CSSE Technical Report UWA-CSSE-07-001
February 2007
A Fast Incremental Hypervolume Algorithm

Lucas Bradstreet, Lyndon While, and Luigi Barone
School of Computer Science & Software Engineering
The University of Western Australia
Crawley WA 6009 Australia
email: {lucas, lyndon, luigi}@csse.uwa.edu.au
tel: +61 8 6488 2720
fax: +61 8 6488 1089

February 27, 2007

Abstract

When hypervolume is used as part of the selection or archiving process in a multi-objective evolutionary algorithm (MOEA), 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 heuristics that re-order objectives to minimise the work required for IHSO to calculate a solution's contribution. Lastly, we describe and analyse search techniques that reduce the hypervolume calculations required for solutions other than the least contributing solution. Combined, these techniques allow MOEAs to calculate hypervolume in-line in increasingly complex and large fronts in many objectives.

Keywords: Multi-objective optimisation, evolutionary computation, diversity, performance metrics, hypervolume.

1 Introduction

Hypervolume [14], also known as the S-metric [18] or the Lebesgue measure [7, 12], has recently been finding favour as a metric for comparing the performance of multi-objective evolutionary algorithms. 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 favoured 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.* [21] 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 maximised if and only if the set of solutions contains only Pareto optima. Hypervolume has some non-ideal 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) [10, 19]. HSO works by processing the objectives in a front, rather than the points. It divides the nD-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.* have described [16] good heuristics that optimise 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 [2] adapt the Overmars and Yap [13] algorithm for solving the Klee's measure problem to instead calculate the hypervolume of a front. Similarly, Fonseca *et al.* [8] apply the Overmars and Yap algorithm for the 3D 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 in-line in some evolutionary algorithms, as part of a diversity mechanism [9], as part of an archiving mechanism [11], or recently as part of the selection mechanism [5, 20]. 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 minimises the expense of repeated invocations.

The principal contributions of this paper are a version of HSO which is customised for in-line incremental hypervolume calculations, and search techniques and heuristics that improve performance for hypervolume algorithms used within a MOEA.

The customised 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 optimisations in IHSO are minimising 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 optimisations 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 in-line incremental hypervolume calculations. We note that although Beume and Rudolph's recent work [2] does improve the complexity of hypervolume algorithms for metric calculations, customised algorithms are not yet available for incremental hypervolume calculations.

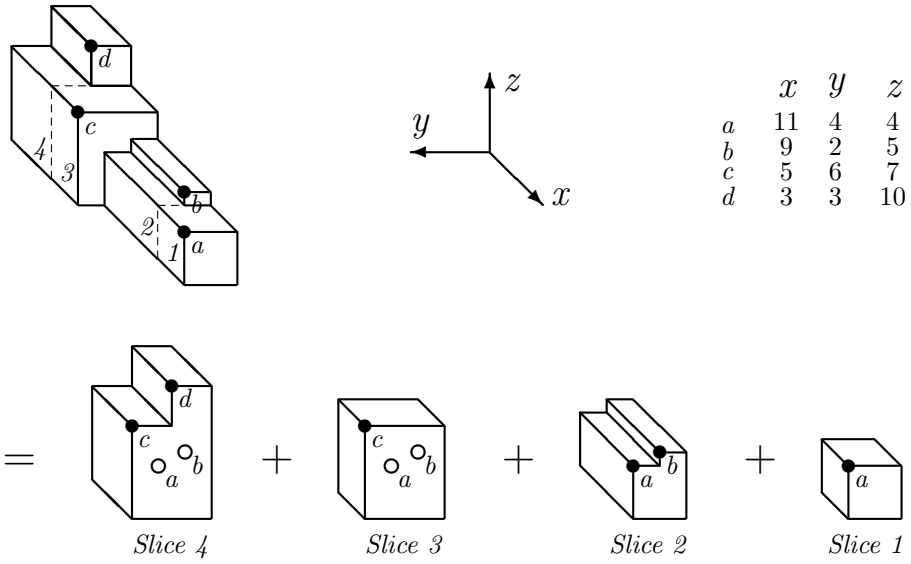
The rest of this paper is structured as follows. Section 2 defines the concepts and notation used in multi-objective optimisation and throughout this paper. Section 3 describes HSO and the heuristics used to optimise its performance. Section 4 describes how HSO can be customised into IHSO and IHSO* to calculate exclusive hypervolume efficiently. Section 5 reports on some experiments to determine the fastest incremental algorithm, and to illustrate some important issues for users of the algorithm. Section 6 concludes the paper and discusses some possibilities for future work.

2 Definitions

In a multi-objective optimisation problem, we aim to find the set of optimal trade-off solutions known as the Pareto optimal set. Pareto optimality is defined with respect to the concept of non-domination 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 *non-dominated* with respect to a set of solutions X iff there is no vector in X that dominates $\overline{x}$. X is a *non-dominated set* iff all vectors in X are mutually non-dominating. Such a set of objective vectors is sometimes called a *non-dominated front*.

A vector $\overline{x}$ is *Pareto optimal* iff $\overline{x}$ is non-dominated with respect to the set of all possible vectors. Pareto optimal vectors are characterised 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 multi-objective 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



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

Figure 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 labelled with letters: unfilled circles represent points that are dominated in y and z. Slices are labelled with numbers, and are separated on the main picture by dashed lines. (Figure reproduced from [17].)

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 [1].

3 Hypervolume by Slicing Objectives

Given a set of mutually non-dominating 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 k^{th} 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 re-sorted in the next objective, and newly-dominated points within each slice are discarded.

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 non-dominated 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 pseudo-code for HSO.

3.1 The Complexity and Performance of HSO

The following recurrence relation captures the worst-case complexity of HSO [17].

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

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

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.

```
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
```

Figure 2: Pseudo-code for HSO. (Code reproduced from [16].)

Solving this recurrence relation gives the following [17].

$$f(m,n) = \begin{pmatrix} m+n-2 \\ n-1 \end{pmatrix} \tag{3}$$

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 maximising 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 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 maximisation problem.

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

Figure 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 generalised for other numbers of points. (Example reproduced from [16].)

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: e.g. 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.

3.2 Optimising 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 [16]. They characterise the better heuristic as "minimising the amount of worst-case work" (MWW). For each objective, MWW

- calculates the number of non-dominated 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 vs. HSO+MWW on randomly-generated fronts and on fronts from the well-known DTLZ test suite [4] shows that MWW can reduce the time to process fronts in 5–9 objectives by 25–98%.

4 Running HSO Incrementally

Hypervolume is used in-line in an evolutionary algorithm in three ways:

- as part of a diversity mechanism,
- as part of an archiving mechanism,
- or 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

$$ExcHyp(p, S) = Hyp(S \cup \{p\}) - Hyp(S) \tag{4}$$

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 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 [3], 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 customising HSO to calculate exclusive hypervolumes directly. We define a new algorithm *Incremental HSO* (IHSO) that takes a point p and a set of mutually non-dominating points S and returns $ExcHyp(p, S)$. We customise 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 optimise the performance of IHSO by selecting a good order in which to process the objectives.

Fig. 4 gives pseudo-code 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]`.

- 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 $ExcHyp(p, S - \{p\})$ 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 likely small points are 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*.

```
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
```

Figure 4: Pseudo-code for IHSO. Functions not defined here are defined in Fig. 2. The re-ordering of the objectives is not shown.

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 process first 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 3.2.

While these heuristics can be used to reorder objectives for all calculations, as in [16], 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, e.g. because of a difficult shape, or 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.

How do we order the points when calculating their exclusive hypervolumes in IHSO*?

This scheme is outlined in Fig. 5.

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

Figure 5: Outline of point ordering scheme in IHSO*.

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

1. *Rank*: the sum of the numbers of points that beat p in each objective.
2. *Volume*: the "inclusive hypervolume" of p: the product of its objectives.

We have also defined a "best-first" queuing scheme that processes the points "concurrently", to avoid the question of ordering. This scheme is outlined in Fig. 6.

```
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
```

Figure 6: Outline of the best-first queuing scheme in IHSO*.

The principal parameter in the queuing scheme is the definition of "a bit", i.e. the granularity of the concurrency. If the granularity is too big, the algorithm will do more calculation than necessary: if it is too small, 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. This dimensionality is set according to (5).

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

The application of max in (5) means that for low dimensionalities, we abandon the queuing 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.

Section 5 describes an empirical comparison of the performance of these methods.

5 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, initialised by generating points with random values x, $0.1 \leq x \leq 10$, in all objectives. In order to guarantee mutual non-domination, we initialised $S = \phi$ and added each point $\overline{x}$ to S only if $\overline{x} \cup S$ would be mutually-non-dominating.
- We used the discontinuous and spherical fronts from the DTLZ test suite [4]. 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 [16, 17], 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 [15].

All timings were performed on a dedicated 2.8GHz Pentium IV machine with 512Mb 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.

5.1 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 thirty points in nine objectives. Each line is a 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 processing the points in the right order can make a huge difference to the performance of the algorithm. Typically, the best order is processed 300–6,000 times faster than the worst order, and 60–200 times faster than the median order.

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

5.2 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, 9, and 10 shows the performance for varying front-sizes of the six combinations of the following heuristics from Section 4.

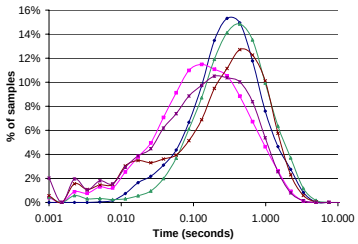
- *Point-ordering:* Rank, Volume, and the best-first queuing scheme.
- *Objective-ordering:* Rank and Dominated.

Other heuristics discussed in Section 4 performed consistently worse than those illustrated in the figures. The graphs plot fronts up to 1,000 points that can be processed in 1.5 seconds: this should be a reasonable amount of time for an incremental hypervolume calculation for most applications.

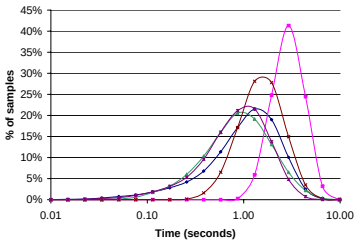
Figs. 8, 9, and 10 show that using the BFS approach gives the best results other than in 5 objectives. Although BFS loses slightly in 5 objectives, this is probably as a result of the overhead caused by the priority queue. If point reordering algorithms make perfect decisions they will always outperform the BFS 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 BFS introduces a greater uncertainty in the results. A bad point ordering decision could, in the worst case, require the calculation of every point's entire exclusive hypervolume. This effect is evident when comparing the BFS/rank algorithm to the rank/rank algorithm for random, 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, BFS/rank compares favourably to or beats all other objective heuristic and point ordering techniques. This dominance increases with the number of objectives.

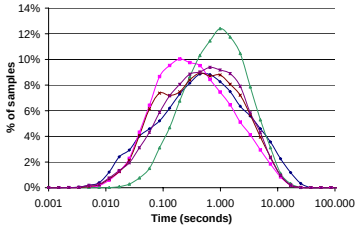
For spherical data, the dominated heuristic performs extremely well. However, we believe spherical data is being especially exploited by this heuristic. Examination of the data reveals that a large proportion of the points are dominated early on if two particular objectives are processed first. As such, 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



(a) Random fronts: 30 points in 9 objectives.



(b) Discontinuous fronts: 30 points in 9 objectives.



(c) Spherical fronts: 30 points in 9 objectives.

Figure 7: Variation in processing time for IHSO* for different point-orderings. Each line represents a histogram of the processing times for 50,000 distinct orderings for one front.

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 BFS approach is superior to point ordering heuristics. In all cases BFS 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 BFS technique is more robust than point ordering techniques.

Table 1 shows what size of front optimised IHSO* can process in one second for each front-type.

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 1: Sizes of fronts in various numbers of objectives that IHSO* can process in one second.

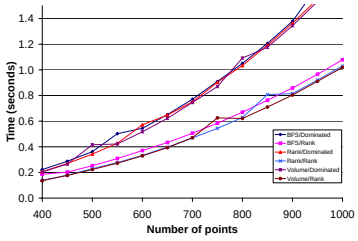
5.3 How does performance vary with the data?

Figs. 8, 9, and 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 optimised IHSO* varies for a given front with the nature of a front.

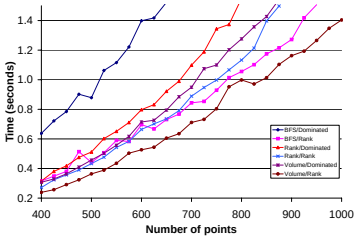
Each graph in Fig. 11 plots a histogram showing the distribution of times needed to process 50,000 different fronts of the relevant type, and also the cumulative proportions of those fronts that are processed within a given time.

While the great majority of fronts are processed within a reasonable amount of time, there are cases where finding the least contributing point takes a disproportionate amount of time.

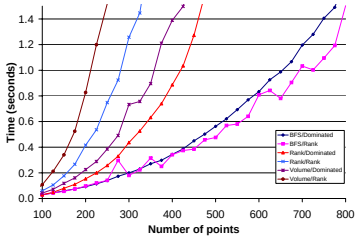
Such outliers could be due to several factors. As the fronts become large, in some cases there are many points that contribute similar hypervolumes and their contribution may be difficult to calculate. For example, in the case where every point contributes the same hypervolume neither the point ordering or BFS techniques help performance. While these fronts do exist, the average case performance of IHSO* is extremely good. However this performance cannot be guaranteed given complex front types or difficult decisions.



(a) Random fronts in 5 objectives.

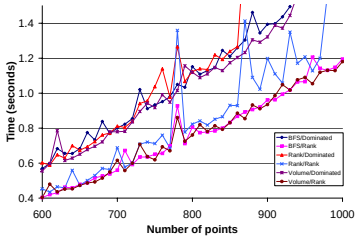


(b) Discontinuous fronts in 5 objectives.

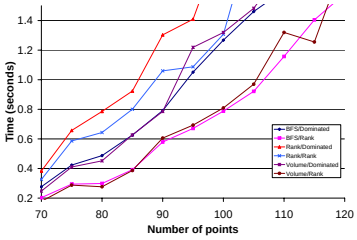


(c) Spherical fronts in 5 objectives.

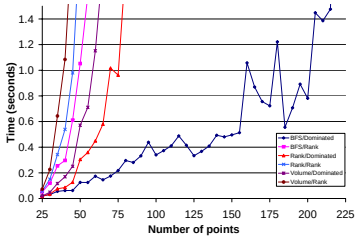
Figure 8: Comparison of the performance of IHSO* with various heuristic combinations. Each line plots the average processing time for 200 distinct fronts in 5 objectives.



(a) Random fronts in 8 objectives.

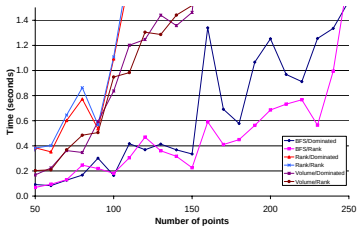


(b) Discontinuous fronts in 8 objectives.

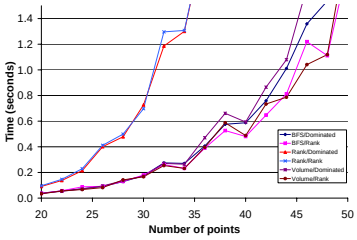


(c) Spherical fronts in 8 objectives.

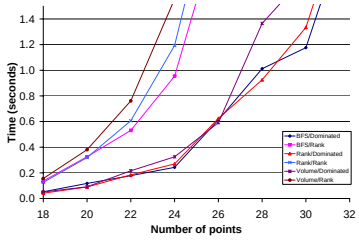
Figure 9: Comparison of the performance of IHSO* with various heuristic combinations. Each line plots the average processing time for 200 distinct fronts in 8 objectives.



(a) Random fronts in 11 objectives.

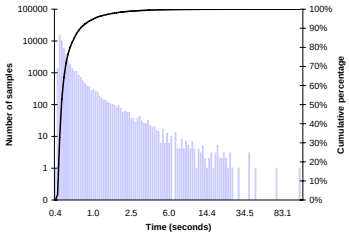


(b) Discontinuous fronts in 11 objectives.

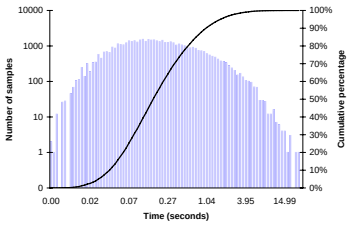


(c) Spherical fronts in 11 objectives.

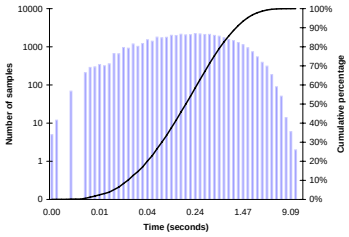
Figure 10: Comparison of the performance of IHSO* with various heuristic combinations. Each line plots the average processing time for 200 distinct fronts in 11 objectives.



(a) Random fronts: 650 points in 9 objectives.



(b) Discontinuous fronts: 65 points in 9 objectives.



(c) Spherical fronts: 30 points in 9 objectives.

Figure 11: Variation in processing time for optimised IHSO* for different fronts. Each graph plots a histogram of the processing times for 50,000 distinct fronts, and also the proportion of the fronts that were processed within a given time.

5.4 Does the choice of reference point affect performance? And does scaling objective values affect performance?

Choice of reference point does affect performance for the best-first search scheme. We demonstrate these effects in Fig. 12.

Each graph in Fig. 12 shows BFS/rank at m points in 9D for the relevant front type, with two lines: the first plots time to determine the smallest point vs. 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, e.g. 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.

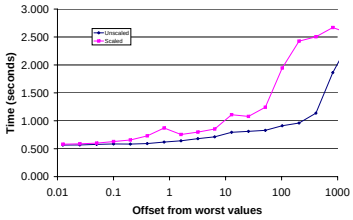
The graphs show the observed results are due to several effects that result from a change in reference point. Firstly, the choice of reference point influences which point has the smallest contribution. Secondly, regardless of whether a change in reference point changes which point is the smallest, a change in a point's contribution may also require further calculation of other points to prove that it is the smallest. These effects are also caused by scaling objectives for similar reasons. Although the reference point should not be chosen for performance criteria and rather so that the 'best' points are retained, the resulting effect on performance should be kept in mind when evaluating IHSO* on difficult fronts.

6 Conclusions and Future Work

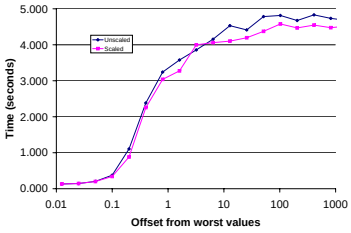
We have described a new algorithm for the calculation of incremental hypervolume when used within evolutionary algorithms, and techniques to apply this algorithm that minimise its cost. By applying heuristics to re-order objectives we are able to increase the size of the fronts we are able to process. Additionally, by applying a best first search 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 multi-objective problems. We recommend the BFS 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.

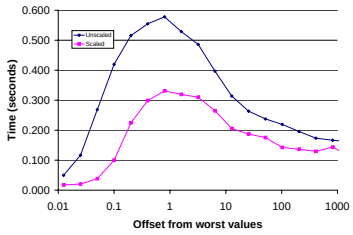
Given the introduction of recent work on hypervolume for metric calculations, future work will look at adapting solutions to the Klee's measure problem to incremental hypervolume calculations. This would involve an adaptation of ideas from the Overmars and Yap algorithm to quickly perform incremental hypervolume calculations, and the application of our BFS strategy for worst-point search. This new algorithm may also benefit from objective reordering heuris-



(a) Random fronts: 650 points in 9 objectives.



(b) Discontinuous fronts: 65 points in 9 objectives.



(c) Spherical fronts: 30 points in 9 objectives.

Figure 12: Variation in processing time for optimised IHSO* for different reference points. Each graph plots the average 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 $[0, 1]$ in each objective.

tics similar to those described. Ideally, the combination of these works would allow the use of hypervolume within EA optimisation to be not only practical but relatively inexpensive for all but the most difficult problems.

Acknowledgments

We thank Simon Huband for providing the raw DTLZ data, and Kevin Murray for advice on statistical issues.

References

[1] Thomas Bäck, David Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Inst of Physics Publishing and Oxford University Press, 1997.

[2] Nicola Beume and Günter Rudolph. Faster s-metric calculation by considering dominated hypervolume as klee's measure problem. Technical Report CI 216/06, University of Dortmund, 2006.

[3] L. Bradstreet, L. Barone, and L. While. Maximising hypervolume for selection in multi-objective evolutionary algorithms. In *Congress on Evolutionary Computation 2006*. IEEE, 2006.

[4] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable multi-objective optimization test problems. In D. B. Fogel *et al.*, editor, *Congress on Evolutionary Computation 2002*, volume 1, pages 825–830. IEEE, 2002.

[5] M. Emmerich, N. Beume, and B. Naujoks. An emo algorithm using the hypervolume measure as selection criterion. In C. Coello Coello *et al.*, editor, *Evolutionary Multi-objective Optimisation 2005*, volume 3410 of *LNCS*, pages 62–76. Springer-Verlag, 2005.

[6] M. Fleischer. The measure of Pareto optima: Applications to multi-objective metaheuristics. Technical Report ISR TR 2002-32, Institute for Systems Research, University of Maryland, 2002.

[7] M. Fleischer. The measure of Pareto optima: Applications to multi-objective metaheuristics. In C. M. Fonseca *et al.*, editor, *Evolutionary Multi-objective Optimisation 2003*, volume 2632 of *LNCS*, pages 519–533. Springer-Verlag, 2003.

[8] Carlos M. Fonseca, Luís Paquete, and Manuel López-Ibáñez. An improved dimension-sweep algorithm for the hypervolume indicator. In *IEEE Congress on Evolutionary Computation*, pages 3973–3979, Vancouver, Canada, July 2006.

[9] S. Huband, P. Hingston, L. While, and L. Barone. An evolution strategy with probabilistic mutation for multi-objective optimization. In Hussein Abbass and Brijesh Verma, editors, *Congress on Evolutionary Computation 2003*, volume 4, pages 2284–2291. IEEE, 2003.

[10] J. Knowles. *Local-search and Hybrid Evolutionary Algorithms for Pareto Optimisation*. PhD thesis, The University of Reading, 2002.

[11] J. Knowles, D. Corne, and M. Fleischer. Bounded archiving using the Lebesgue measure. In Hussein Abbass and Brijesh Verma, editors, *Congress on Evolutionary Computation 2003*, volume 4, pages 2490–2497. IEEE, 2003.

[12] M. Laumanns, E. Zitzler, and L. Thiele. A unified model for multi-objective evolutionary algorithms with elitism. In *Congress on Evolutionary Computation 2000*, volume 1, pages 46–53. IEEE, 2000.

[13] Mark H. Overmars and Chee-Keng Yap. New upper bounds in klee's measure problem (extended abstract). In *IEEE Symposium on Foundations of Computer Science*, pages 550–556, 1988.

[14] R. Purshouse. *On the Evolutionary Optimisation of Many Objectives*. PhD thesis, The University of Sheffield, Sheffield, UK, 2003.

[15] Walking Fish Group. Hypervolume test data. 2006.

[16] L. While, L. Bradstreet, L. Barone, and P. Hingston. Heuristics for optimising the calculation of hypervolume for multi-objective optimisation problems. In *Congress on Evolutionary Computation 2005*, pages 2225–2232. IEEE, 2005.

[17] L. While, P. Hingston, L. Barone, and S. Huband. A faster algorithm for calculating hypervolume. *IEEE Transactions on Evolutionary Computation*, 2005.

[18] E. Zitzler. *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*. PhD thesis, Swiss Federal Inst of Technology (ETH) Zurich, 1999.

[19] E. Zitzler. Hypervolume metric calculation. *ftp://ftp.tik.ee.ethz.ch/pub/people/zitzler/hypervol.c*, 2001.

[20] Eckart Zitzler and Simon Künzli. Indicator-based Selection in Multiobjective Search. In Xin Yao et al., editor, *Parallel Problem Solving from Nature - PPSN VIII*, pages 832–842, Birmingham, UK, September 2004. Springer-Verlag. Lecture Notes in Computer Science Vol. 3242.

[21] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, April 2003.