2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

An Improved Dimension-Sweep Algorithm for the Hypervolume Indicator

Carlos M. Fonseca, Luís Paquete, and Manuel López-Ibáñez

Abstract— **This paper presents a recursive, dimension-sweep algorithm for computing the hypervolume indicator of the quality of a set of n non-dominated points in $d > 2$ dimensions. It improves upon the existing HSO (Hypervolume by Slicing Objectives) algorithm by pruning the recursion tree to avoid repeated dominance checks and the recalculation of partial hypervolumes. Additionally, it incorporates a recent result for the three-dimensional special case. The proposed algorithm achieves $O(n^{d-2} \log n)$ time and linear space complexity in the worst-case, but experimental results show that the pruning techniques used may reduce the time complexity exponent even further.**

I. Introduction

The performance assessment of algorithms for multiobjective optimization problems is far from being a trivial issue. Recent results indicate that unary performance measures, i.e. performance measures which assign a single value to each non-dominated point set, are inherently limited in their inferential power [1]. Despite these limitations, the hypervolume indicator is still considered to possess some reasonable properties, having also been proposed as a guidance criterion for accepting solutions in Multiobjective Evolutionary Algorithms [2], [3]. Therefore, the computational time taken for computing the hypervolume indicator is a crucial factor for the performance of such algorithms.

The paper is organized as follows. In the next section, the hypervolume indicator is defined, and some background on its computation is provided. Then, an algorithm is proposed which improves upon current practice in the Evolutionary Computing community, while remaining simple to implement. The results of an experimental study conducted to validate the implementation are presented and compared to previously published results to the extent possible. The paper concludes with a discussion of the relative merits of the various techniques proposed and with directions for future work.

II. Background

The unary hypervolume indicator [4] is a measure of the quality of a set $P = \{p^{(1)}, p^{(2)}, \ldots, p^{(n)}\}$ of n non-dominated objective vectors produced in a run of a multiobjective optimizer, such as a multiobjective evolutionary algorithm. Assuming a minimization problem involving d objectives, this indicator consists of the measure of the region which is simultaneously dominated by P and bounded above by a reference point $r \in \mathbb{R}^d$ such that $r \geq (\max_p p_1, \ldots, \max_p p_d)$, where $p = (p_1, \ldots, p_d) \in P \subset \mathbb{R}^d$, and the relation $\geq$ applies componentwise. As illustrated in Fig. 1, this region consists of an orthogonal polytope, and may be seen as the union of n axis-aligned hyper-rectangles with one common vertex (the reference point, r).

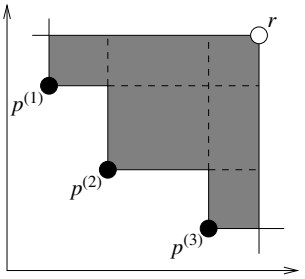

Fig. 1. The hypervolume indicator in the two-objective case

Several algorithms have been proposed for computing the hypervolume indicator in any number of dimensions. The inclusion-exclusion algorithm [5] is a natural, brute-force approach based on the inclusion-exclusion principle [6] which exploits the known structure of the polytope directly. Fleischer [7] proposed an apparently efficient algorithm, called LebMeasure, but a more detailed analysis has shown the worst-case time complexity of this algorithm to be at least $O(n^d)$ [8].

More recently, While et al. [9] have proposed the so-called HSO (Hypervolume by Slicing Objectives) algorithm, which is, in fact, a dimension-sweep [10] approach. The point set is swept by a $(d-1)$-dimensional hyperplane along the first coordinate, defining d-dimensional slices between consecutive points. The total hypervolume is computed as the sum of the hypervolumes of each individual slice, which can be obtained by multiplying its height (measured along the first dimension) by the hypervolume of its $(d-1)$-dimensional base. This leads to the computation of n different hypervolumes in $d-1$ dimensions, each of which can be solved in a similar way. The process is repeated until a sufficiently small number of dimensions (two, in this case) is reached. Indeed, the two-dimensional case can be easily solved in linear time after a preliminary sorting step. The HSO algorithm runs in $O(n^{d-1})$ time for $d > 2$.

Remarkably, the more general form of this problem which is obtained by removing the common vertex constraint has

Carlos M. Fonseca (cmfonsec@ualg.pt) is with Centro de Sistemas Inteligentes, Faculdade de Ciências e Tecnologia, Universidade do Algarve and **Luís Paquete** (lpaquete@ualg.pt) is with Faculdade de Economia and Centro de Sistemas Inteligentes, Universidade do Algarve, 8005-139 Faro, Portugal

Manuel López-Ibáñez (m.lopez-ibanez@napier.ac.uk) is with Centre for Emergent Computing, School of the Built Environment, Napier University, EH10 5DT, Edinburgh, UK

0-7803-9487-9/06/$20.00/©2006 IEEE

already been studied in the field of Computational Geometry. It is known both as the *measure of the union of rectangles in d-dimensions* and as the *d-dimensional Klee's Measure Problem* [11][12], and can be solved in $O(n^{d/2} \log n)$-time in the decision-tree computational model [12].

Very recently, Paquete et al. [13] have noted the connection between these two problems. Since the same upper-bound applies to the computation of the hypervolume indicator, HSO is in fact sub-optimal for all $d > 2$. Furthermore, they have produced an even faster, $O(n \log n)$-time, algorithm to compute the hypervolume indicator in three dimensions, also based on dimension-sweeping.

Extending these results [9][13], this article presents a recursive dimension-sweep algorithm which exhibits $O(n^{d-2} \log n)$-time and linear space complexity for $d > 2$, matching the time complexity of Overmars and Yap's algorithm [12] for $d = 4$ and improving upon HSO [9] for $d > 2$ by a factor of $n/\log n$. Although this new algorithm has higher time complexity than Overmars and Yap's algorithm for $d > 4$, it is conceptually much simpler, and is much easier to implement.

III. Proposed algorithm

The algorithm proposed here follows up directly on While's work on HSO [9], improving upon it in three ways. Firstly, a new efficient algorithm by Paquete et al. [13] is used for the three-dimensional case, allowing one level of recursion (or slicing) to be avoided. Secondly, the set of non-dominated points is stored in a dedicated data structure throughout the operation of the algorithm, avoiding repeated slicing and merging of lists, and maintaining linear space complexity. Finally, this data structure is extended to store the current dominance state of each point as well as intermediate hypervolume values, allowing the recursion tree to be pruned whenever results computed previously can be reused. Throughout the paper, time complexity is derived while considering the decision tree computational model.

A. Three-dimensional case

Paquete et al. [13] have developed a dimension-sweep algorithm for the three-dimensional case based on an existing algorithm for the maxima of a point set [10]. Considering minimization, the operation of the algorithm may be briefly described as follows.

Let the non-dominated points in the input set P be sorted in ascending order of their z-coordinate values p_z, where $p \in P$, and stored in a list L. Let $\mathit{next}_z(p)$ and $\mathit{prev}_z(p)$ denote the points immediately above and below p on the z coordinate, respectively (and similarly for the x and y coordinates). Furthermore, maintain a threaded, height-balanced binary tree T, which will be used to store the non-dominated projections on the (x, y)-plane of the points in L, using the y coordinate as key. Note that any non-dominated set in two dimensions may be strictly ordered with respect to either dimension. Insert the first point in L into T, determine the area A of the region which its projection on the (x, y)-plane dominates (given the reference point), and initialize the volume V to zero.

Then, for each new p in L, add to V the product of A by the difference between the z coordinates of p and $\mathit{prev}_z(p)$. Query T to determine where p should be inserted on the y axis, and determine $q = \mathit{prev}_y(p)$, if it exists. If $q_x \leq p_x$, then p is dominated by q and it should be discarded. Otherwise, determine $t = \mathit{next}_y(p)$, if it exists. If $p_x \leq t_x$, then the (x, y)-projection of t is now dominated by the corresponding projection of p. In this case, proceed by deleting t from T, and update A accordingly. Note that A can be updated in constant time given the knowledge of q, $\mathit{next}_y(t)$ and the reference point r. Repeat this procedure for each new $t = \mathit{next}_y(p)$ until no more points in T are dominated by p. Finally, insert p into T and update A to reflect the insertion. The whole procedure is repeated until all points in L are considered.

Since the algorithm performs at most n tree insertions and n tree deletions, each at a cost of $O(\log n)$ time, the time complexity of this algorithm is $O(n \log n)$. Note that T is threaded, which makes it possible to determine $\mathit{prev}_y(p)$ and $\mathit{next}_y(p)$ in constant time.

B. A recursive algorithm for $d > 3$

To support dimension sweeping in more than three dimensions, the input set P is now sorted in ascending order along each dimension i, for $3 \leq i \leq d$, and stored in $d - 2$ circular double-linked lists L_i with sentinels $\mathit{nil}(L_i)$, such that $\mathit{next}_i(\mathit{nil}(L_i)) \leftarrow \mathit{head}(L_i)$ and $\mathit{prev}_i(\mathit{nil}(L_i)) \leftarrow \mathit{tail}(L_i)$. This preprocessing stage takes $O(dn \log n)$ time.

The main advantage of the data structure adopted is that, when sweeping along a given dimension i, it becomes rather easy to delete points in lower dimensions, and to reinsert them later in the correct position, all in $O(d)$-time, provided deletion and reinsertion always occur in reverse order. A similar data structure has previously been used in the computation of the multivariate empirical cumulative distribution function for a set of points in $\mathbb{R}^d$ [14].

Pseudo-code is given in Algorithm 1. The algorithm is organized as two consecutive loops, and is invoked with $i = d$. In the first loop, the current dimension i is swept in descending order of p_i, and the points found are deleted (via $\mathit{delete}(\cdot)$) from all lists corresponding to dimensions lower than i, until the minimum value of coordinate i is reached. Then, the algorithm recurses for lower dimensions in order to compute the hypervolume indicator of the $(i-1)$-dimensional polytope defined by the remaining point(s). Note that, whenever a single point is left, the corresponding indicator value could also be computed directly. As soon as dimension three is reached, the hypervolume indicator is computed by the procedure explained in Section III-A.

In the second loop, the same dimension i is swept in reverse (i.e., ascending) order, reinserting one point at the time in the lower-dimension lists (via $\mathit{reinsert}(\cdot)$). As in HSO, the hypervolume indicator associated with the slice

Algorithm 1 $H(i, L_i, r, len)$

Require: i is the dimension, L_i is the linked list at dimension i, r is the reference point, and $len = |L_{i-1}|$

1: **if** $i = 3$ **then**
2: {see Section III-A.}
3: **else if** $i > 3$ **then**
4: $hvol \leftarrow 0$
5: $p \leftarrow nil(L_i)$
6: **while** $len > 1$ **do**
7: $p \leftarrow prev_i(p)$
8: $delete(p, i)$
9: $len \leftarrow len - 1$
10: $q \leftarrow prev_i(p)$
11: $\mathcal{H} \leftarrow H(i-1, L_{i-1}, r, len)$
12: **while** $p \neq nil(L_i)$ **do**
13: $hvol \leftarrow hvol + \mathcal{H} \cdot (p_i - q_i)$
14: $reinsert(p, i)$
15: $len \leftarrow len + 1$
16: $q \leftarrow p$
17: $p \leftarrow next_i(p)$
18: $\mathcal{H} \leftarrow H(i-1, L_{i-1}, r, len)$
19: $hvol \leftarrow hvol + \mathcal{H} \cdot (r_i - q_i)$
20: **return** $hvol$

defined by the i-th coordinates of the two consecutive points p and q ($p_i > q_i$) is obtained by multiplying the height of the slice ($p_i - q_i$) by the hypervolume indicator of the $(i-1)$-dimensional polytope defined by the points in L_{i-1}, including q. Here, this indicator value is computed recursively in steps 11 and 18 of the algorithm.

The three-dimensional special case allows $O(n^{d-2} \log n)$-time complexity to be achieved, which is faster than HSO. In addition, linear space complexity is maintained, since deletions and reinsertions operate on the given lists in place. HSO's description is less specific in this regard: the algorithm is explained in terms of a breath-first tree-expansion, which would lead to exponential space complexity, but the authors do mention processing intermediate lists of points as they are generated as a useful improvement, suggesting a depth-first, linear-space implementation.

The basic algorithm just described may be improved by using the data structure adopted to remember intermediate calculations, so that the recursion tree may be effectively pruned in certain common circumstances. This is possible without compromising on space complexity, as will be explained next.

C. Further improvements

1) Skipping over dominated points: Whenever a new point p is found to be dominated by another point q already in L_i (with respect to dimensions $1, \ldots, i$ only), the same will occur for dimensions $1, \ldots, j$ when $j < i$, because q will never be deleted before p when sweeping along one of these dimensions. If p is dominated with respect to dimensions $1, \ldots, i$, then the hypervolume $\mathcal{H}[p, i]$ of the

Algorithm 2 $skipdom(q, i, L_i, r, len)$

Require: q is a point, i is the dimension, L_i is the linked list at dimension i, r is the reference point, and $len = |L_{i-1}|$

1: **if** $flag[q] \geq i$ **then**
2: $\mathcal{H}[q, i] \leftarrow \mathcal{H}[prev_i(q), i]$
3: **else**
4: $\mathcal{H}[q, i] \leftarrow H(i-1, L_{i-1}, r, len)$
5: **if** $\mathcal{H}[q, i] \leq \mathcal{H}[prev_i(q), i]$ **then**
6: $flag[q] \leftarrow i$

$(i-1)$-dimensional polytope defined by all points now in L_{i-1} (including p) will be the same as $\mathcal{H}[q, i]$, with $q = prev_i(p)$. In this case, p may be flagged as dominated in dimensions $1, \ldots, i$, and $\mathcal{H}[p, i]$ may be simply set equal to $\mathcal{H}[q, i]$ as long as no points are removed from L_i. For this reason, any point flagged lower than the current dimension must have its flag reset (to indicate non-dominated) before the first loop executes.

Algorithm 2 shows the pseudo-code of the function $skipdom(\cdot)$, which is called instead of steps 11 and 18 of Algorithm 1. Point q is flagged with the dimension value i whenever $\mathcal{H}[q, i] \leq \mathcal{H}[prev_i(q), i]$ (but note that $\mathcal{H}[q, i]$ may only be less than $\mathcal{H}[prev_i(q), i]$ due to numerical accuracy issues). Subsequently, the recursion step is skipped whenever q is found again at dimension i or lower and, in that case, $\mathcal{H}[q, i]$ is set equal to $\mathcal{H}[prev_i(q), i]$. In addition, the hypervolume indicator is updated at steps 13 and 19 of Algorithm 1 by substituting $\mathcal{H}[q, i]$ for $\mathcal{H}$.

2) Reusing previous calculations: Computation time may be reduced further by breaking out of the first loop whenever the hypervolume of the i-dimensional polytope defined by the remaining points in L_i is known to have been computed before. Algorithm 3 shows the pseudo-code for the final proposal. Let the value of the hypervolume indicator of the i-dimensional polytope defined by p and all the remaining points below it along coordinate i be stored in $\mathcal{V}[p, i]$. Clearly, this value will become stale whenever points are deleted or reinserted below p. Thus, a vector of bound values b, all of which are initially set to $-\infty$, is maintained. In order to keep track of which $\mathcal{V}[p, i]$ are current, these bounds must be updated at each deletion and at each reinsertion, as performed in steps 9 and 20. Note that, again, only linear space is required.

IV. Experimental results

In order to study the impact of the various techniques proposed in the previous section on the global performance of the algorithm, four different versions were implemented, as follows:

- Version 1 corresponds to Algorithm 1, but using a linear time algorithm for the two-dimensional case instead of the algorithm for the three-dimensional case described in Section III-A. Its time complexity is $O(n^{d-1})$;
- Version 2 is the same algorithm as Version 1, but with detection of dominated points and skipping of

Algorithm 3 $H(i, L_i, r, len)$

Require: i is the dimension, L_i is the linked list at dimension i, r is the reference point, and $len = |L_{i-1}|$

1: **if** $i = 3$ **then**
2: {see Section III-A.}
3: **else if** $i > 3$ **then**
4: {Reset *flag* for all points in L_i (see Section III-C.1)}
5: $hvol \leftarrow 0$
6: $p \leftarrow nil(L_i)$
7: **while** $prev_i(p) > b_i \wedge len > 1$ **do**
8: $p \leftarrow prev_i(p)$
9: $b_j \leftarrow \min\{b_j, p_j\}, \forall j < i$
10: $delete(p, i)$
11: $len \leftarrow len - 1$
12: $q \leftarrow prev_i(p)$
13: **if** $len > 1$ **then**
14: $hvol \leftarrow \mathcal{V}[prev_i(q), i] + \mathcal{H}[prev_i(q), i] \cdot (q_i - prev_i(q))$
15: $\mathcal{V}[q, i] \leftarrow hvol$
16: $skipdom(q, i, L_i, r, len)$
17: **while** $p \neq nil(L_i)$ **do**
18: $hvol \leftarrow hvol + \mathcal{H}[q, i] \cdot (p_i - q_i)$
19: $b_i \leftarrow p_i$
20: $b_j \leftarrow \min\{b_j, p_j\}, \forall j < i$
21: $reinsert(p, i)$
22: $len \leftarrow len + 1$
23: $q \leftarrow p$
24: $p \leftarrow next_i(p)$
25: $\mathcal{V}[q, i] \leftarrow hvol$
26: $skipdom(q, i, L_i, r, len)$
27: $hvol \leftarrow hvol + \mathcal{H}[q, i] \cdot (r_i - q_i)$
28: **return** *hvol*

points known to be dominated (see Section III-C.1). This version should be roughly comparable to HSO as described in [9], and its time complexity is still $O(n^{d-1})$;

- Version 3 corresponds to Algorithm 3, but still using the linear time algorithm for the two-dimensional case. Since it reuses previous calculations where possible, as described in Section III-C.2, it should improve upon Version 2. However, its worst-case time complexity is still $O(n^{d-1})$;
- Version 4 is a full implementation of Algorithm 3. Its time-complexity is $O(n^{d-2} \log n)$;

The code was written in C and compiled using GCC 3.3.5 with optimization level 3. Source code is available at `http://sbe.napier.ac.uk/~manuel/hypervolume`. The program was run under Debian GNU/Linux on an Intel Pentium IV (Prescott) 3.2 GHz processor with 1MB of cache.

Experiments were run on two different families of non-dominated point set instances, namely the randomly generated sets and the spherical fronts produced and made available[1] by While et al. [9]. Since both HSO and the proposed algorithm are sensitive to the order in which objectives are considered, data was presented to the algorithm in such a way that objectives were processed exactly in the same order as in [9]. Likewise, the randomly generated fronts were transformed so as to correspond to a minimization problem.

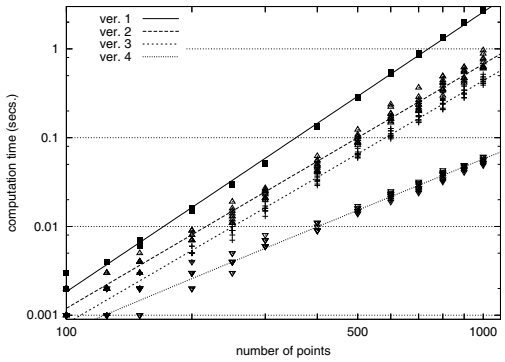

Fig. 2. Results for random $d = 4$.

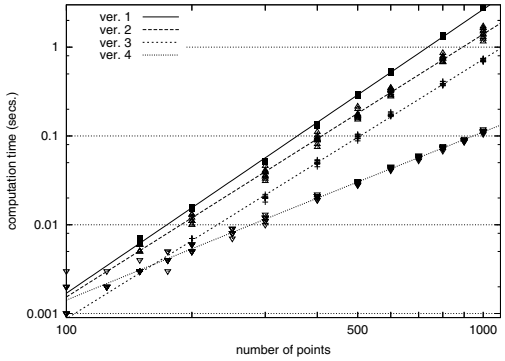

Fig. 3. Results for spherical $d = 4$.

Figures 2 to 11 show the measured CPU times *vs.* the number of points for the four algorithm versions tested. For each combination of family, number of objectives, and size of the set, the algorithms were run once on each of the 10 different instances available. The number of objectives and size of the sets were selected so as to keep the maximum individual run times below 1000 seconds.

To study the effect of the improvements introduced in the actual asymptotic behaviour of the algorithms, a linear least-squares fit of the theoretical complexity bounds to the run times t achieved by each algorithm on each set of instances was performed. For Versions 1 to 3, the regression model was simply

$$\log_{10} t = \alpha \log_{10} n + \log_{10} c$$

corresponding to a theoretical complexity of the form $t = c \cdot n^{\alpha}$. In the case of Version 4, the model used was

$$\log_{10}(t/\log_2 n) = \alpha \log_{10} n + \log_{10} c_4$$

to account for the additional $\log n$ in the theoretical time complexity of this version. Base 10 was chosen for the

[1] `http://wfg.csse.uwa.edu.au/Hypervolume`

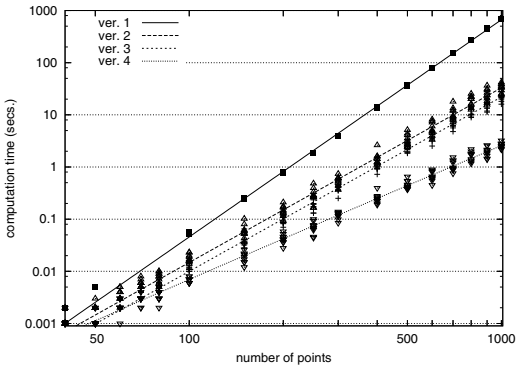

Fig. 4. Results for random $d = 5$.

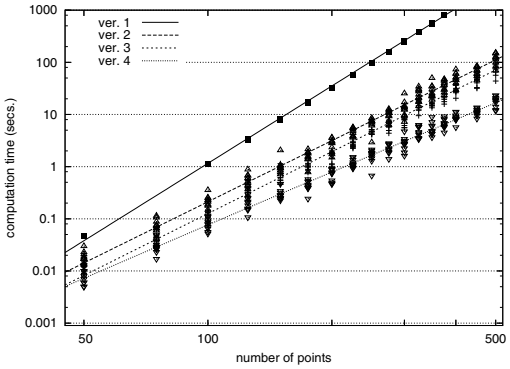

Fig. 5. Results for random $d = 6$.

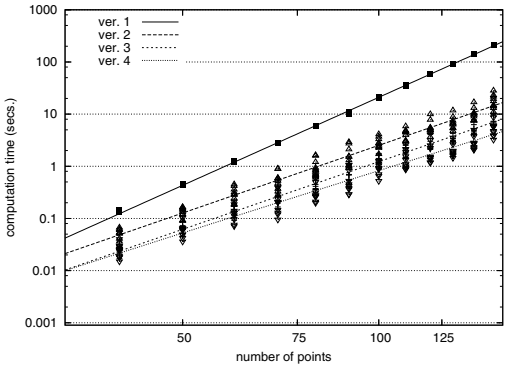

Fig. 6. Results for random $d = 7$.

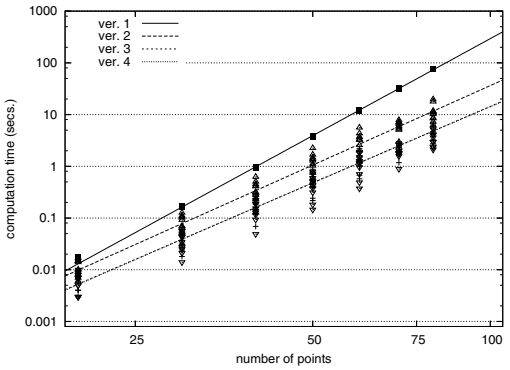

Fig. 7. Results for random $d = 8$.

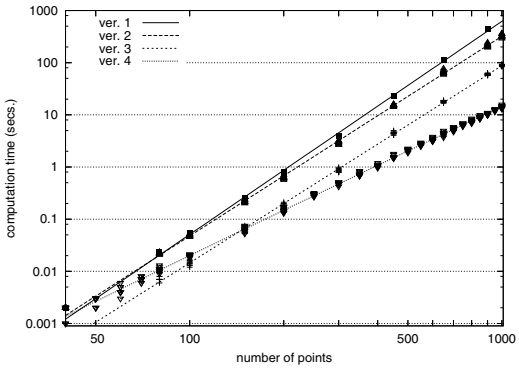

Fig. 8. Results for spherical $d = 5$.

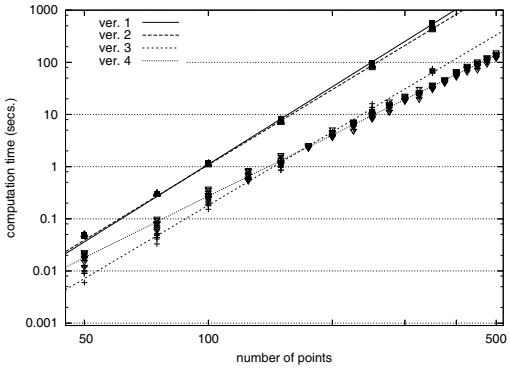

Fig. 9. Results for spherical $d = 6$.

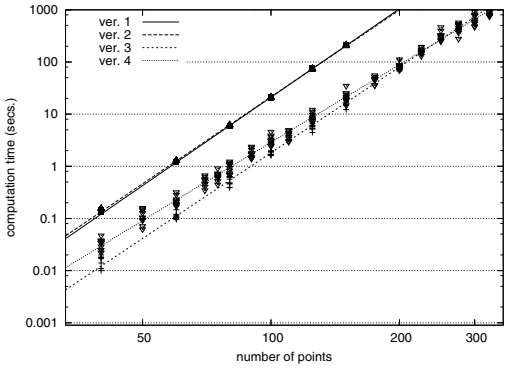

Fig. 10. Results for spherical $d = 7$.

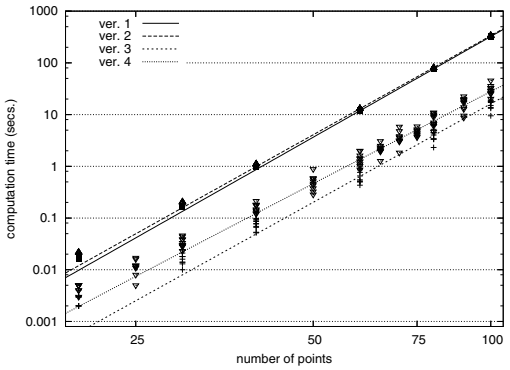

Fig. 11. Results for spherical $d = 8$.

TABLE I

CURVE-FITTING COEFFICIENTS FOR THE RANDOMLY GENERATED INSTANCES.

d	ver. 1		ver. 2		ver. 3		ver. 4		min.
	α	$\log_{10} c$	α	$\log_{10} c$	α	$\log_{10} c$	α	$\log_{10} c_4$	size
4	3.15	-9.03	2.75	-8.42	2.74	-8.56	1.76	-7.53	100
5	4.16	-9.67	3.35	-8.53	3.34	-8.69	2.39	-7.76	50
6	4.93	-9.80	3.89	-8.46	3.94	-8.78	3.16	-8.27	50
7	5.60	-9.89	4.31	-8.22	4.31	-8.53	3.73	-8.37	50
8	6.23	-9.99	5.10	-8.64	4.91	-8.66	4.65	-8.97	30

TABLE II

CURVE-FITTING COEFFICIENTS FOR THE SPHERICAL FRONT INSTANCES.

d	ver. 1		ver. 2		ver. 3		ver. 4		min.
	α	$\log_{10} c$	α	$\log_{10} c$	α	$\log_{10} c$	α	$\log_{10} c_4$	size
4	3.20	-9.17	2.96	-8.74	2.92	-8.90	1.72	-7.11	100
5	4.09	-9.46	3.83	-8.98	3.78	-9.40	2.68	-7.88	50
6	4.93	-9.83	4.79	-9.55	4.67	-10.08	3.69	-8.76	50
7	5.63	-9.94	5.54	-9.74	5.46	-10.66	4.75	-9.86	50
8	6.46	-10.42	6.36	-10.20	6.32	-11.44	5.68	-10.73	30

"outer" logarithms to make it easier to perceive the actual values of c and c_4 from their logarithms. On the other hand, the choice of base 2 for the $\log n$ factor in the complexity bound of Version 4 reflects this algorithm's use of a height-balanced binary tree. Since the regression models adopted refer to asymptotic complexity, only instances larger than a given number of points were used for fitting, depending on the number of objectives. Tables I and II show the estimated coefficients for these models, as well as the minimum instance size used in the regression. The curve fits obtained are shown on the plots together with the observed data.

It is worth noting that the theoretical exponent of the complexity of Version 1 is recovered from the data for dimensions 4–6, and that, for dimensions 7 and 8, the results are in fact better than expected. This may indicate that the sizes of the instances are just too small for the true asymptotic behaviour to be observed, for example, due to the cache of the processor. Note that the algorithms implicitly assume a flat memory model and do not attempt to exploit any memory hierarchy.

Interestingly, the speedup introduced by Version 2 is achieved in terms of an actual reduction of the complexity exponent, at the expense of slightly greater overhead (greater value of c). Version 3 does not generally improve the exponent any further, but does recover some of the overhead introduced in Version 2. Finally, Version 4 generally reduces the exponent by about 1, as expected, although it does introduce considerably greater overhead, as it is evident from the plots. Although this is not exactly unusual, it should be possible to improve the special case implementation in order to avoid repeating calculations as was done in the recursive part of the algorithm.

Since While's HSO implementation is not available at the time of this study, it is only possible to compare the experimental results obtained to those published in [9] by correcting for differences in processor speed. Clearly, such a comparison disregards many important differences between the two experimental studies, including different implementation languages, but should still serve to rule out any major differences between the two sets of results. In Table III, some results extracted from [9] are given, and compared to the results obtained in this study with algorithm Versions 1 to 4. Column HSO presents the actual CPU times reported in that study for various combinations of number of dimensions (column d) and number of points (column points); column HSO* presents the same CPU times scaled to reflect the different processor speeds (HSO* $= 1.9/3.2 \cdot$HSO). The remaining columns show the corresponding CPU times estimated from the curve fittings described above. It can be seen that Version 2 compares well to HSO. Version 3 is faster and, for large sets, Version 4 is much faster.

TABLE III

COMPARISON WITH HSO FOR A GIVEN NUMBER OF POINTS AND DIMENSIONS; SEE TEXT FOR MORE DETAILS.

d	points	HSO	HSO*	ver. 1	ver. 2	ver. 3	ver. 4
4	750	1.0	0.6	1.0	0.3	0.2	0.1
5	200	1.0	0.6	0.8	0.2	0.1	0.1
	1300	100	60	2000	80	53	5
6	80	1.0	0.6	0.4	0.1	0.05	0.04
	340	100	60	480	25	16	5
7	45	1.0	0.6	0.25	0.08	0.04	0.03
	145	100	60	168	13	6	4
8	30	1.0	0.6	0.16	0.08	0.04	0.04
	80	100	60	73	12	5	5

V. CONCLUSIONS AND FURTHER WORK

This article presented a dimension-sweep algorithm to compute the hypervolume indicator. Although it implements the same principle as the HSO algorithm proposed by While et al. [9], it is based on a data structure which facilitates

not only dimension-sweeping but also the implementation of some additional speed-ups without sacrificing linear space complexity. In addition, this algorithm exploits the solution for the three-dimensional case proposed in [13], which reduces the overall time complexity by a factor of $n/\log n$.

The asymptotic behaviour of the algorithm with respect to CPU time was experimentally derived from runs on two types of benchmark data instances. The speed-ups proposed were found to cause a reduction of the complexity exponent at the expense of slightly greater overhead. In addition, the experimental results indicate that the full algorithm proposed clearly outperforms HSO as the number of points grows, although there should still be some room for improving the integration of the three-dimensional case code with the recursive part of the algorithm.

Although not examined here, the permutation heuristics developed for HSO in [15] should also have a positive effect on the performance of the algorithms tested. Another promising direction for further work consists of developing a special case of Overmars and Yap's algorithm [12] specifically for the computation of the hypervolume indicator. Indeed, even in its generic form, that algorithm already guarantees strictly lower time complexity than other alternatives for computing the hypervolume indicator in $d > 4$ dimensions.

Acknowledgement

Carlos M. Fonseca acknowledges funding from the Portuguese Foundation for Science and Technology under grant POCTI/EME/48448/2002, with the support of FEDER and the Portuguese State. Manuel López-Ibáñez wishes to acknowledge the Faculty of Economics of the University of Algarve for receiving him as a visitor in December 2005 in connection with this work.

REFERENCES

[1] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

[2] J. Knowles, D. Corne, and M. Fleischer, "Bounded archiving using the lebesgue measure," in *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'03)*, R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, Eds. IEEE Press, Piscataway, NJ, 2003, pp. 2490–2498.

[3] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *Parallel Problem Solving from Nature (PPSN VIII)*, X. Yao *et al.*, Eds. Berlin, Germany: Springer-Verlag, 2004, pp. 832–842.

[4] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms – A comparative case study," in *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*. Springer Verlag, Berlin, Germany, 1998, pp. 292–301.

[5] J. Wu and S. Azam, "Metrics for quality assessment of a multiobjective design optimization solution set," *Journal of Mechanical Design*, vol. 123, no. 1, pp. 18–25, 2001.

[6] R. P. Stanley, *Enumerative Combinatorics*. Cambridge University Press, 1999, vol. 1.

[7] M. Fleischer, "The measure of Pareto optima: Applications to multi-objective metaheuristics," in *Evolutionary Multi-criterion Optimization (EMO 2003)*, ser. Lecture Notes in Computer Science, C. M. Fonseca, P. Fleming, E. Zitzler, K. Deb, and L. Thiele, Eds., vol. 2632. Springer Verlag, Berlin, Germany, 2003, pp. 519–533.

[8] L. While, "A new analysis of the lebmeasure algorithm for calculating hypervolume," in *Evolutionary Multi-criterion Optimization (EMO 2005)*, ser. Lecture Notes in Computer Science, C. A. C. Coello, A. H. Aguirre, and E. Zitzler, Eds., vol. 3410. Springer Verlag, Berlin, Germany, 2005, pp. 326–340.

[9] L. While, P. Hingston, L. Barone, and S. Husband, "A faster algorithm for calculating hypervolume," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 1, pp. 29–38, 2006.

[10] F. P. Preparata and M. I. Shamos, *Computational Geometry*. Springer Verlag, 1985.

[11] J. Leeuwen and D. Wood, "The measure problem for rectangular ranges in d-space," *Journal of Algorithms*, vol. 2, no. 3, pp. 282–300, 1981.

[12] M. H. Overmars and C. K. Yap, "New upper bounds in Klee's measure problem," *SIAM Journal on Computing*, vol. 20, no. 6, pp. 1034–1045, 1991.

[13] L. Paquete, C. M. Fonseca, and M. López-Ibáñez, "An optimal algorithm for a special case of Klee's measure problem in three dimensions," Centre for Intelligent Systems, University of Algarve, Faro, Portugal, Tech. Rep. CSI-RT-I-01/2006, 2006.

[14] C. M. Fonseca, "Output-sensitive computation of the multivariate ECDF and related problems," in *Proceedings of the Conference COMPSTAT 2002*, S. Klinke, P. Ahrend, and L. Richter, Eds., vol. 2002, 2002, p. 30.

[15] L. While, L. Bradstreet, L. Baroneand, and P. Hingston, "Heuristics for optimising the calculation of the hypervolume for multi-objective optimisation problems," in *Proceedings of the 2005 Congress on Evolutionary Computation (CEC'05)*, vol. 3. IEEE, September 2005, pp. 2225–2232.