EF was called n times when filling an archive of size n, corresponding to an overall $O(n^2 \log n)$ time complexity. The two HVC3D variants, -R (recomputing) and -U (contribution updates), take overall $O(n^2)$ time due to the linear-time updates. Both HVC3D-U and HVC3D-R clearly outperform EF in this scenario, showing speed-ups of up to 56 and 21, respectively. HVC3D-U performed up to 4 times faster than HVC3D-R.

Results for the simulation of a bounded archive similar to the environmental selection process in SMS-EMOA are presented in Figure 8 (c). A fixed archive of size 200 was updated $n - 200$ times by each algorithm by adding a new point and then removing a least contributor. In the case of HVC3D-U, contributions were updated after adding the new point and again after removing a least contributor. On the other hand, since contributions only need to be known after a new point is added in order to identify the least contributor, with HVC3D-R only the data structure was updated when a least contributor is removed. Insertion of a new point caused a data structure update followed by the computation of all contributions. Similarly, contributions were recomputed only on point insertions when using EF. The runtimes include the computation of the initial archive with 200 solutions, in $O(n \log n)$ time in all cases. HVC3D-U and HVC3D-R were up to 50 and 30 times faster than EF, respectively. The results show that, even though contributions were recomputed by HVC3D-R only half the time, it was still up to 2 times slower than HVC3D-U.

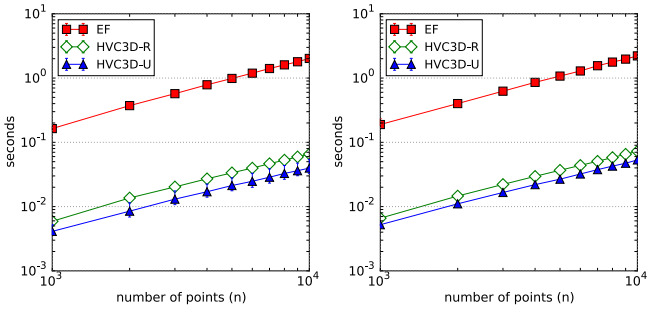
In Figure 9, results are presented for the decremental greedy approximation to the HSSP in three dimensions, computed by iterating over HVC3D-U to discard the least contributor until k of the initial $n = 10^4$ points are left, and denoted gHSSD. This algorithm has a time complexity of $O(n(n-k)+n \log n)$, which contrasts with the $O(nk + n \log n)$ complexity of the incremental greedy algorithm (gHSS) [14]. Regarding the quality of the approximation, it can be observed that the



(a) ALLCONTRIBUTIONS computation for $d = 3$



(b) Sequential incremental ALLCONTRIBUTIONSUPDATE for $d = 3$



(c) Sequential ALLCONTRIBUTIONSUPDATE for $d = 3$

Fig. 8. Runtime performance of algorithms on different all contribution problems and data sets: cliff (left) and spherical (right).

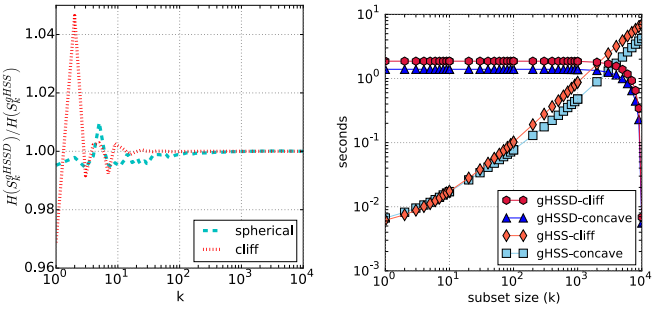


Fig. 9. Comparison of decremental (gHSSD) and incremental (gHSS) greedy algorithms for the HSSP: Approximation quality (left) and runtime (right).

quality of the subsets produced by the two algorithms is very similar (the hypervolume ratio is very close to one), except for smaller values of k. Also, the decremental greedy approach produced slightly worse subsets than the incremental approach on spherical data for intermediate values of k. Regarding runtime, gHSSD was observed to be faster than gHSS for $k \geq 2n/5$ on the spherical front and for $k \geq n/5$ on the cliff front.

Finally, runtimes for ALLCONTRIBUTIONS in four dimensions are shown in Figure 10, where HVC4D is compared

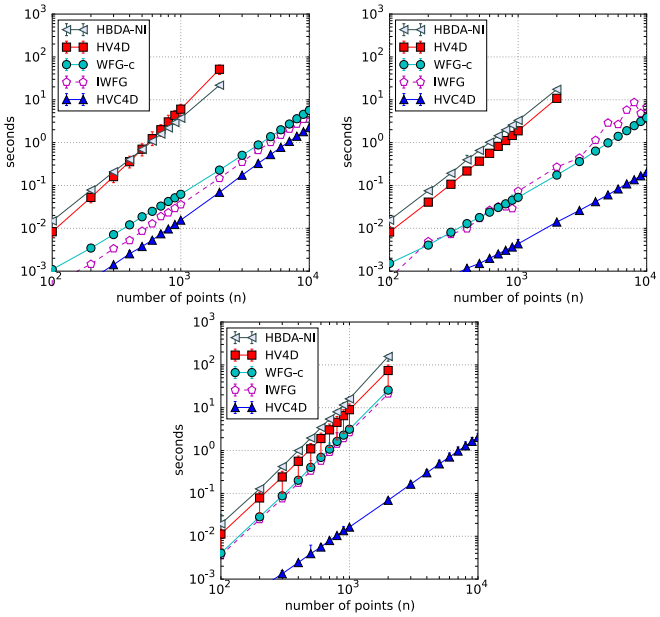


Fig. 10. Runtime performance of algorithms for all contributions in four dimensions on the cliff (top left), spherical (top right) and hard (bottom) data sets.

to WFG-c, HV4D and HBDA-NI. HV4D and HBDA-NI are called $n + 1$ times for each set, and therefore the whole computation has a time complexity of $O(n^3)$. As expected, HVC4D significantly outperformed both HV4D and HBDA-NI, with observed speed-ups ranging between 45 and 1069 and between 81 and 2270, respectively. HV4D and HBDA-NI were also significantly outperformed by WFG-c, but HVC4D was still 3 to 372 times faster than WFG-c. IWFG was also included for reference.

VI. CONCLUDING REMARKS

Computational problems related to the hypervolume indicator frequently arise in connection with the design, implementation, and experimental evaluation of evolutionary algorithms and other metaheuristics for multiobjective optimization. Arguably, the development of algorithms for such problems in the literature has taken three main directions to date, one aiming for algorithms that are fast in practice, especially for large numbers of objectives, a second one focusing on algorithm complexity in relation to the number of objectives, and a third directed at low-dimensional cases. The last direction typically encompasses two and three objectives, with occasional incursions into four objectives, which remain the most common use cases in multiobjective optimization in spite of growing interest in so-called many-objective optimization, and for which it has been possible to develop algorithms that are both asymptotically efficient, or even optimal, and very fast in practice.

In this work, new algorithms for the computation and update of hypervolume contributions were developed by building upon existing algorithmic approaches to the computation of the hypervolume indicator in three and four dimensions. A novel $O(n \log n)$-time preprocessing step for the three-dimensional case was the key ingredient in the

development of $O(n)$-time algorithms for the subsequent computation of HYPERVOLUME, ONECONTRIBUTION and ALLCONTRIBUTIONS, as well as for the corresponding HYPERVOLUMEUPDATE and ALLCONTRIBUTIONSUPDATE problems in three dimensions, even under reference point changes. As a direct result, a novel algorithm for ALLCONTRIBUTIONS in four dimensions was obtained, and a new time complexity upper bound of $O(n^2)$ was established for this problem. Using the proposed algorithms, the decremental greedy approximation to the HSSP can now be computed in $O(n(n-k) + n \log n)$ and $O(n^2(n-k))$ time, in three and four dimensions, respectively.

The experimental results obtained indicate that the better complexity bounds achieved by the proposed algorithms do translate into considerable speed-ups in practice.

ACKNOWLEDGMENTS

This work was supported by national funds through the Portuguese Foundation for Science and Technology (FCT) and by the European Regional Development Fund (FEDER) through COMPETE 2020 – Operational Program for Competitiveness and Internationalization (POCI). Andreia P. Guerreiro acknowledges Fundação para a Ciência e a Tecnologia (FCT) for Ph.D. studentship SFHR/BD/77725/2011, co-funded by the European Social Fund and by the State Budget of the Portuguese Ministry of Education and Science in the scope of NSRF–HPOP–Type 4.1–Advanced Training.

REFERENCES

[1] M. Ehrgott, *Multicriteria Optimization*. Springer, second ed., 2005.
[2] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc., 2001.
[3] J. D. Knowles, D. W. Corne, and M. Fleischer, "Bounded archiving using the Lebesgue measure," in *IEEE CEC*, vol. 4, pp. 2490–2497, 2003.
[4] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms – A comparative case study," in *PPSN V*, vol. 1498 of *LNCS*, pp. 292–301, Springer, 1998.
[5] S. Huband, P. Hingston, L. While, and L. Barone, "An evolution strategy with probabilistic mutation for multi-objective optimisation," in *IEEE CEC*, vol. 4, pp. 2284–2291, 2003.
[6] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *PPSN VIII*, vol. 3242 of *LNCS*, pp. 832–842, Springer, 2004.
[7] M. Emmerich, N. Beume, and B. Naujoks, "An EMO algorithm using the hypervolume measure as selection criterion," in *EMO*, vol. 3410 of *LNCS*, pp. 62–76, Springer, 2005.
[8] J. Bader and E. Zitzler, "HypE: An algorithm for fast hypervolume-based many-objective optimization," *Evol. Comput.*, vol. 19, pp. 45–76, 2011.
[9] K. Bringmann, T. Friedrich, and P. Klitzke, "Two-dimensional subset selection for hypervolume and epsilon-indicator," in *GECCO*, pp. 589–596, ACM, 2014.
[10] T. Kuhn, C. M. Fonseca, L. Paquete, S. Ruzika, M. M. Duarte, and J. R. Figueira, "Hypervolume subset selection in two dimensions: Formulations and algorithms," *Evol. Comput.*, vol. 24, no. 3, pp. 411–425, 2016.
[11] K. Bringmann and T. Friedrich, "An efficient algorithm for computing hypervolume contributions," *Evol. Comput.*, vol. 18, pp. 383–402, 2010.
[12] L. Bradstreet, L. While, and L. Barone, "Incrementally maximising hypervolume for selection in multi-objective evolutionary algorithms," in *IEEE CEC*, pp. 3203–3210, 2007.
[13] T. Friedrich and F. Neumann, "Maximizing submodular functions under matroid constraints by multi-objective evolutionary algorithms," in *PPSN XIII*, vol. 8672 of *LNCS*, pp. 922–931, Springer, 2014.
[14] A. P. Guerreiro, C. M. Fonseca, and L. Paquete, "Greedy hypervolume subset selection in low dimensions," *Evol. Comput.*, vol. 24, pp. 521–544, Fall 2016.
[15] N. Beume, B. Naujoks, and M. Emmerich, "SMS-EMOA: Multiobjective selection based on dominated hypervolume," *Eur. J. Oper. Res.*, vol. 181, pp. 1653–1669, 2007.
[16] T. M. Chan, "Klee's measure problem made easy," in *IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 410–419, IEEE Computer Society, 2013.
[17] M. T. M. Emmerich and C. M. Fonseca, "Computing hypervolume contributions in low dimensions: Asymptotically optimal algorithm and complexity results," in *EMO*, vol. 6576 of *LNCS*, pp. 121–135, Springer, 2011.
[18] C. Igel, N. Hansen, and S. Roth, "Covariance matrix adaptation for multi-objective optimization," *Evol. Comput.*, vol. 15, pp. 1–28, 2007.
[19] I. Hupkens and M. Emmerich, "Logarithmic-time updates in SMS-EMOA and hypervolume-based archiving," in *EVOLVE IV*, vol. 227 of *Advances in Intelligent Systems and Computing*, pp. 155–169, Springer, 2013.
[20] L. Bradstreet, L. Barone, and L. While, "Updating exclusive hypervolume contributions cheaply," in *IEEE CEC*, pp. 538–544, 2009.
[21] S. Jiang, J. Zhang, Y. Ong, A. Zhang, and P. Tan, "A simple and fast hypervolume indicator-based multiobjective evolutionary algorithm," *IEEE Trans. on Cybern.*, vol. 45, pp. 2202–2213, 2015.
[22] K. Bringmann and T. Friedrich, "Convergence of hypervolume-based archiving algorithms," *IEEE Trans. Evol. Comput.*, vol. 18, pp. 643–657, 2014.
[23] N. Beume, C. Fonseca, M. López-Ibáñez, L. Paquete, and J. Vahrenhold, "On the complexity of computing the hypervolume indicator," *IEEE Trans. Evol. Comput.*, vol. 13, pp. 1075–1082, 2009.
[24] A. P. Guerreiro, C. M. Fonseca, and M. T. M. Emmerich, "A fast dimension-sweep algorithm for the hypervolume indicator in four dimensions.," in *CCCG*, pp. 77–82, 2012.
[25] L. While, L. Bradstreet, and L. Barone, "A fast way of calculating exact hypervolumes," *IEEE Trans. Evol. Comput.*, vol. 16, pp. 86–95, 2012.
[26] L. Russo and A. Francisco, "Quick hypervolume," *IEEE Trans. Evol. Comput.*, vol. 18, pp. 481–502, 2014.
[27] C. M. Fonseca, L. Paquete, and M. López-Ibáñez, "An improved dimension-sweep algorithm for the hypervolume indicator," in *IEEE CEC*, pp. 1157–1163, IEEE Press, 2006.
[28] H. Yıldız and S. Suri, "On Klee's measure problem for grounded boxes," in *Proceedings of the Twenty-eighth Annual Symposium on Computational Geometry*, SoCG '12, pp. 111–120, ACM, 2012.
[29] L. While, P. Hingston, L. Barone, and S. Huband, "A faster algorithm for calculating hypervolume," *IEEE Trans. Evol. Comput.*, vol. 10, pp. 29–38, 2006.
[30] W. Cox and L. While, *Improving and Extending the HV4D Algorithm for Calculating Hypervolume Exactly*, pp. 243–254. Springer, 2016.
[31] L. Bradstreet, L. While, and L. Barone, "A fast incremental hypervolume algorithm," *IEEE Trans. Evol. Comput.*, vol. 12, pp. 714–723, 2008.
[32] L. Bradstreet, L. While, and L. Barone, "A fast many-objective hypervolume algorithm using iterated incremental calculations," in *IEEE CEC*, pp. 179–186, 2010.
[33] R. Lacour, K. Klamroth, and C. M. Fonseca, "A box decomposition algorithm to compute the hypervolume indicator," *Computers & Operations Research*, vol. 79, pp. 347–360, 2017.
[34] W. Cox and L. While, "Improving the IWFG algorithm for calculating incremental hypervolume," in *IEEE CEC*, pp. 3969–3976, 2016.
[35] K. Bringmann and T. Friedrich, "Approximating the least hypervolume contributor: NP-hard in general, but fast in practice," in *EMO*, vol. 5467 of *LNCS*, pp. 6–20, Springer, 2009.
[36] K. Nowak, M. Märtens, and D. Izzo, "Empirical performance of the approximation of the least hypervolume contributor," in *PPSN XIII*, pp. 662–671, Springer, 2014.
[37] G. Rote, "Selecting k points that maximize the convex hull volume," in *The 19th Japan Conference on Discrete and Computational Geometry, Graphs, and Games*, pp. 58–59, 2016.
[38] L. Bradstreet, L. Barone, and L. While, "Maximising hypervolume for selection in multi-objective evolutionary algorithms," in *IEEE CEC*, pp. 1744–1751, 2006.
[39] K. Yang, A. Emmerich, Michael Deutz, and C. M. Fonseca, "Computing 3-D expected hypervolume improvement and related integrals in asymptotically optimal time," in *EMO*, pp. 685–700, Springer, 2017.

Andreia P. Guerreiro is currently a Ph.D. student and a member of the Evolutionary and Complex Systems (ECOS) group of the Centre for Informatics and Systems of the University of Coimbra (CISUC), Portugal. She obtained her Masters degree in Information Systems and Computer Engineering from Instituto Superior Técnico, Portugal, in 2011. Her research work is concerned with the development and analysis of efficient indicator-based selection algorithms for evolutionary multiobjective optimization, as well as algorithms for performance assessment in multiobjective optimization. Additional research interests include computational geometry and algorithm design.

Carlos M. Fonseca is an Associate Professor at the Department of Informatics Engineering of the University of Coimbra, Portugal, and a member of the Evolutionary and Complex Systems (ECOS) group of the Centre for Informatics and Systems of the University of Coimbra (CISUC). He graduated in Electronic and Telecommunications Engineering from the University of Aveiro, Portugal, in 1991, and obtained his doctoral degree from the University of Sheffield, U.K., in 1996. His research has been devoted mainly to evolutionary computation and multiobjective optimization. His current research interests include multiobjective optimization, evolutionary algorithms, dynamical systems, and engineering-design optimization.