A Faster Algorithm for Calculating Hypervolume

Lyndon While, *Senior Member, IEEE*, Phil Hingston, *Member, IEEE*, Luigi Barone, *Member, IEEE*, and Simon Huband, *Member, IEEE*

***Abstract*—We present an algorithm for calculating hypervolume exactly, the Hypervolume by Slicing Objectives (HSO) algorithm, that is faster than any that has previously been published. HSO processes objectives instead of points, an idea that has been considered before but that has never been properly evaluated in the literature. We show that both previously studied exact hypervolume algorithms are exponential in at least the number of objectives and that although HSO is also exponential in the number of objectives in the worst case, it runs in significantly less time, i.e., two to three orders of magnitude less for randomly generated and benchmark data in three to eight objectives. Thus, HSO increases the utility of hypervolume, both as a metric for general optimization algorithms and as a diversity mechanism for evolutionary algorithms.**

***Index Terms*—Evolutionary computation, hypervolume, multiobjective optimization, performance metrics.**

I. Introduction

Multiobjective optimization problems abound, and many evolutionary algorithms have been proposed to derive good solutions for such problems, for example [1]–[8]. However, the question of what metrics to use in comparing the performance of these algorithms remains difficult [1], [9], [10]. One metric that has been favored by many people is *hypervolume* [1], [11], also known as the S metric [12] or the Lebesgue measure [13], [14]. The hypervolume of a set of solutions measures the size of the portion of objective space that is dominated by those solutions collectively. Generally, hypervolume is favored because it captures in a single scalar both the closeness of the solutions to the optimal set and, to some extent, the spread of the solutions across objective space. Hypervolume also has nicer mathematical properties than many other metrics; Zitzler *et al.* [15] state that hypervolume is the only unary metric of which they are aware that is capable of detecting that a set of solutions X is not worse than another set X', and Fleischer [16] has proved that hypervolume is maximized if and only if the set of solutions contains only Pareto optima. Hypervolume has some nonideal properties too; it is sensitive to the relative scaling of the objectives and to the presence or absence of extremal points in a front.

Hypervolume has also been proposed as a diversity mechanism in multiobjective algorithms [1], [16], [17], for ensuring that an algorithm generates a front with a good spread over the range of optimal solutions. Clearly, if hypervolume calculations are incorporated into the execution of an algorithm (as opposed to hypervolume used as a metric after execution is completed), there is a much stronger requirement for those calculations to be efficient. The ideal for such use is an incremental algorithm that minimizes the expense of repeated invocations.

However, previously studied algorithms for calculating hypervolume exactly are very expensive, probably too expensive to facilitate the use of hypervolume for problems with more than two to three objectives. In particular, the published complexity of the new algorithm LebMeasure [14], [16] is incorrect. We show in this paper that although LebMeasure is polynomial in the number of points, in the worst case it is exponential in the number of objectives, not polynomial as previously claimed, and that the general performance of LebMeasure deteriorates rapidly as the number of objectives increases beyond two to three.

The principal contribution of this paper is a faster algorithm for calculating hypervolume, called the Hypervolume by Slicing Objectives (HSO) algorithm. HSO processes *objectives*, as opposed to *points*, repeatedly making slices through a hypervolume in fewer and fewer objectives, then sums the volumes of these slices to calculate the total hypervolume. This idea has been suggested independently before [18], [19], but it has never been seriously studied, possibly because of the (incorrectly) perceived efficiency of LebMeasure. Indeed, in the worst case, HSO still exhibits exponential complexity, but this complexity is much lower than that of LebMeasure. Moreover, we show that HSO is significantly faster than LebMeasure, by two to three orders of magnitude over typical fronts in three to eight objectives. HSO thus extends the utility of hypervolume to problems with more objectives and allows the evaluation of much bigger fronts for such problems.

The rest of the paper is structured as follows. Section II defines the concepts and notation used in multiobjective optimization and throughout this paper. Section III describes previous published algorithms that calculate hypervolume, with particular emphasis on Fleischer's LebMeasure algorithm. Section IV describes the HSO algorithm. Section V describes our experimental setup and compares the performance of HSO and LebMeasure. Section VI concludes the paper and outlines some ideas for future work in this area.

II. Definitions

In a multiobjective optimization problem, we aim to find the set of optimal tradeoff solutions known as the Pareto optimal set. Pareto optimality is defined with respect to the concept of nondomination between points in objective space. Without loss of generality, consider a maximization problem in n objectives

Manuscript received June 7, 2004; revised December 13, 2004 and February 17, 2005. This work was supported in part by the Australian Research Council.

L. While and L. Barone are with the School of Computer Science and Software Engineering, The University of Western Australia, Western Australia 6009, Australia (e-mail: lyndon@csse.uwa.edu.au; luigi@csse.uwa.edu.au).

P. Hingston and S. Huband are with the School of Computer and Information Sciences, Edith Cowan University, Western Australia 6050, Australia (e-mail: p.hingston@ecu.edu.au; s.huband@ecu.edu.au).

Digital Object Identifier 10.1109/TEVC.2005.851275

for which we generate a set X containing m solutions. We assume that all values in all objectives are positive.

Given two objective vectors $\overline{x}$ and $\overline{y}$, $\overline{x}$ *dominates* $\overline{y}$ if and only if $\overline{x}$ is at least as good as $\overline{y}$ in all objectives and better in at least one. A vector $\overline{x}$ is *nondominated* with respect to the set X if and only if there is no vector in X that dominates $\overline{x}$. X is a *nondominated set* if and only if all vectors in X are mutually nondominating. Such a set of objective vectors is sometimes called a *nondominated front*.

A vector $\overline{x}$ is *Pareto optimal* if and only if $\overline{x}$ is nondominated with respect to the set of all possible vectors. Pareto optimal vectors are characterized by the fact that improvement in any one objective means worsening at least one other objective. The *Pareto optimal set* is the set of all possible Pareto optimal vectors. The goal in a multiobjective problem is to find the Pareto optimal set, although for continuous problems a representative subset will usually suffice.

Given the set X of solutions produced by an algorithm, the question arises how good the set X is, i.e., how well it approximates the Pareto optimal set. One metric used for comparing sets of solutions is to measure the *hypervolume* of each set. The hypervolume of X is the total size of the space that is dominated by one (or more) of the solutions in X. The hypervolume of a set is measured relative to a reference point, the anti-optimal point or "worst possible" point in space; for our maximization problem with positive objectives, we take the origin as the reference point. If a set X has a greater hypervolume than a set X', then X is taken to be a better set of solutions than X'.

III. Previous Algorithms

Calculating the hypervolume of a Pareto front is expensive, and few algorithms have been published. In this section, we describe two previous exact algorithms, then we describe briefly an algorithm for approximating hypervolume, then we consider the general question of what level of performance is required from a hypervolume algorithm for it to be usable for practical applications.

A. Inclusion–Exclusion Algorithm

The most obvious algorithm for calculating hypervolume is the inclusion–exclusion algorithm (IEA) [10]. Fig. 1 gives the pseudocode for the IEA.

The IEA works exactly like the standard algorithm for calculating the size of the union of a set of sets. It sums the volumes dominated by each point in the set individually, then subtracts the volumes dominated by the intersection of each pair of points, then adds back in the volumes dominated by the intersection of each triple of points, and so on, until all subsets of the original set have been accounted for. It is clear from this description that the IEA calculates the volume of every subset of a set of m points. Each calculation is at least $O(n)$, so the complexity of the IEA is at least $O(n2^m)$. Thus, the IEA is exponential in the number of points; hence, it is unusable for practical applications.

B. LebMeasure Algorithm

Possibly the best known algorithm for calculating hypervolume is the LebMeasure algorithm, due to Fleischer [14].

```
ie (ps):
  vol = 0
  for each non-empty subset s of ps
    vol = vol + intersection (s) * (-1)^(|s|+1)
  return vol

intersection (s) returns the volume of the
largest hyper-cuboid that is dominated by
all members in the set of points s
```

Fig. 1. Pseudocode for inclusion–exclusion algorithm.

LebMeasure works by processing the points in a set one at a time. It calculates the volume that is dominated exclusively by one point then discards that point and moves on to the subsequent points until all points have been processed and all volumes have been summed. This is particularly efficient when the volume dominated exclusively by a point p is "hypercuboid," but where this is not the case, LebMeasure lops off a hypercuboid that is dominated exclusively by p and replaces p with a set of "spawned" points that dominate the remainder of p's exclusive hypervolume. Spawns which dominate no exclusive hypervolume, either because they have one or more zero dimensions or because they are dominated by an unprocessed point, are discarded.

As an example, consider the four three-objective points

$$(6, 7, 4), (9, 5, 5), (1, 9, 3), (4, 1, 9).$$

(6, 7, 4) dominates exclusively the hypercuboid, which is bounded at the opposite corner by (4, 5, 3). Thus, the three potential spawns of (6, 7, 4) are

$$(4, 7, 4), (6, 5, 4), (6, 7, 3).$$

However, (6, 5, 4) is dominated by (9, 5, 5) (from the main list of points), so only the other two spawns dominate exclusive hypervolume of their own, and only those two are added to the main list to be processed.

LebMeasure continues lopping off volume and replacing points with their spawns until all points (and spawns) have been processed.

Fleischer claims that the worst case complexity of LebMeasure is $O(n^2m^{3)}$ [14], [16]. The calculation is based partly on the observation that for spawned points, many of their potential spawns dominate no exclusive hypervolume. The potential spawns are, in fact, guaranteed to be dominated by some other point in the remaining set. However, a simple example illustrates that this fact is not sufficient to make the LebMeasure polynomial in the number of points. It is clear that the complexity of LebMeasure is at least equal to the number of hypercuboids that are processed and summed (including spawns, spawns of spawns, etc.). Consider the sets of points described by the pattern shown in Fig. 2.

By tracing the detailed operation of LebMeasure using the code given in Fig. 4, we can see that the numbers of hypercuboids that are processed for this pattern are as given in Table I. The numbers clearly show growth that is exponential in the number of objectives, indicating that this pattern generates m^{n-1} hypercuboids. Given the operations involved in processing each point in LebMeasure (principally generating

1	5	···	5
2	4	···	4
3	3	···	3
4	2	···	2
5	1	···	1

Fig. 2. Pathological example for LebMeasure. Pattern describes sets of five points in n objectives, $n \geq 2$. All columns except first are identical. Pattern can be generalized for other numbers of points.

TABLE I
NUMBERS OF HYPERCUBOIDS PROCESSED FOR PATTERN FROM FIG. 2, EQUAL TO m^{n-1}

n	$m = 2$	$m = 5$	$m = 8$	$m = 10$
2	2	5	8	10
3	4	25	64	100
4	8	125	512	1,000
5	16	625	4,096	10,000
6	32	3,125	32,768	100,000
7	64	15,625	262,144	1,000,000
8	128	78,125	2,097,152	10,000,000
9	256	390,625	16,777,216	100,000,000

1	1	2	3	4	5	1	···	5
2	2	3	4	5	1	2	···	4
3	3	4	5	1	2	3	···	3
4	4	5	1	2	3	4	···	2
5	5	1	2	3	4	5	···	1

Fig. 3. Second pathological example for LebMeasure. Pattern describes sets of five points in n objectives, $n \geq 2$. First and last columns are fixed; if $n > 2$, second column is fixed and each other column is a simple rotation of the previous column. Pattern can be generalized for other numbers of points.

TABLE II
BEST CASE NUMBERS OF HYPERCUBOIDS PROCESSED FOR PATTERN FROM FIG. 3, APPROXIMATELY EQUAL TO (BUT NOT BOUNDED BY) $m(m!)^{(n-2) div\ m}((n-2) mod\ m)!$. VALUES OF m REPORTED WERE LIMITED BY NUMBERS OF PERMUTATIONS THAT MUST BE TESTED

n	$m = 2$	$m = 3$	$m = 4$	$m = 5$
2	2	3	4	5
3	2	3	4	5
4	4	6	8	10
5	4	17	23	29
6	8	17	88	112
7	8	35	88	549
8	16	105	180	549
9	16	105	558	1,115
10	32	213	2,248	3,421
11	32	641	2,248	14,083
12	64	641	4,528	70,889
13	64	1,289	13,708	70,889
14	128	3,873	54,976	142,309
15	128	3,873	54,976	428,449
16	256	7,761	110,160	1,721,605
17	256	23,297	331,128	8,618,577

TABLE III
NUMBERS OF HYPERCUBOIDS PROCESSED FOR FIRST POINT OF PATTERN FROM FIG. 2, EQUAL TO $m^{n-1} - (m-1)^{n-1}$, i.e., $O(m^{n-2})$

n	$m = 2$	$m = 5$	$m = 8$	$m = 10$
2	1	1	1	1
3	3	9	15	19
4	7	61	169	271
5	15	369	1,695	3,439
6	31	2,101	15,961	40,951
7	63	11,529	144,495	468,559
8	127	61,741	1,273,609	5,217,031
9	255	325,089	11,012,415	56,953,279

potential spawns and checking them for dominance), we estimate its worst case complexity to be $O(n^2m^n)$.

However, this pattern raises the question of the order in which the points are processed. If the points from Fig. 2 are reversed before being presented to LebMeasure, then the number of hypercuboids is m, for all values of n. The order shown in Fig. 2 is, in fact, the worst case for this pattern.

We shall return to the question of ordering points when we evaluate the general performance of LebMeasure in Section V, but we illustrate now that there are patterns of points that exhibit exponential growth under LebMeasure even in their best case ordering. Consider the sets of points described by the pattern shown in Fig. 3.

By evaluating all $m!$ permutations of each set of points, we can see that the numbers of hypercuboids that are processed for this pattern when the points are presented in their optimal ordering are as given in Table II. Again, we see exponential growth; this pattern generates approximately $m(m!)^{(n-2) div\ m}((n-2) mod\ m)!$ hypercuboids, even for the best case ordering of the points.

Finally, Knowles *et al.* propose the use of LebMeasure in an evolutionary algorithm that calculates hypervolume incrementally for archiving purposes [17]. Structurally, LebMeasure is well suited to this task, because it calculates explicitly the hypervolume dominated exclusively by its first point. However, even evaluating the hypervolume dominated by a single point can be exponential in LebMeasure, as shown by the numbers in Table III. Note that in this context, where we need to evaluate the hypervolume contribution of a single point, changing the order of the other points makes no difference to the algorithm's performance.

In summary, it is clear that the worst case complexity of LebMeasure is at least $O(m^n)$, and while reordering points can help somewhat, we shall show in Section V that even with reordering, the usefulness of LebMeasure deteriorates rapidly as the number of objectives increases.

C. *Algorithm for Approximating Hypervolume*

Given that hypervolume is so expensive to calculate exactly, a valid approach would be to design an algorithm to approximate it to within a reasonable error. Everson *et al.* [20] describe a Monte Carlo algorithm for comparing two nondominated sets X and X' by calculating the fraction of space dominated by X and not by X', and vice versa. They do this by normalizing all objectives onto the range $[0 \ldots 1]$ then randomly generating points in the unit hypercube and testing each point to see if it is dominated by X and/or X'. This algorithm has complexity $O(nmN)$ in the worst case, where N is the number of samples used, and this complexity can be reduced by using a more advanced data structure to store the points. The error induced by the random sampling decreases as $\sqrt{N}$, and the authors state that it is independent of n.

The Monte Carlo approach could clearly be adapted to calculate hypervolume directly, and an approximation might be sufficient for many purposes. However, an efficient exact algorithm

```
hypercuboids (ps):
  pl = a list containing ps in some order
  hypercuboids = 0
  while pl /= []
    p = head (pl)
    pl = tail (pl)
    ql = spawns (p, corner (p, pl), pl)
    prepend ql to pl
    hypercuboids = hypercuboids + 1
  return hypercuboids

corner (p, pl):
  a = refPoint
  while pl /= []
    p' = head (pl)
    pl = tail (pl)
    for k = 1 to n
      if p[k] beats p'[k] beats a[k]
        a[k] = p'[k]
  return a

spawns (p, a, pl):
  ql = []
  for k = 1 to n
    if a[k] /= refPoint[k]
      q = p
      q[k] = a[k]
      if not (dominated (q, pl))
        append q to ql
  return ql

dominated (p, pl) returns True iff the point p
is dominated by at least one point on pl
```

Fig. 4. Pseudocode for counting hypercuboids processed in LebMeasure. Description does not include trick for automatically rejecting spawns which are known to be dominated: however, trick is irrelevant here, as we are only counting number of contributing points.

would have clear advantages; it would always give precise results for comparing fronts, and it would be more amenable to mathematical analysis.

D. *Is Hypervolume Too Expensive to be Useful?*

Given that both previously studied exact algorithms have complexity that is exponential in at least the number of objectives, it may be thought that hypervolume is not, after all, a practical idea as a metric for comparing fronts. However, in reality, the number of objectives is seldom large. A survey of papers from the proceedings of the CEC 2003 reveals that the great majority of authors who considered more than two objectives studied problems with only three or four objectives, and none studied problems with more than 12 objectives. Thus, we might still hope to make use of hypervolume, given a (relatively!) efficient exponential algorithm.

A related question is: "what population size is required for a problem in n objectives, for an evolutionary algorithm to derive good solutions?". It is to be expected that population size will increase with n, in order to achieve reasonable coverage of the Pareto optimal front (see for example [21] and [22]). Some theoretical guidance is given in [23], where it is shown that, for their adaptive grid archiving strategy, a guarantee of a weak kind of convergence requires an archive size that is exponential in the number of objectives. However, again in practice, we find that authors tend to limit their population size in order to get reasonable performance out of their algorithm. According to [24], population sizes for real-world problems tend to lie in the range 10–1000. Thus, in Section V we follow Purshouse [11] and Khare *et al.* [22] and use this range of population sizes to evaluate algorithms.

IV. HSO Algorithm

Given m mutually nondominating points in n objectives, the HSO algorithm is based on the idea of processing the points *one objective at a time*. This idea has been suggested independently before [18], [19]. But, in this paper we apply the idea differently to reduce repeated calculations, and we report a thorough theoretical and empirical analysis of both the complexity and performance of the algorithm. We discuss HSO as it is applied to a maximization problem relative to the origin. The transformation to a minimization problem, or to another reference point, is trivial. (We do not address here the problem of choosing a reference point, if the anti-optimal point is not known or does not exist. One suggestion is to take, in each objective, the worst value from any of the fronts being compared.)

Initially, the points are sorted by their values in the first objective. These values are then used to cut cross-sectional "slices" through the hypervolume; each slice will itself be an $n-1$-objective hypervolume in the remaining objectives. The $n-1$-objective hypervolume of each slice is calculated and is multiplied by the depth of the slice in the first objective, then these n-objective values are summed to obtain the total hypervolume.

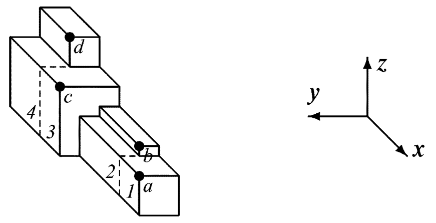
Each slice through the hypervolume will contain a different subset of the original points. Because the points are sorted, they can be allocated to the slices easily. The top slice can contain only the point with the highest value in the first objective; the second slice can contain only the points with the two highest values; the third slice can contain only the points with the three highest values; and so on, until the bottom slice, which can contain all of the points. However, not all points "contained" by a slice will contribute volume to that slice. Some points may be dominated in whatever objectives remain and will contribute nothing. After each step (i.e., after each slicing action), the number of objectives is reduced by one, the points are resorted in the next objective, and newly dominated points within each slice are discarded.

Fig. 5 shows the operation of one step in HSO, including the slicing of the hypervolume, the allocation of points to each slice, and the elimination of newly dominated points.

The most natural base case for HSO is when the points are reduced to one objective, when there can be only one nondominated point left in each slice. The value of this point is then the one-objective hypervolume of its slice. However, in practice, for efficiency reasons, HSO terminates when the points are reduced to two objectives, which is an easy and fast special case.

A. *Operational Behavior of HSO*

Fig. 6 gives the pseudocode for HSO. The *modus operandi* is to start from a set containing a single list S of m points in n objectives with (nominal) depth one. The algorithm performs $n-1$ steps. In the first step, S is expanded into a set containing m lists of points in $n-1$ objectives, each paired with the depth of the corresponding slice in the first objective. In the second step, each of these lists is expanded into a set of lists in $n-2$ objectives,



	x	y	z
a	11	4	4
b	9	2	5
c	5	6	7
d	3	3	10

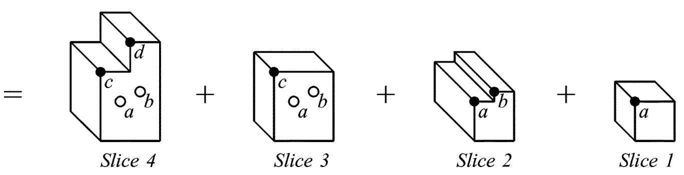


Fig. 5. One step in HSO for four three-objective points shown. Objective x is processed, leaving four two-objective shapes in y and z. Points are labeled with letters. Unfilled circles represent points that are dominated in y and z. Slices are labeled with numbers and are separated on main picture by dashed lines.

each paired with the product of its depths in the first two objectives. These sets of lists are combined to again make a single set. This expansion occurs $n-1$ times, until we have a large number of lists in one objective, each paired with the product of its depths in the first $n-1$ objectives. The volumes represented by these lists are summed to give the total hypervolume.

The three functions from Fig. 6 perform the following tasks.

- The function `slice` takes a list of points `pl` in Objectives $k \ldots n$ and returns a set containing the lists of points derived from `pl` in Objectives $k+1 \ldots n$, each paired with the depth of the corresponding slice. `slice` essentially performs the actions depicted in Fig. 5.
- The function `insert` accumulates incrementally the list of points in each slice. `insert` maintains the invariant that all lists of points are sorted in the next relevant objective. It also eliminates dominated points.
- The function `hso` iterates $n-1$ times, starting from the original list of points with a depth of one. In each step, each list of points is broken down in the next objective, and the depths are accumulated multiplicatively. The effect is to reduce the original list of points in Objectives $1 \ldots n$ to a large number of (singleton) lists of points in Objective n, each paired with its cumulative depth. `hso` sums these one-objective hypervolumes to calculate the total hypervolume.

As mentioned previously, the performance of HSO can be improved by using $n = 2$ as the base case, instead of $n = 1$. A further improvement comes from processing each intermediate list of points on-the-fly, instead of constructing an explicit set.

B. Complexity of HSO

We can construct a recurrence relation describing the worst case complexity of HSO by considering the operation of each step. Each step starts with a list of m points in n objectives. In the worst case, m points with different values in the first objective will generate m distinct slices through the hypervolume. One of these slices (the top one) will contain one $n-1$ objective point; the next slice will contain two points; and so on, until the bottom slice, which will contain all m points. Each of these

```
hso (ps):
  pl = sort ps worsening in Objective 1
  s = {(1, pl)}
  for k = 1 to n-1
    s' = {}
    for each (x, ql) in s
      for each (x', ql') in slice (ql, k)
        add (x * x', ql') into s'
    s = s'
  vol = 0
  for each (x, ql) in s
    vol = vol + x * |head (ql)[n] - refPoint[n]|
  return vol

slice (pl, k):
  p = head (pl)
  pl = tail (pl)
  ql = []
  s = {}
  while pl /= []
    ql = insert (p, k+1, ql)
    p' = head (pl)
    add (|p[k] - p'[k]|, ql) into s
    p = p'
    pl = tail (pl)
  ql = insert (p, k+1, ql)
  add (|p[k] - refPoint[k]|, ql) into s
  return s

insert (p, k, pl):
  ql = []
  while pl /= [] && head (pl)[k] beats p[k]
    append head (pl) to ql
    pl = tail (pl)
  append p to ql
  while pl /= []
    if not (dominates (p, head (pl), k))
      append head (pl) to ql
    pl = tail (pl)
  return ql

dominates (p, q, k) returns True iff the point p
dominates the point q in Objectives k..n
```

Fig. 6. Pseudocode for HSO.

slices must be processed separately, so the total amount of work involved is given by

$$f(m, 1) = 1 \tag{1}$$

and

$$f(m, n) = \sum_{k=1}^{m} f(k, n-1). \tag{2}$$

Note that, typically, some points will be dominated and discarded at each step, reducing the sizes of some sets and speeding up the calculation; but for the worst case analysis, we assume that this never happens.

The base case is given by (1). When only one objective remains, there will be only one point left, whose single value is the result. f thus counts how many one-objective "volumes" are summed by HSO.

From this recurrence relation, we can show that

$$f(m, n) = \binom{m+n-2}{n-1}.$$

We first need a lemma.

Lemma 1: For all nonnegative integers x, y, r such that $y \leq x$ and $y \leq x - r$, the following identity holds:

$$\binom{x}{y} = \binom{x-r}{y} + \sum_{i=1}^{r} \binom{x-i}{y-1}. \tag{3}$$

Proof: The proof is by induction on r.

Base case: $r = 1$. In this case, the formula becomes

$$\binom{x}{y} = \binom{x-1}{y} + \binom{x-1}{y-1}. \tag{4}$$

This is a well-known identity [25] that can be checked by expanding out the binomial coefficients.

Induction step: suppose the formula holds for $r = j$, i.e.,

$$\binom{x}{y} = \binom{x-j}{y} + \sum_{i=1}^{j} \binom{x-i}{y-1}.$$

Invoking (4), we expand the first term on the right-hand side, giving

$$\begin{aligned}\binom{x}{y} &= \binom{x-j-1}{y} + \binom{x-j-1}{y-1} + \sum_{i=1}^{j} \binom{x-i}{y-1} \\ &= \binom{x-(j+1)}{y} + \sum_{i=1}^{j+1} \binom{x-i}{y-1}.\end{aligned}$$

Hence, the formula holds for $r = j + 1$, and by induction it holds for all r, as claimed. ■

Corollary 1: For all nonnegative integers x, y such that $y \leq x$, the following identity holds:

$$\binom{x}{y} = \sum_{i=1}^{x-y+1} \binom{x-i}{y-1}.$$

Proof: Substituting $r = x - y$ in (3), we obtain

$$\begin{aligned}\binom{x}{y} &= \binom{y}{y} + \sum_{i=1}^{x-y} \binom{x-i}{y-1} \\ &= \binom{y-1}{y-1} + \sum_{i=1}^{x-y} \binom{x-i}{y-1} \\ &= \binom{x-x+y-1}{y-1} + \sum_{i=1}^{x-y} \binom{x-i}{y-1} \\ &= \binom{x-(x-y+1)}{y-1} + \sum_{i=1}^{x-y} \binom{x-i}{y-1} \\ &= \sum_{i=1}^{x-y+1} \binom{x-i}{y-1}\end{aligned}$$

as required. ■

Corollary 2: $f(m,n) = \binom{m+n-2}{n-1}$ satisfies the recurrence defined by (1) and (2).

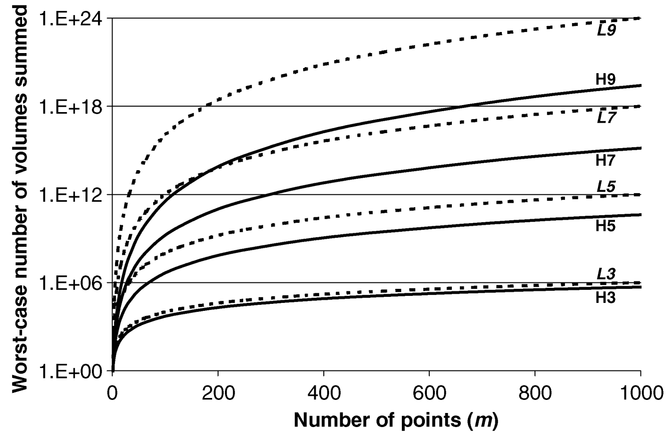


Fig. 7. Log-scale comparison of worst case complexities of HSO and LebMeasure. Each line Fn plots complexity of Algorithm F for m points in n objectives. Solid lines represent HSO and dashed lines represent LebMeasure.

Proof: Equation (1) is easily checked by substituting $n = 1$ in the definition of f. To show that (2) holds, we apply Corollary 1

$$\begin{aligned}f(m,n) &= \binom{m+n-2}{n-1} \\ &\quad \text{by Cor. 1 with } x = m+n-2,\ y = n-1 \\ &= \sum_{i=1}^{m} \binom{m+n-2-i}{n-2} \\ &\quad \text{making the substitution } k = m-i+1 \\ &= \sum_{k=m}^{1} \binom{m+n-2-(m-k+1)}{n-2} \\ &\quad \text{rearranging and simplifying} \\ &= \sum_{k=1}^{m} \binom{k+n-3}{n-2} \\ &= \sum_{k=1}^{m} \binom{(k+(n-1))-2}{(n-1)-1} \\ &\quad \text{by definition} \\ &= \sum_{k=1}^{m} f(k, n-1).\end{aligned}$$

■

It is instructive to plot the complexity of HSO against that of LebMeasure, as shown in Fig. 7. Despite both algorithms being exponential in the number of objectives, the complexity of HSO is vastly lower than that of LebMeasure. This means that HSO is significantly more useful than LebMeasure in practical applications, as will be confirmed by the experimental results described in Section V.

V. Performance

We compared the performance of HSO and LebMeasure on two different types of data: randomly generated data and samples taken from the four distinct Pareto optimal fronts of the problems from the DTLZ test suite [26].

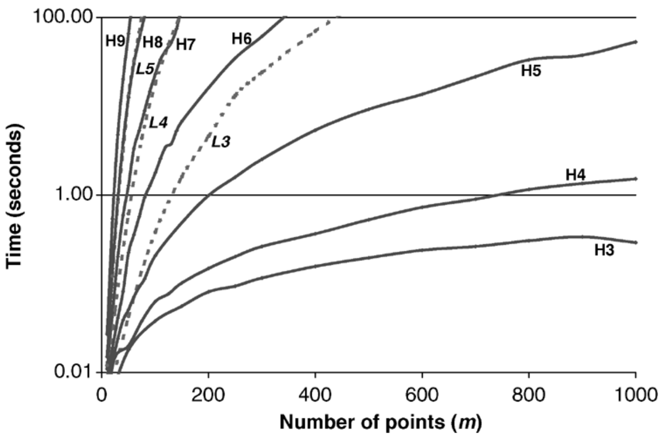


Fig. 8. Log-scale comparison of performance of HSO and (optimized) LebMeasure on randomly generated data. Each line Fn plots the time taken to run Algorithm F with m points in n objectives. Solid lines represent HSO and dashed lines represent LebMeasure. Each data point is average of ten runs on different data sets. L_x coincides with $H_{x+3}, 4 \leq x \leq 5$.

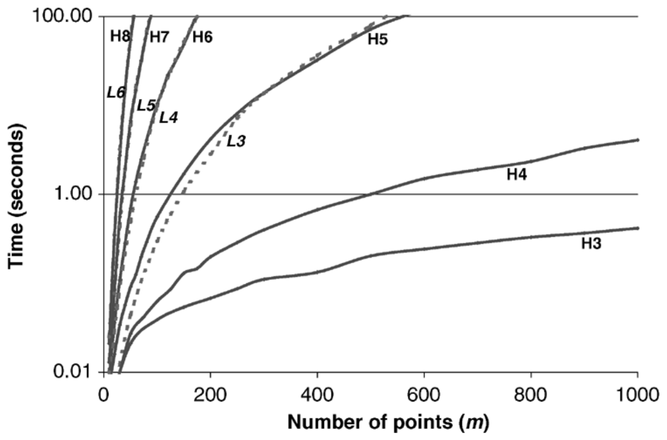


Fig. 9. Log-scale comparison of performance of HSO and (optimized) LebMeasure on spherical front from DTLZ. Each line Fn plots time taken to run Algorithm F with m points in n objectives. Solid lines represent HSO and dashed lines represent LebMeasure. Each data point is average of ten runs on different data sets. L_x coincides with $H_{x+2}, 3 \leq x \leq 6$.

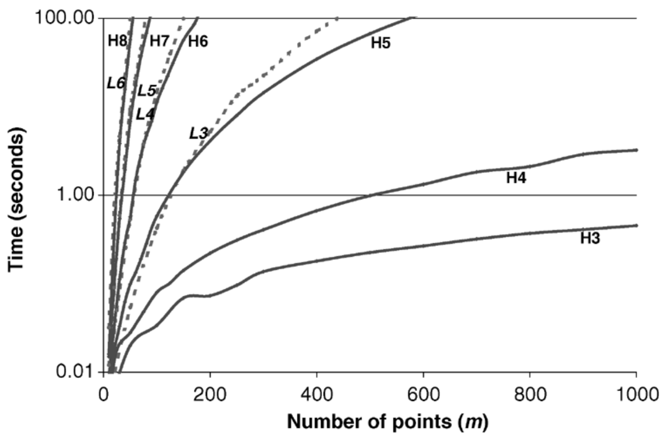


Fig. 10. Log-scale comparison of performance of HSO and (optimized) LebMeasure on linear front from DTLZ. Each line Fn plots time taken to run Algorithm F with m points in n objectives. Solid lines represent HSO and dashed lines represent LebMeasure. Each data point is average of ten runs on different data sets. L_x coincides with $H_{x+2}, 4 \leq x \leq 6$.

However, we discovered that the performance of the basic LebMeasure algorithm was so poor as to make meaningful comparisons with HSO infeasible. To compensate, we incorporated into LebMeasure an attempt to optimize its performance by ordering the points before processing them, as discussed briefly in Section III-B. We used a straightforward ordering based on the closeness of a point to the "edges" of objective space (as also suggested independently by Fleischer (private communication) and others). For each point $\overline{x}$, we count the number of points that are worse than $\overline{x}$ in each objective and sum these counts. We process the points in increasing the order of these sums; thus, we process first the points that are likely to generate the fewest hypercuboids. We note that we performed some (minor) experimentation with other ordering schemes, but we found no convincing evidence generally favoring one scheme over others.

We shall see in Section V-B that this optimization can have a dramatic effect with some patterns of data. We note also that this optimization would induce polynomial complexity for the pathological example in Fig. 2 (but not for the example in Fig. 3).

All timings were performed on a dedicated 1.9-GHz Pentium IV machine with 512 Mb of RAM, running Linux Red Hat 8.0. All algorithms were implemented in Haskell [27] and compiled with $ghc\ -O$. The data used in the experiments are available [28].

A. Randomly Generated Data

We generated sets of m mutually nondominating points in n objectives simply by generating points with random values x, $0.1 \leq x \leq 10$, in all objectives. In order to guarantee mutual nondomination, we initialized $S = \phi$ and added each point $\overline{x}$ to S only if $\overline{x} \cup S$ would be mutually nondominating. We kept adding points until $|S| = m$.

Fig. 8 shows the resulting comparison. The top of the figure marks a time of 100 s, and the center line across the figure marks a time of 1 s. Our experience shows that these times are indicative of the performance required to use hypervolume as a metric and as a diversity mechanism, respectively, although clearly the availability of an efficient incremental algorithm would ease the performance requirement for the latter.

B. Benchmark Data

We also compared the performance of the two algorithms on the four distinct fronts from the well-known DTLZ test suite: the spherical front, the linear front, the discontinuous front, and the degenerate front. For each front, we generated mathematically a representative set of 10 000 points from the (known) Pareto optimal set. Then, to generate a front of size m, we sampled this set randomly.

Figs. 9–12 show the resulting comparisons.

C. Discussion

Figs. 8–11 all have the same basic shape, and they all show the superior performance of HSO compared to that of LebMeasure. Typically, in under 100 s, HSO can process fronts with several thousand points in three to four objectives, fronts with around 1000 points in five objectives, fronts with 100–300 points in

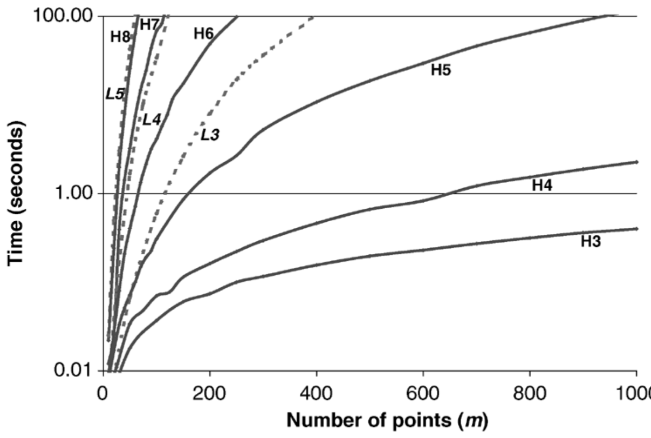


Fig. 11. Log-scale comparison of performance of HSO and (optimized) LebMeasure on discontinuous front from DTLZ. Each line Fn plots time taken to run Algorithm F with m points in n objectives. Solid lines represent HSO and dashed lines represent LebMeasure. Each data point is average of ten runs on different data sets. L_x coincides with $H_{x+3}, 4 \leq x \leq 5$.

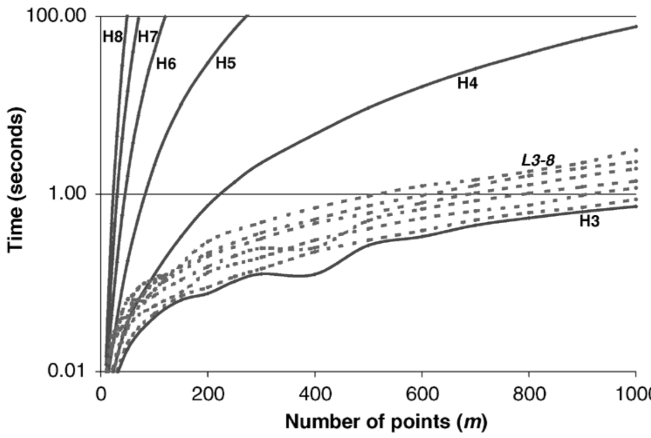


Fig. 12. Log-scale comparison of performance of HSO and (optimized) LebMeasure on degenerate front from DTLZ. Each line Fn plots time taken to run Algorithm F with m points in n objectives. Solid lines represent HSO and dashed lines represent LebMeasure. Each data point is average of ten runs on different data sets.

six to seven objectives, and fronts with 50–80 points in eight to nine objectives. The optimized LebMeasure is unable to process fronts with even 50 points in more than five to six objectives in under 100 s. Thus, HSO increases the utility of hypervolume by enabling calculations with reasonable-sized fronts in almost any likely number of objectives.

Table IV summarizes the sizes of fronts that HSO can process in our two nominated times, for randomly generated data in various numbers of objectives. It is notable that HSO performs generally better on the randomly generated data than on the DTLZ fronts, although it is unclear which is likely to represent more closely the fronts in real-world applications.

Fig. 12 makes a stark contrast to the other graphs. The point-ordering optimization that we implemented enables LebMeasure to process these fronts in polynomial time, as shown by the closely grouped lines Ln on the graph. Note that the order of the points is irrelevant to HSO, as the points are sorted before the algorithm is run.

TABLE IV
SIZES OF FRONTS IN VARIOUS NUMBERS OF OBJECTIVES THAT HSO CAN PROCESS IN TIMES INDICATED (RANDOMLY GENERATED DATA)

n	1 second	100 seconds
3	2,200	$\gg$40,000
4	750	$\approx$9,000
5	200	1,300
6	80	340
7	45	145
8	30	80
9	20	55

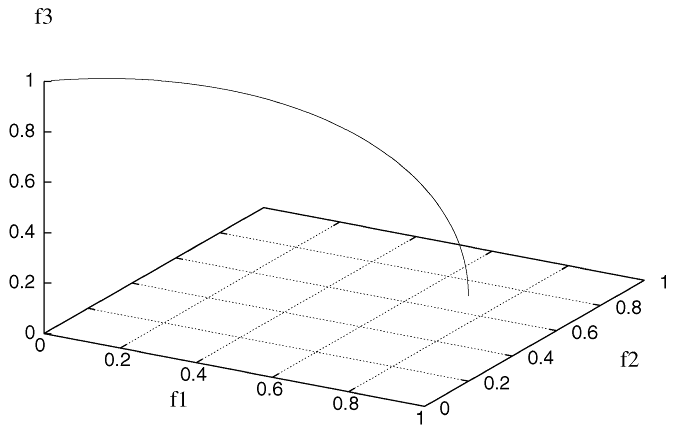


Fig. 13. DTLZ5 front in three objectives. Front is defined for any number of objectives; it forms an arc embedded in n-objective space, where as the values in f_n decrease, the values in all other objectives increase.

It is worth investigating the nature of the degenerate data. Fig. 13 plots the front of the DTLZ5 test function [26] in three objectives. The degenerate front is basically an arc embedded in n-objective space. The important feature of this front for LebMeasure is that it is (in overall form) identical to the pathological example from Fig. 2. The point with the biggest value in f_n has the smallest values in $f_1 \cdots f_{n-1}$; the point with the second biggest value in f_n has the second smallest values in $f_1 \cdots f_{n-1}$; and so on. Thus, when these points are in their optimal ordering (decreasing values of f_n), LebMeasure can process them without generating any nondominated spawns, thus summing only m hypercuboids.

We believe that this sort of degenerate front is likely to be rare in real-world problems. Moreover, the front as defined presents itself in the worst case form for HSO. But while HSO cannot benefit from reordering the *points*, it can benefit from reordering the *objectives*. If we reverse the objective values in each point, and then rerun the experiments, we get the comparison shown in Fig. 14. (Note the different scale used in this figure.) We can see that both HSO and LebMeasure are able to process the reversed degenerate data in polynomial time, with appropriate preprocessing.

Thus, permuting objectives allows HSO to beat LebMeasure on the degenerate front, and it should deliver a general improvement to the performance of HSO [29].

VI. CONCLUSION AND FUTURE WORK

Hypervolume is a popular metric for comparing the performance of multiobjective optimization algorithms, particularly

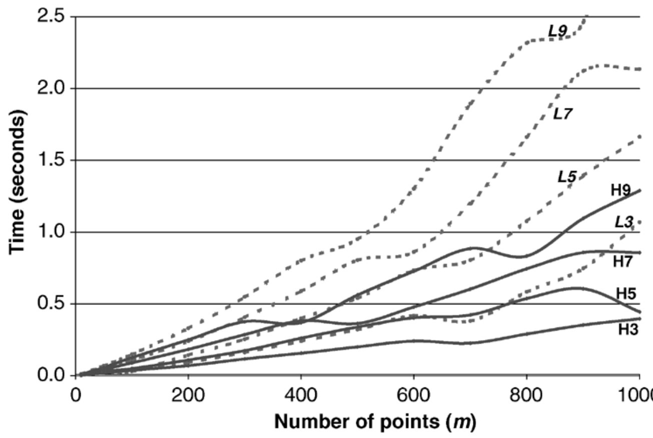


Fig. 14. Comparison of performance of HSO and (optimized) LebMeasure on degenerate front from DTLZ, with objectives in reverse order. Each line Fn plots time taken to run Algorithm F with m points in n objectives. Solid lines represent HSO and dashed lines represent LebMeasure. Each data point is average of ten runs on different data sets.

evolutionary algorithms. However, calculating the hypervolume of a front is expensive. We have shown that both previously studied algorithms for calculating hypervolume exactly are exponential in at least the number of objectives, and we have described a hypervolume algorithm HSO, that, while also exponential in the number of objectives in the worst case, runs in significantly less time than previous algorithms, i.e., two to three orders of magnitude less for randomly generated and benchmark data in three to eight objectives. HSO extends the utility of hypervolume by enabling calculations on larger fronts in more objectives. Indeed, we believe that the performance of HSO will be sufficient for many real-world applications and setups. HSO also raises the possibility of an algorithm that is genuinely fast enough to support the use of hypervolume as a diversity mechanism in multiobjective evolutionary algorithms.

We intend to take this research in several directions.

Speed-up HSO: We plan to make HSO faster, for example by minimizing the cost of dominance checking, by minimizing repeated work in HSO, or by discovering heuristics that allow HSO to permute objectives intelligently. Either of the latter two possibilities could make a huge difference to the practical performance of HSO.

Develop a fast incremental algorithm: We plan to develop an incremental version of HSO to support the use of hypervolume as a diversity mechanism. Given a set of points S, if we add a point $\overline{x}$ to S such that $\overline{y} < \overline{x} < \overline{z}$ in objective k, it is clear that some slices in the hypervolume of S will not be affected by $\overline{x}$. Slices above $\overline{z}$ in objective k will be unaffected, and slices below $\overline{y}$ in objective k will be unaffected if $\overline{y}$ dominates $\overline{x}$ in the other objectives. If we choose k to maximize the number of unaffected slices, this should make it relatively cheap to calculate the hypervolume that $\overline{x}$ adds to S.

Decide the complexity question: We plan to search for a polynomial-time algorithm for calculating hypervolume, or, alternatively, we plan to try to prove that a polynomial-time algorithm is impossible. We suspect the latter.

ACKNOWLEDGMENT

The authors would like to thank M. Fleischer for useful discussions on the LebMeasure algorithm, its complexity, and possible optimizations, and for helpful feedback on an earlier draft of the paper. They also thank G. Royle for helpful comments on hypervolume in general.

REFERENCES

[1] S. Huband, P. Hingston, L. While, and L. Barone, "An evolution strategy with probabilistic mutation for multi-objective optimization," in *Proc. Congr. Evol. Comput.*, vol. 4, 2003, pp. 2284–2291.

[2] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization," in *Proc. EUROGEN*, K. C. Giannakoglou *et al.*, Eds., Barcelona, Spain, 2001, pp. 95–100.

[3] R. C. Purshouse and P. J. Fleming, "The multiobjective genetic algorithm applied to benchmark problems—An analysis," Department Automatic Control Systems Eng., Univ. Sheffield, U.K., Res. Rep. 796, 2001.

[4] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[5] J. Knowles and D. Corne, "M-PAES: A memetic algorithm for multiobjective optimization," in *Proc. Congr. Evol. Comput.*, vol. 1, 2000, pp. 325–332.

[6] M. Laumanns, L. Thiele, and E. Zitzler, "Running time analysis of multiobjective evolutionary algorithms on pseudo-Boolean functions," *IEEE Trans. Evol. Comput.*, vol. 8, no. 2, pp. 170–182, Apr. 2004.

[7] C. A. Coello Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 256–279, Jun. 2004.

[8] M. Farina, K. Deb, and P. Amato, "Dynamic multi-objective optimization problems: Test cases, approximations, and applications," *IEEE Trans. Evol. Comput.*, vol. 8, no. 5, pp. 425–442, Oct. 2004.

[9] T. Okabe, Y. Jin, and B. Sendhoff, "A critical survey of performance indexes for multi-objective optimization," in *Proc. Congr. Evol. Comput.*, vol. 2, 2003, pp. 878–885.

[10] J. Wu and S. Azarm, "Metrics for quality assessment of a multiobjective design optimization solution set," *J. Mechanical Design*, vol. 123, pp. 18–25, 2001.

[11] R. Purshouse, "On the evolutionary optimization of many objectives," Ph.D. dissertation, Univ. Sheffield, Sheffield, U.K., 2003.

[12] E. Zitzler, "Evolutionary algorithms for multiobjective optimization: Methods and applications," Ph.D. dissertation, Swiss Federal Inst. Technology (ETH) Zurich, Switzerland, 1999.

[13] M. Laumanns, E. Zitzler, and L. Thiele, "A unified model for multiobjective evolutionary algorithms with elitism," in *Proc. Congr. Evol. Comput.*, vol. 1, 2000, pp. 46–53.

[14] M. Fleischer, "The measure of Pareto optima: Applications to multi-objective metaheuristics," in *Evolutionary Multiobjective Optimization*. ser. Lecture Notes in Computer Science, C. M. Fonseca *et al.*, Eds. Berlin, Germany: Springer-Verlag, 2003, vol. 2632, pp. 519–533.

[15] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Trans. Evol. Comput.*, vol. 7, no. 2, pp. 117–132, Apr. 2003.

[16] M. Fleischer, "The measure of Pareto optima: Applications to multi-objective metaheuristics," Inst. Systems Research, Univ. Maryland, College Park, MD, Tech. Rep. ISR TR 2002-32, 2002.

[17] J. Knowles, D. Corne, and M. Fleischer, "Bounded archiving using the lebesgue measure," in *Proc. Congr.Univ. Maryland*, vol. 4, H. Abbass and B. Verma, Eds., 2003, pp. 2490–2497.

[18] E. Zitzler. (2001) Hypervolume metric calculation. ftp://ftp.tik.ee.ethz.ch/pub/people/zitzler/hypervol.c

[19] J. Knowles, "Local-Search and hybrid evolutionary algorithms for pareto optimization," Ph.D. dissertation, Univ. Reading, Reading, U.K., 2002.

[20] R. Everson, J. Fieldsend, and S. Singh, "Full elite sets for multi-objective optimization," in *Proc. 5th Int. Conf. Adapt. Comput. Design Manuf.*, 2002, pp. 87–100.

[21] K. Deb, *Multiobjective Optimization Using Evolutionary Algorithms*. New York: Wiley, 2001.

[22] V. R. Khare, X. Yao, and K. Deb, "Performance scaling of multi-objective evolutionary algorithms," in *Evolutionary Multi-Objective Optimization*. ser. Lecture Notes in Computer Science, C. M. Fonseca *et al.*, Eds. Berlin, Germany: Springer-Verlag, 2003, vol. 2632, pp. 376–390.

[23] J. Knowles and D. Corne, "Properties of an adaptive archiving algorithm for storing nondominated vectors," *IEEE Trans. Evol. Comput.*, vol. 7, no. 2, pp. 100–116, Apr. 2003.

[24] C. Coello Coello, *Evolutionary Algorithms for Solving Multi-Objective Problems*. New York: Kluwer, 2002.

[25] W. Feller, *An Introduction to Probability Theory and Its Applications*. New York: Wiley, 1968.

[26] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multi-objective optimization test problems," in *Proc. Congr. Evol. Comput.*, vol. 1, 2002, pp. 825–830.

[27] S. Peyton-Jones, *Haskell 98 Language and Libraries: The Revised Report*. Cambridge, U.K.: Cambridge Univ. Press, 2003.

[28] *http://wfg.csse.uwa.edu.au/Hypervolume*.

[29] L. While, L. Bradstreet, L. Barone, and P. Hingston, "Heuristics for optimizing the calculation of hypervolume for multi-objective optimization problems," in *Proc. Congr. Evol. Comput.*, vol. 3, 2005, pp. 2225–2232.

Lyndon While (M'01–SM'03) received the B.Sc. and Ph.D. degrees from the Imperial College of Science and Technology, London, U.K., in 1985 and 1988, respectively.

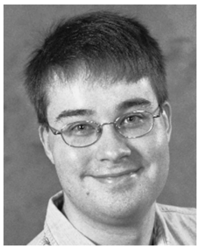
He is currently a Senior Lecturer in the School of Computer Science and Software Engineering, the University of Western Australia, Australia. His research interests include evolutionary algorithms, multiobjective optimization, and the semantics and implementation of functional programming languages.

Phil Hingston (M'00) received the B.Sc. degree from the University of Western Australia, Australia, in 1978, and the Ph.D. degree from Monash University, Melbourne, Australia, in 1984.

He is currently a Senior Lecturer in the School of Computer and Information Science, Edith Cowan University, Australia. His research interests include artificial intelligence and its application to industrial design tasks, as well as the modeling of social and natural systems.

Luigi Barone (M'04) received the B.Sc. and Ph.D. degrees from the University of Western Australia, Australia, in 1994 and 2004, respectively.

He is currently an Associate Lecturer in the School of Computer Science and Software Engineering, the University of Western Australia. His research interests include evolutionary algorithms and their use for optimization and opponent modeling and the modeling of biological systems.



Simon Huband (M'04) received the B.Sc. and Ph.D. degrees from the University of Western Australia, Australia, in 1997 and 2003, respectively.

He is currently a Research Fellow in the School of Computer and Information Science, Edith Cowan University, Australia. His research interests include parallel programming, the design of test problems, and the optimization of industrial systems using evolutionary algorithms.