A Fast Incremental Hypervolume Algorithm

Lucas Bradstreet, *Student Member, IEEE*, Lyndon While, *Senior Member, IEEE*, and Luigi Barone, *Member, IEEE*

***Abstract*—When hypervolume is used as part of the selection or archiving process in a multiobjective evolutionary algorithm, it is necessary to determine which solutions contribute the least hypervolume to a front. Little focus has been placed on algorithms that quickly determine these solutions and there are no fast algorithms designed specifically for this purpose. We describe an algorithm, IHSO, that quickly determines a solution's contribution. Furthermore, we describe and analyse heuristics that reorder objectives to minimize the work required for IHSO to calculate a solution's contribution. Lastly, we describe and analyze search techniques that reduce the amount of work required for solutions other than the least contributing one. Combined, these techniques allow multiobjective evolutionary algorithms to calculate hypervolume inline in increasingly complex and large fronts in many objectives.**

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

I. Introduction

Hypervolume [1], also known as the S-metric [2] or the Lebesgue measure [3], [4], has recently been finding favor as a metric for comparing the performance of multiobjective evolutionary algorithms (MOEAs). 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.* [5] 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 [6] 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.

A fast algorithm for calculating hypervolume exactly is the *hypervolume by slicing objectives* algorithm (HSO) [7]–[9]. HSO works by processing the objectives in a front, rather than the points. It divides the n D-hypervolume to be measured into separate $n-1$ D-slices through one of the objectives, then it calculates the hypervolume of each slice and sums these values to derive the total. In the worst case, HSO is exponential in the number of objectives, but until recently, it had better complexity than other algorithms. In addition, While *et al.* [10] have described good heuristics that optimize the order in which the objectives should be processed for a given front by estimating the "worst-case work" required to process the slices remaining after eliminating each objective. These heuristics reduce the running time of HSO for representative data by 25%–98%.

Algorithms from the computational geometry field have recently been applied to hypervolume calculation by Beume and Rudolph and separately by Fonseca *et al.* Beume and Rudolph [11] adapt the Overmars and Yap [12] algorithm for solving the Klee's measure problem to instead calculate the hypervolume of a front. Similarly, Fonseca *et al.* [13] apply the Overmars and Yap algorithm for the 3-D base case in order to provide a performance boost to HSO. Beume and Rudolph's adaptation boasts an impressive improvement in worst-case complexity, from $O(n^{d-1})$ to $O(n \log n + n^{d/2})$, however, as of yet there are no performance comparisons between their algorithm and HSO with heuristics.

Hypervolume is also used inline in some evolutionary algorithms, as part of a diversity mechanism [14], as part of an archiving mechanism [15], or recently as part of the selection mechanism [16], [17]. The requirement in such cases is to compare the *exclusive hypervolume* contributed by different points, i.e., the amount by which each point increases the hypervolume of the set. 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 uses is an incremental algorithm that minimizes the expense of repeated invocations.

The principal contributions of this paper are a version of HSO which is customized for inline incremental hypervolume calculations, and queueing techniques and heuristics that improve performance for hypervolume algorithms used within a MOEA.

The customized algorithm has two parts.

- The algorithm *incremental HSO* (IHSO) calculates the exclusive hypervolume of a point p relative to a set of points S. The principal optimizations in IHSO are minimizing the number of slices that have to be processed, and ordering the objectives intelligently.
- The algorithm IHSO* performs point selection for diversity, archiving, or fitness. IHSO* works by repeated application of IHSO to calculate the exclusive hypervolume for each point in a set. The principal optimizations in IHSO* are ordering the points intelligently, and calculating as little hypervolume as possible for each point.

IHSO* will provide a substantial performance improvement for evolutionary algorithms that perform inline incremental hypervolume calculations. We note that although Beume and Rudolph's recent work [11] does improve the complexity of hypervolume algorithms for metric calculations, customized algorithms are not yet available for incremental hypervolume calculations.

Manuscript received December 21, 2006; revised July 30, 2007. First published March 28, 2008; current version published December 12, 2008.

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

Digital Object Identifier 10.1109/TEVC.2008.919001

The rest of this paper is structured as follows. Section II defines the concepts and notation used in multiobjective optimization and throughout this paper. Section III describes HSO and the heuristics used to optimize its performance. Section IV describes how HSO can be customized into IHSO and IHSO* to calculate exclusive hypervolume efficiently. Section V reports on some experiments to determine the fastest incremental algorithm, and to explore some important issues for users of the algorithm. Section VI concludes this paper and discusses some possibilities for future work.

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. Given two objective vectors $\overline{x}$ and $\overline{y}$, $\overline{x}$ *dominates* $\overline{y}$ iff $\overline{x}$ 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 a set of solutions X iff there is no vector in X that dominates $\overline{x}$. X is a *nondominated set* iff 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* iff $\overline{x}$ 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.

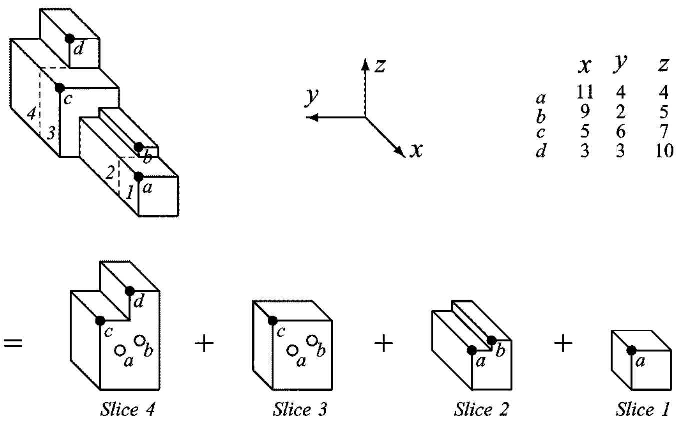
Given a set X of solutions returned by an algorithm, the question arises how good the set X is, i.e., how well it approximates the Pareto optimal set. One metric used for comparing sets of solutions is to measure the *hypervolume* of each set. The hypervolume of X is the total size of the space that is dominated by the solutions in X. The hypervolume of a set is measured relative to a reference point, usually the anti-optimal point or "worst possible" point in space. (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.) 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'.

Precise definitions of these terms can be found in [18].

III. Hypervolume By Slicing Objectives

Given a set of mutually nondominating points in n objectives, HSO is based on the idea of processing the points *one objective at a time*.

Initially, the points are sorted by their values in the first objective to be processed. 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 in each slice is calculated and each slice is multiplied by its depth 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. The kth slice from the top can contain only the points with the best k values in the first objective. However, not all points "contained" by a slice will contribute volume to that slice: some points may be dominated in the remaining objectives and will contribute nothing. After each step, 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.



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

Fig. 1. One step in HSO for the four three-objective points shown. Objective x is processed, leaving four two-objective shapes in y and z. Points are marked by circles and labeled with letters: unfilled circles represent points that are dominated in y and z. Slices are labeled with numbers, and are separated on the main picture by dashed lines. (Figure reproduced from [9].)

Fig. 1 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 natural base case for HSO is when only one objective remains, 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 two objectives remain, which is an easy and fast special case.

Fig. 2 gives pseudocode for HSO.

A. The Complexity and Performance of HSO

The following recurrence relation captures the worst-case complexity of HSO [9]:

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

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

The summation in (2) represents the fact that each slicing action generates m slices that are processed independently to derive the hypervolume of the front.

Solving this recurrence relation gives the following [9]:

$$f(m,n) = \binom{m+n-2}{n-1}. \tag{3}$$

Thus, HSO is exponential in the number of objectives n, in the worst case (we assume that $m > n$).

The "worst case" in this context means we assume that no (partial) point is ever dominated during the execution of HSO, thus maximizing the number of points in each slice that is processed. However, this is unlikely to be true for real-world fronts. The amount of time required to process a given front depends

```
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):
  d = True
  while d && k <= n
    d = not (q[k] beats p[k])
    k = k + 1
  return d
```

Fig. 2. Pseudocode for HSO. (Code reproduced from [10].)

5	$\cdots$	5	1
4	$\cdots$	4	2
3	$\cdots$	3	3
2	$\cdots$	2	4
1	$\cdots$	1	5

Fig. 3. A pathological example for HSO. This pattern describes sets of five points in n objectives, $n \geq 3$. All columns except the last are identical. The pattern can be generalized for other numbers of points. (Example reproduced from [10].)

crucially on how many points are dominated at each stage and, in addition, on how early in the process points dominate other points.

From this fact, we can infer that the time to process a given front varies with the order in which the objectives are processed. A simple example illustrates how. Consider the set of points in Fig. 3, in a maximization problem.

If we process the first objective (or, in fact, any objective except the last), no point dominates any other point in the list in the remaining $n-1$ objectives. Thus, we do indeed have the worst case for HSO, generating m slices containing, respectively, $1, 2, \ldots, m$ points.

If we process the last objective, each point dominates all subsequent points in the list in the remaining $n-1$ objectives. Then, we generate m slices *each containing only one point*. Specifically, the top slice (corresponding to the highest value in the last objective) contains only the point $1 \cdots 1$, the second slice contains only the point $2 \cdots 2$, all the way down to the bottom slice, which contains only the point $m \cdots m$. This is of course the best case for HSO, and the hypervolume is calculated much more quickly.

Note that, in general, there is a continuum of performance improvement available: for example, for the points in Fig. 3, the earlier the last objective is processed, the faster the hypervolume will be calculated. Thus, enhancing HSO with a mechanism to identify a good order in which to process the objectives in a given front can make a substantial difference to its performance.

B. Optimizing the Performance of HSO

While *et al.* describe and evaluate two heuristics for choosing the order in which the objectives should be processed for a given front [10]. They characterize the better heuristic as "minimizing the amount of worst-case work" (MWW). For each objective, MWW:

- calculates the number of nondominated partial points that will be in each slice;
- estimates the worst-case amount of work required to process each slice, using (3);
- and sums these values to estimate the amount of work required if HSO processes this objective first.

Then, HSO processes the objective that represents the least work. MWW is applied at each iteration of HSO until only four objectives remain.

An empirical comparison of HSO versus HSO+MWW on randomly generated fronts and on fronts from the well-known DTLZ test suite [19] shows that MWW can reduce the time to process fronts in 5–9 objectives by 25%–98%.

IV. Running HSO Incrementally

Hypervolume is used inline in an evolutionary algorithm in three ways:

- as part of a diversity mechanism;
- as part of an archiving mechanism;
- as part of the selection mechanism.

In all three contexts, the requirement is to calculate the *exclusive hypervolume* contributed by a point p relative to a set of points S, i.e., how much additional hypervolume we get by adding p to S. This can be defined as

$$\mathrm{ExcHyp}(p, S) = \mathrm{Hyp}(S \cup \{p\}) - \mathrm{Hyp}(S). \qquad (4)$$

For example, the exclusive hypervolume contributed by Point **b** in Fig. 1 is the cuboid bounded by **b** and by the point (5, 0, 4), i.e., the long thin cuboid on which **b** sits. Note that exclusive hypervolumes usually have a much more complicated shape than this: consider as an example Point **c** in Fig. 1.

A typical requirement when hypervolume is used in this way is to calculate the exclusive hypervolume contributed by each of a set of points S, then to discard the point in S that contributes

```
ihso (z, 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 (z, 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 (z, pl, k):
  ql = []
  s = {}
  v = z[k]
  dominated = false
  while pl /= [] and not dominated
    p = head (pl)
    pl = tail (pl)
    if v beats p[k]
      then add (|v - p[k]|, ql) into s
           v = p[k]
    ql = insert (p, k+1, ql)
    dominated = dominates (p, z, k+1)
  if not dominated
    then add (|v - refPoint[k]|, ql) into s
  return s
```

Fig. 4. Pseudocode for IHSO. Other functions are defined in Fig. 2. The reordering of the objectives is not shown.

the least exclusive hypervolume. Thus, we return the subset of S of cardinality $|S|-1$ that has the largest hypervolume. This idea can be extended to situations where we need to discard multiple points from S[20], [21], but we do not deal with this issue here.

Obviously, we can calculate the exclusive hypervolume contributed by each point in S by $|S|+1$ applications of HSO: one to S itself, and one to each subset of size $|S|-1$. However, we can do far better than this performance-wise by customizing HSO to calculate exclusive hypervolumes directly. We define a new algorithm IHSO that takes a point p and a set of mutually nondominating points S and returns $\mathrm{ExcHyp}(p, S)$. We customize HSO in three ways to derive IHSO.

1) Disregarding "higher" slices: p will not contribute to any slice above itself in the current objective, therefore, the hypervolumes of these slices need not be calculated. For example, in Fig. 1, Point **b** contributes nothing to Slice 1.
2) Disregarding some "lower" slices: if p is dominated by a point q in S in the objectives after the current one, then p will not contribute to any slice containing q (or any point that dominates q), and the hypervolumes of these slices need not be calculated. For example, in Fig. 1, Point **b** contributes nothing to Slices 3 or 4, because it is dominated by Point **c** in y and z.
3) Processing the objectives in the right order: as with HSO, we can optimize the performance of IHSO by selecting a good order in which to process the objectives.

Fig. 4 gives pseudocode for IHSO. The code assumes that none of the points in `ps` dominates `z`, although the converse is not true: `z` may dominate one or more points in `ps`. The principal differences from HSO in Fig. 2 are in the function `slice`.

- A slice is added to `s` only if it is below `z` in the current objective, i.e., below `z`[k].

```
Order the points by some metric
Evaluate the first point
Save this point as the current smallest
for each subsequent point
  Evaluate point while worse than the smallest
  Save point if it is the smallest
return the smallest point
```

Fig. 5. Outline of the point-ordering scheme in IHSO*.

- If at any time `z` is dominated in the remaining objectives, no more slices are added to `s`.

Given IHSO, we can define an algorithm IHSO* to identify the point in a set S that contributes the least exclusive hypervolume to S. We use $|S|$ applications of IHSO to calculate $\mathrm{ExcHyp}(p, S-\{p\})$ for each point p in S, then simply return the point with the smallest value. Within IHSO*, it is useful to order the calculations so that small points are likely processed first. This enables early termination for subsequent points: if the exclusive hypervolume for p is known, then as soon as the exclusive hypervolume for q is known to be bigger, we can eliminate q from consideration as the smallest contributor. Note also that the order in which the objectives are processed can be different for different points in S.

Thus, there are two questions to be answered in order to derive an efficient implementation of IHSO and IHSO*.

A. How Do We Order the Objectives When Calculating $ExcHyp(p, S)$ in IHSO?

We have tried several heuristics that can be used to order the objectives for a point p.

1) *Rank*: process first the objective in which p is best, so that it is more likely to be dominated early.
2) *Reverse rank*: process first the objective in which p is worst, so that there are fewer slices to calculate.
3) *Dominated*: find the point q that beats p in the most objectives, and first process the slices in which p beats q. This method partitions the objectives between those where q beats p, and those where p beats q. Within these partitions, order objectives by the rank heuristic.
4) *MWW*: as defined in Section III-B.

While these heuristics can be used to reorder objectives for all calculations [10], we find experimentally that it is more effective to reorder the objectives for each individual slice recursively calculated by IHSO. Although this comes at additional cost, savings are made on slices that are expensive to calculate, for example, a slice with a difficult shape, or one with many points or objectives. One example of where savings are made using this approach is for the dominated heuristic, where the point q used to reorder the objectives may not even exist in a given slice.

B. How Do We Order the Points When Calculating Their Exclusive Hypervolumes in IHSO*?

We have devised two schemes to improve the performance of IHSO* when used to find the worst contributing point. The first scheme reorders the points with the aim of calculating the worst point early. Unnecessary calculations are then saved on subsequent points. This scheme is outlined in Fig. 5.

```
Evaluate each point a bit
Identify the smallest point
while the smallest point is not completed
  Evaluate the smallest point a bit more
  Identify the new smallest point
return the smallest point
```

Fig. 6. Outline of the best-first queueing scheme in IHSO*.

We have defined two measures that can be used to order the points. Each point p can be assessed by the following.

1) *Rank*: the sum of the number of points that beat p in each objective. Points are sorted in descending order.
2) *Volume*: the "inclusive hypervolume" of p: the product of its objectives. Points are sorted in ascending order.

The point with the least hypervolume contribution to the set is more likely to have a large rank value and to have a small inclusive hypervolume.

We have also defined an alternative "best-first" queueing (BFQ) scheme that processes the points "concurrently," to avoid the question of ordering. This scheme is outlined in Fig. 6. The principal parameter in the queueing scheme is the definition of "a bit," i.e., the granularity of the concurrency. If the granularity is too coarse, the algorithm will do more calculation than necessary: if it is too fine, the overhead of managing the queue will become significant. At present, we use a simple granularity scheme based on specifying the dimensionality of a hypervolume to be calculated in each iteration of the loop. A granularity of ρ means that one slice in ρ objectives is calculated in each iteration. This dimensionality is set according to (5)

$$\rho = \max(\min(n, 4), n - 2). \tag{5}$$

The application of $\max$ in (5) means that for low dimensionalities, we abandon the queueing scheme and just calculate the complete exclusive hypervolume of each point. This system works well for the limited range of dimensionalities studied so far, but it is likely to need updating in the future.

We have implemented the BFQ scheme using a heap based priority queue. Note that this approach requires the IHSO* algorithm to be modified to update the overall hypervolume contribution of points whenever a slice is calculated in ρ dimensions.

Section V describes an empirical comparison of the performance of these methods.

V. Experiments and Evaluation

We performed a series of experiments to explore issues with IHSO and IHSO*, and to determine the combination of heuristics that offers the best performance. We used two types of data in the experiments.

- We used randomly generated fronts, initialized 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 used the discontinuous and spherical fronts from the DTLZ test suite [19]. For each front, we generated mathematically a representative set of 10 000 points from the (known) Pareto optimal set: then to form a front of a given size, we sampled this set randomly. We omit the linear front from DTLZ because it gives very similar performance to the spherical front, and we omit the degenerate front because it can be processed in polynomial time [9], [10], and it is somewhat unrealistic anyway.

The DTLZ fronts may not realistically represent real-world data, and therefore we believe that random fronts provide a better performance baseline for most problems. As it is hard to give performance comparisons for all front shapes and types, random data may provide a better approximation of IHSO*'s performance on these fronts than specific DTLZ fronts.

The data used in the experiments are available [22]. Source code for our optimized algorithm is available from the same site. All timings were performed on a dedicated 2.8 GHz Pentium IV machine with 512 Mb of RAM, running Red Hat Enterprise Linux 3.0. All algorithms were implemented in C and compiled with *gcc -O3*. All times include the costs of calculating the heuristics, where appropriate.

A. Does Point-Ordering Matter?

We performed a series of experiments to establish whether the time needed to identify the least-contributing point in a front depends on the order in which the points in the front are processed. Each graph in Fig. 7 shows five lines, each of which corresponds to one front with 30 points in nine objectives. Each line plots a continuous cumulative histogram of the distribution of times needed to determine the least-contributing point for 50 000 randomly generated point-orderings of that front. No objective-reordering is applied.

Fig. 7 shows clearly that evaluating points in the right order can make a huge difference to the performance of the algorithm. The raw data show that typically the best order is processed 300–6000 times faster than the worst order, and 60–200 times faster than the median order.

Thus, point-ordering will play an important role in optimizing the performance of IHSO*.

B. What Is the Best Algorithm?

We performed a series of experiments to identify a good combination of heuristics to use in IHSO and IHSO*. Each graph in Figs. 8–10 shows the performance for varying front-sizes of the six combinations of the following heuristics from Section IV.

- *Point-ordering:* Rank, volume, and the best-first queueing scheme.
- *Objective-ordering:* Rank and dominated.

Other heuristics discussed in Section IV performed consistently worse than those illustrated in the figures. The graphs plot fronts up to 1000 points that can be processed in 1.5 s: this should be a reasonable amount of time for an incremental hypervolume calculation for most applications. Figs. 8–10 show that using the BFQ approach gives the best results other than in five objectives. Although BFQ loses slightly in five objectives, this is probably as a result of the overhead caused by the priority queue. If point reordering algorithms make perfect decisions, they will always outperform the BFQ approach. However, the savings made for more complex fronts outweigh the cost of maintaining the queue as can be observed for all front types in 8 and 11 objectives. Additionally, using a point-ordering algorithm rather than BFQ introduces a greater uncertainty in the

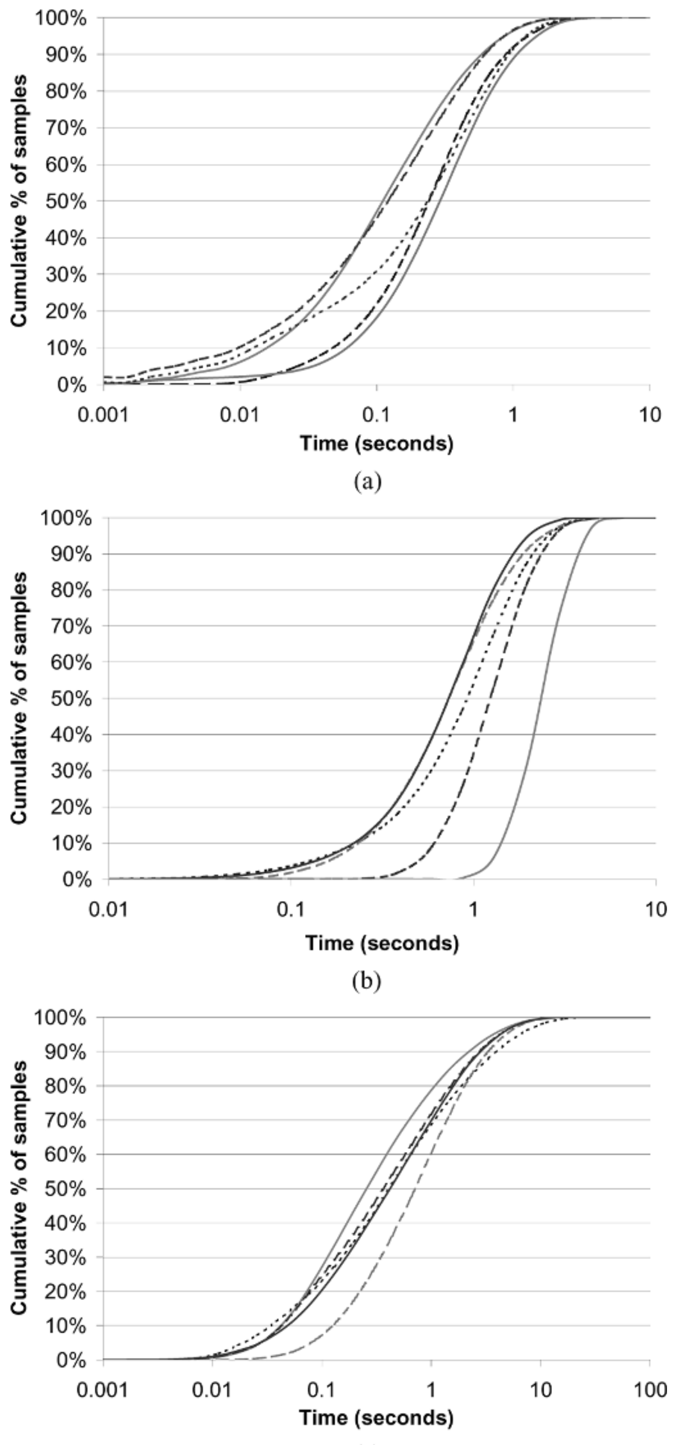


Fig. 7. Variation in processing time for IHSO* for different point-orderings. The five lines on each graph each plot a continuous cumulative histogram of the (log-scale) processing times for 50 000 distinct orderings for one front. (a) Random fronts: 30 points in nine objectives. (b) Discontinuous fronts: 30 points in nine objectives. (c) Spherical fronts: 30 points in nine objectives.

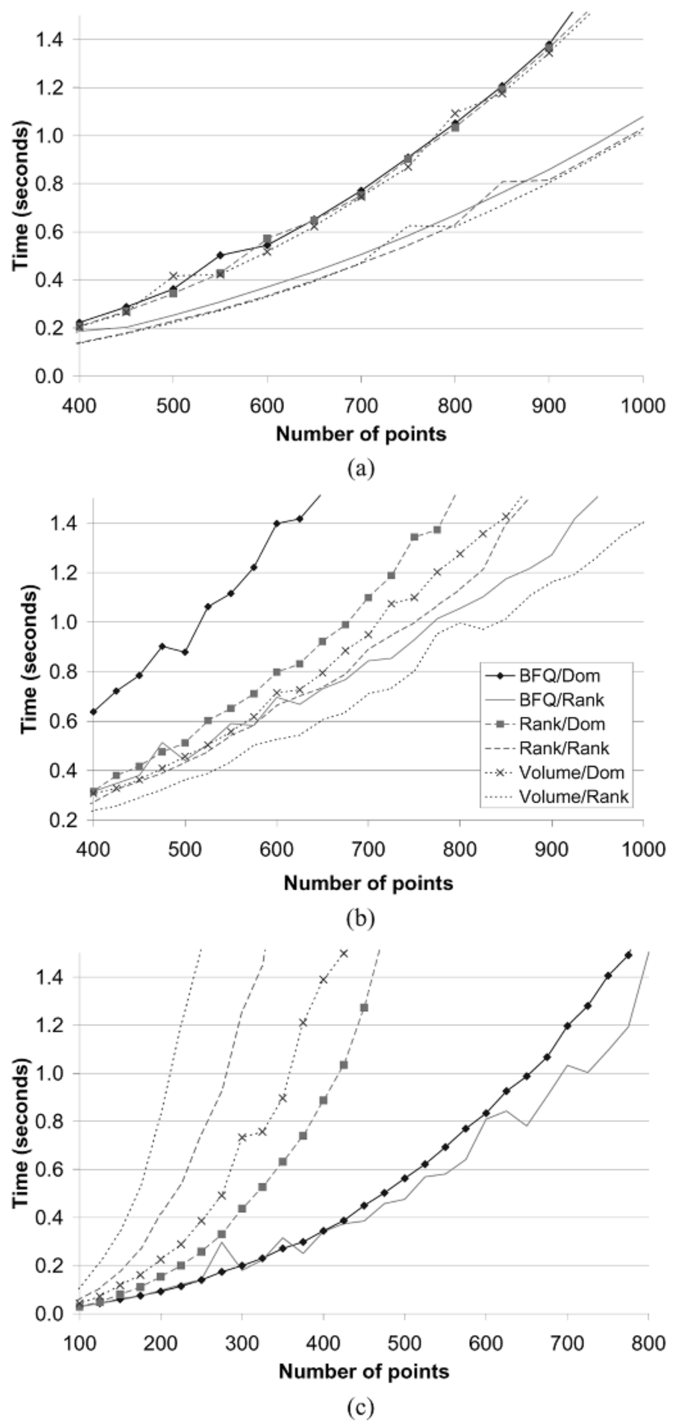


Fig. 8. Comparison of the performance of IHSO* with various heuristic combinations. Each line plots the average processing time for 200 distinct fronts in five objectives. The legend on the middle graph applies for all three. (a) Random fronts in five objectives. (b) Discontinuous fronts in five objectives. (c) Spherical fronts in five objectives.

results. A bad point-ordering decision can, in the worst case, require the calculation of every point's entire exclusive hypervolume. This effect is evident when comparing the BFQ/rank algorithm to the rank/rank algorithm for random fronts, shown in Fig. 9(a). While the results are reasonably close for most of the data points, large fluctuations are observed.

For all discontinuous and random data, BFQ/rank compares favorably to or beats all other objective heuristic and point-ordering techniques. This dominance increases with the number of objectives.

For spherical data, the dominated heuristic performs extremely well. However, we believe spherical data is being especially exploited by this heuristic. Examination of the data

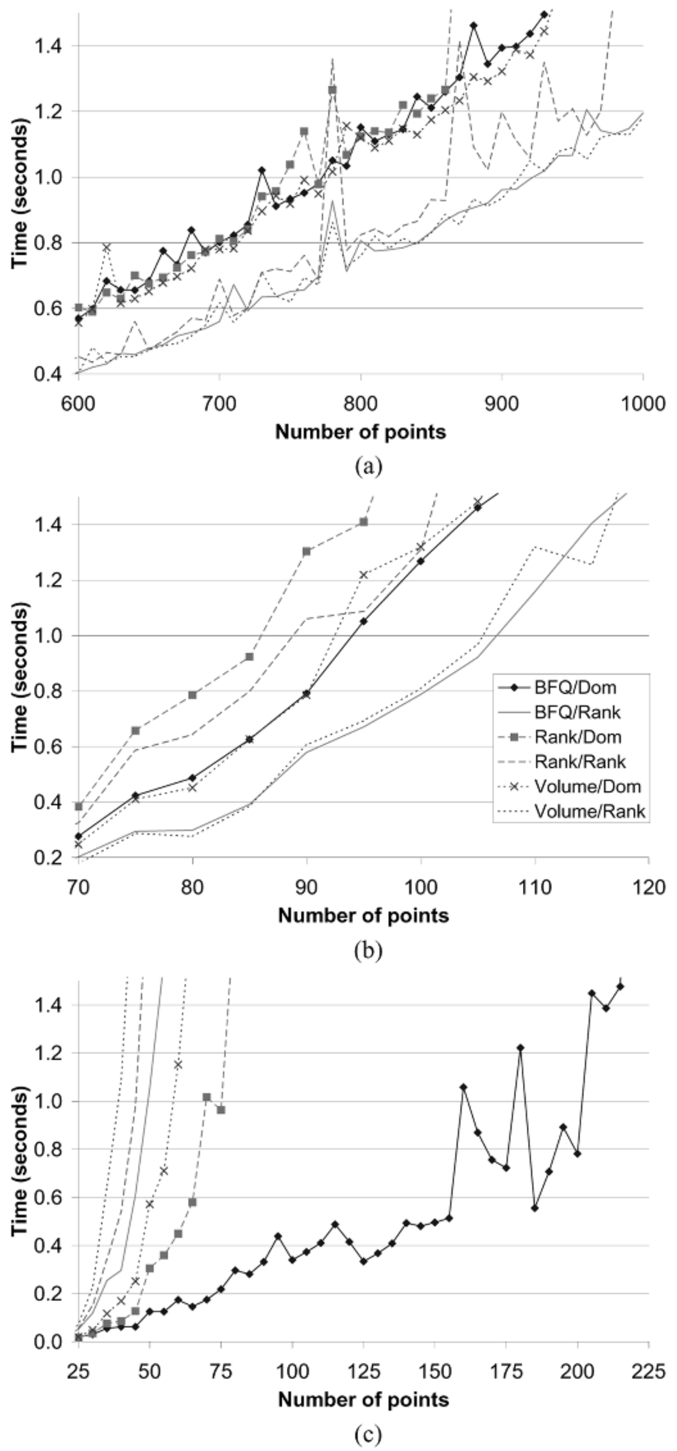


Fig. 9. Comparison of the performance of IHSO* with various heuristic combinations. Each line plots the average processing time for 200 distinct fronts in eight objectives. The legend on the middle graph applies for all three. (a) Random fronts in eight objectives. (b) Discontinuous fronts in eight objectives. (c) Spherical fronts in eight objectives.

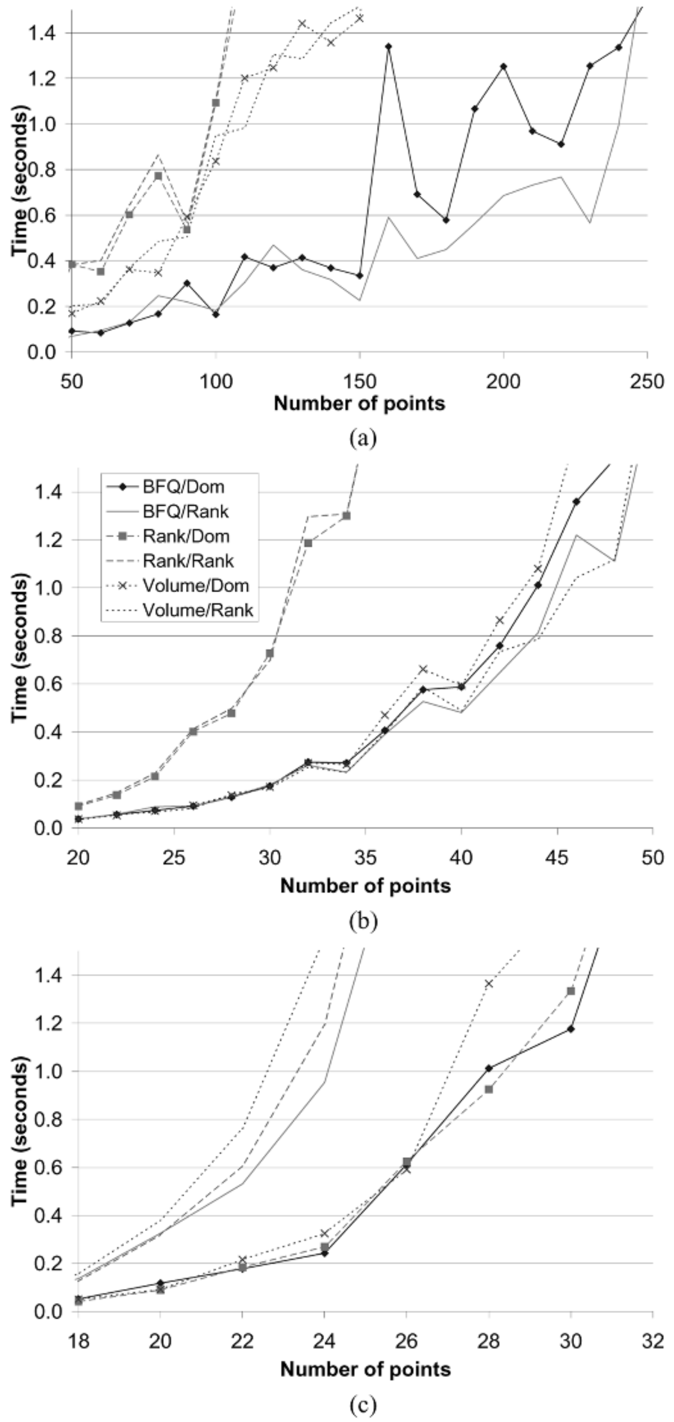


Fig. 10. Comparison of the performance of IHSO* with various heuristic combinations. Each line plots the average processing time for 200 distinct fronts in 11 objectives. The legend on the middle graph applies for all three. (a) Random fronts in 11 objectives. (b) Discontinuous fronts in 11 objectives. (c) Spherical fronts in 11 objectives.

reveals that a large proportion of the points are dominated early if two particular objectives are processed first. As such, we believe that spherical data does not very well represent real-world data. However, the dominated heuristic will, due to its nature, provide better results for exploitable real-world fronts, where many of the points contribute in only a small proportion of objectives. Additionally, the spherical data does point out a reason why the BFQ approach is superior to point-ordering heuristics. In all cases, BFQ commands a massive lead over the point heuristics which demonstrates the sensitivity of our point-ordering heuristics to front shapes. We take this as further evidence that the BFQ technique is more robust than point-ordering techniques.

TABLE I
TYPICAL SIZES OF FRONTS IN VARIOUS NUMBERS OF OBJECTIVES THAT IHSO* CAN PROCESS IN 1 s

n	Random	Discontinuous	Spherical
5	955	750	700
6	950	280	240
7	940	170	92
8	880	105	47
9	830	75	32
10	490	58	28
11	220	44	24
12	70	36	20
13	42	28	16

Table I shows what size of front optimized IHSO* (rank heuristic and BFQ) can process in 1 s, on average, for each front-type. Average processing time is the most important consideration, as IHSO will be called many times in a typical MOEA run.

C. How Does Performance Vary With the Data?

Figs. 8–10 only plot the average performance of the various algorithms as front size increases. We also performed a series of experiments to investigate how the performance of optimized IHSO* varies for a given front with the nature of a front.

Each graph in Fig. 11 plots a histogram showing the distribution of times needed to process 50 000 different fronts of the relevant type, and also the cumulative proportions of those fronts that are processed within a given time.

While the great majority of fronts are processed very quickly, there are cases where finding the least contributing point takes a disproportionate amount of time. Such outliers could be due to several factors. As the fronts become large, in some cases there are many points that contribute similar hypervolumes and their contribution may be difficult to calculate. For example, in the case where every point contributes the same hypervolume, neither the point-ordering nor BFQ techniques help performance.

Thus, while the average performance of IHSO* is extremely good, this performance cannot be guaranteed for complex fronts.

D. Does the Choice of Reference Point Affect Performance? and Does Scaling Objective Values Affect Performance?

Choice of reference point can significantly affect performance for the BFQ scheme. Fig. 12 illustrates these effects.

Each graph in Fig. 12 shows the performance of optimized IHSO* at m points in 9-D for the relevant front type, with two lines: the first plots time to determine the smallest point versus reference point offset, averaged over 200 fronts, and the second plots the same with all objectives scaled to $[0, 1]$. Keep in mind that reference point offsets are relatively "larger" for the scaled fronts, for example, compare point $(10, 10)$ with reference point of $(11, 11)$ to a point $(1, 1)$ with reference point $(2, 2)$. Therefore, scaled and unscaled fronts are not necessarily comparable for a given offset value.

The graphs show the observed results are due to several effects that result from a change in reference point. First, the choice of reference point influences which point has the smallest contribution. Second, regardless of whether a change in reference point changes which point is the smallest, a change in a point's contribution may also require further calculation of other points to prove that it is the smallest. Similar effects are caused by scaling objectives.

Although the reference point should not be chosen for performance criteria and rather so that the "best" points are retained,

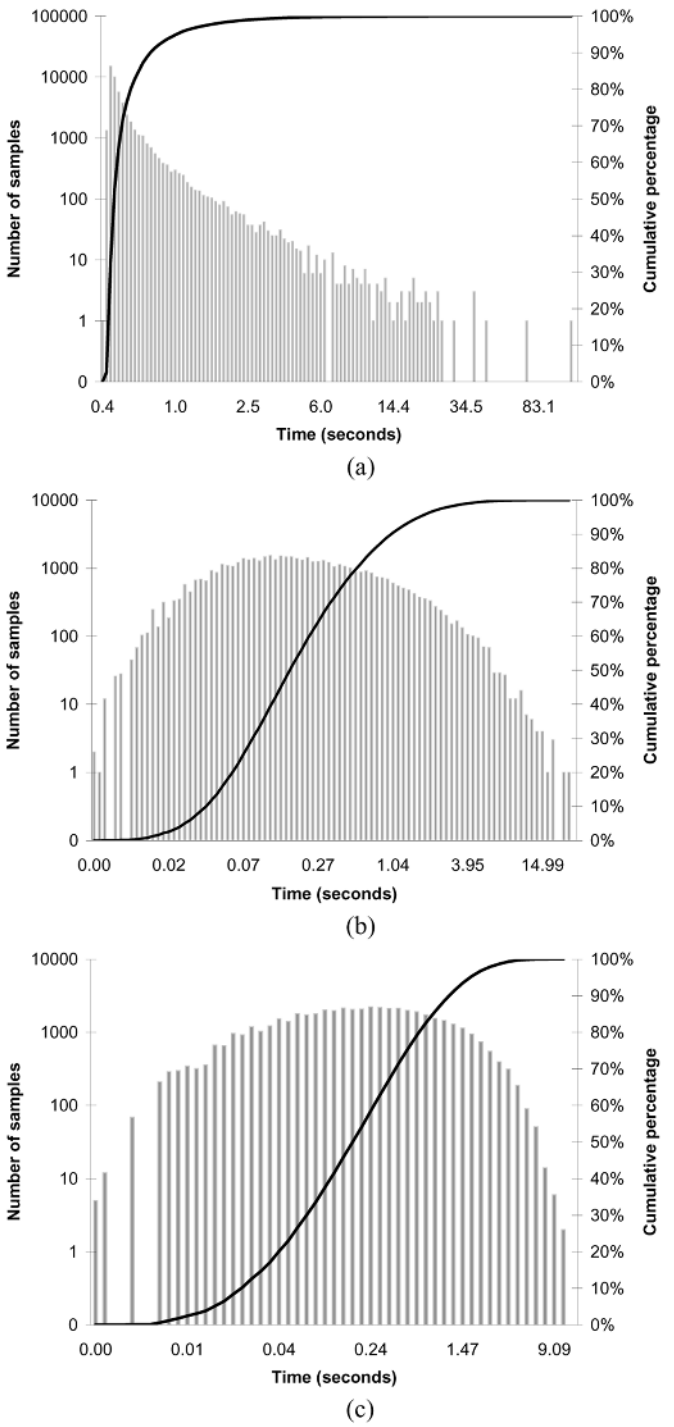


Fig. 11. Variation in processing time for optimized IHSO* for different fronts. Each graph plots a histogram of the (log-scale) processing times for 50 000 distinct fronts, and also the proportion of the fronts that were processed within a given time. (a) Random fronts: 650 points in nine objectives. (b) Discontinuous fronts: 65 points in nine objectives. (c) Spherical fronts: 30 points in nine objectives.

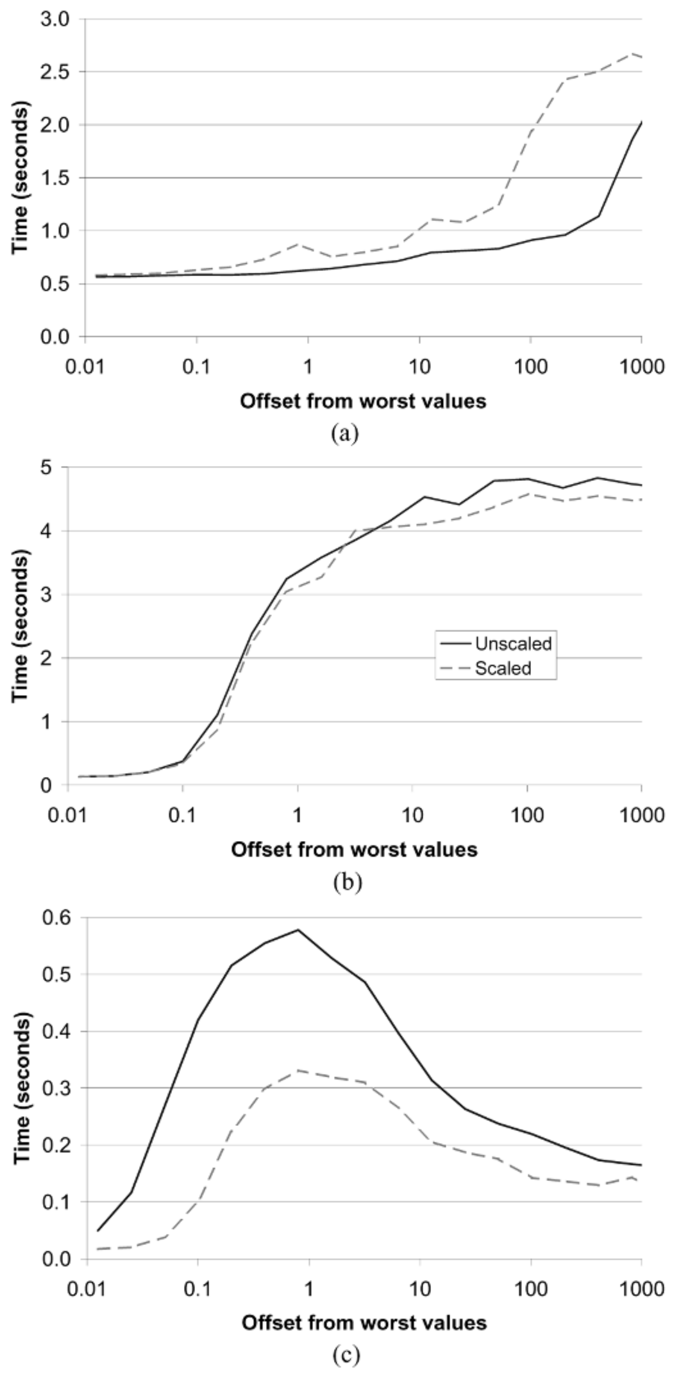


Fig. 12. Variation in processing time for optimized IHSO* for different reference points. Each graph plots the average (log-scale) processing time for 200 distinct fronts against the offset of the reference point from the worst value in each objective. The graphs show IHSO* applied to raw data, and to data scaled to [1] in each objective. The legend on the middle graph applies for all three. (a) Random fronts: 650 points in nine objectives. (b) Discontinuous fronts: 65 points in nine objectives. (c) Spherical fronts: 30 points in nine objectives.

the resulting effect on performance should be kept in mind when evaluating IHSO* on difficult fronts.

VI. Conclusion and Future Work

We have described a new algorithm for the calculation of incremental hypervolume when used within evolutionary algorithms, and techniques to apply this algorithm that minimize its cost. By applying heuristics to reorder objectives, we are able to increase the size of the fronts we are able to process. Additionally, by applying a best-first queueing approach we are able to calculate only as much of a point's hypervolume as is necessary to prove that it is not the smallest. We have demonstrated that, in general, this approach is superior to processing points using point reordering heuristics.

Through the combination of these techniques, we have described a method to effectively deal with very large numbers of points in many objectives within an EA. In doing so, the use of hypervolume should be computationally practical in tackling most complex real-world multiobjective problems. We recommend the BFQ strategy and the rank objective heuristic as a combination that performs well on a range of problems. However, better objective heuristics may exist for some particular front types.

Given the introduction of recent work on hypervolume for metric calculations, future work will look at adapting solutions to the Klee's measure problem to incremental hypervolume calculations. This would involve an adaptation of ideas from the Overmars and Yap algorithm to quickly perform incremental hypervolume calculations, and the application of our BFQ strategy for worst-point search. This new algorithm may also benefit from objective reordering heuristics similar to those described. Ideally, the combination of these works would allow the use of hypervolume within EA optimization to be not only practical but relatively inexpensive for all but the most difficult problems.

Acknowledgment

The authors would like to thank S. Huband for providing the raw DTLZ data, and K. Murray for advice on statistical issues.

References

[1] R. Purshouse, "On the evolutionary optimization of many objectives," Ph.D. dissertation, Univ. of Sheffield, Sheffield, U.K., 2003.
[2] E. Zitzler, "Evolutionary algorithms for multiobjective optimization: Methods and applications," Ph.D. dissertation, Swiss Federal Inst. Technol. (ETH, Zurich, Switzerland, 1999.
[3] M. Laumanns, E. Zitzler, and L. Thiele, "A unified model for multi-objective evolutionary algorithms with elitism," in *Proc. Congr. Evol. Comput.*, R. Eberhart, Ed., 2000, pp. 46–53.
[4] M. Fleischer and L. E. Thiele, "The measure of Pareto optima: Applications to multi-objective metaheuristics," in *Proc. Evol. Multi-objective Opt.*, C. M. Fonseca, P. J. Fleming, E. Zitzler, and K. Deb, Eds., 2003, vol. 2632, pp. 519–533.
[5] 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, pp. 117–132, Apr. 2003.
[6] M. Fleischer, The measure of Pareto optima: Applications to multi-objective metaheuristics Inst. Syst. Res., Univ. Maryland, College Park, MD, Tech. Rep. ISR TR 2002–32, 2002.
[7] E. Zitzler, "Hypervolume metric calculation." 2001 [Online]. Available: ftp://ftp.tik.ee.ethz.ch/pub/people/zitzler/hypervol.c
[8] J. Knowles, "Local-search and hybrid evolutionary algorithms for Pareto optimization," Ph.D., Univ. Reading, Reading, U.K., 2002.
[9] L. While, P. Hingston, L. Barone, and S. Huband, "A faster algorithm for calculating hypervolume," *IEEE Trans. Evol. Comput.*, vol. 10, pp. 29–38, Feb. 2006.
[10] 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.*, B. McKay, Ed., 2005, pp. 2225–2232.

[11] N. Beume and G. Rudolph, "Faster s-metric calculation by considering dominated hypervolume as Klee's measure problem," Univ. Dortmund, Dortmund, Germany, Tech. Rep. CI 216/06, 2006.

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

[13] C. M. Fonseca, L. Paquete, and M. López-Ibáñez, "An improved dimension-sweep algorithm for the hypervolume indicator," in *Proc. Congr. Evol. Comput.*, C. L. P. Chen, Ed., 2006, pp. 3973–3979.

[14] S. Huband, P. Hingston, L. While, and L. Barone, "An evolution strategy with probabilistic mutation for multi-objective optimization," in *Proc. Congr. Evol. Comput.*, H. Abbass and B. Verma, Eds., 2003, pp. 2284–2291.

[15] J. Knowles, D. Corne, and M. Fleischer, "Bounded archiving using the Lebesgue measure," in *Proc. Congr. Evol. Comput.*, H. Abbass and B. Verma, Eds., 2003, pp. 2490–2497.

[16] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *Proc. Parallel Problem Solving from Nature VIII*, X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervos, J. A. Bullinaria, J. Rowe, P. Tino, A. Kaban, and H. Schwefel, Eds., 2004, vol. 3242, pp. 832–842.

[17] M. Emmerich, N. Beume, and B. Noujoks, "An EMO algorithm using the hypervolume measure as selection criterion," in *Proc. Evol. Multi-Objective Opt.*, M. Emmerich, N. Beume, B. Naujoks, C. A. C. Coello, A. H. Aguirre, and E. Zitzler, Eds., 2005, vol. 3410, pp. 62–76.

[18] , T. Bäck, Ed., *Handbook of Evolutionary Computation*. Oxford, U.K.: Oxford Univ. Press, 1997.

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

[20] L. Bradstreet, L. Barone, and L. While, "Maximizing hypervolume for selection in multi-objective evolutionary algorithms," in *Proc. Congr. Evol. Comput.*, C. L. P. Chen, Ed., 2006, pp. 1744–1751.

[21] L. Bradstreet, L. Barone, and L. While, "Incrementally maximizing hypervolume for selection in multi-objective evolutionary algorithms," in *Proc. Congr. Evol. Comput.*, A. Tay, Ed., 2007, pp. 3203–3210.

[22] Hypervolume Test Data "Walking fish group", 2006. [Online]. Available: http://wfg.csse.uwa.edu.au/Hypervolume

Lucas Bradstreet (S'06) received a BCM degree in computer and mathematical sciences from the University of Western Australia, Nedlands, in 2004. He is currently working towards the Ph.D. degree at the School of Computer Science and Software Engineering, University of Western Australia.

His research interests include multiobjective evolutionary algorithms and metrics and benchmarks for assessing their performance.

Lyndon While (M'01–SM'04) received the B.Sc.(Eng) 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, University of Western Australia. His research interests include evolutionary algorithms, multiobjective optimization, and the semantics and implementation of functional programming languages.

Luigi Barone (M'01) received the B.Sc. and Ph.D. degrees from the University of Western Australia, Nedlands, in 1994 and 2004, respectively.

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