2005

Heuristics for Optimising the Calculation of Hypervolume for Multi-objective Optimisation Problems

Lyndon While
University of Western Australia

Lucas Bradstreet
University of Western Australia

Luigi Barone
University of Western Australia

Philip Hingston
Edith Cowan University

Heuristics for Optimising the Calculation of Hypervolume for Multi-objective Optimisation Problems

Lyndon While, Lucas Bradstreet, Luigi Barone
The University of Western Australia
Nedlands, Western Australia 6009
{lyndon, lucas, luigi}@csse.uwa.edu.au

Phil Hingston
Edith Cowan University
Mount Lawley, Western Australia 6050
p.hingston@ecu.edu.au

Abstract- The fastest known algorithm for calculating the hypervolume of a set of solutions to a multi-objective optimisation problem is the HSO algorithm (Hypervolume by Slicing Objectives). However, the performance of HSO for a given front varies a lot depending on the order in which it processes the objectives in that front. We present and evaluate two alternative heuristics that each attempt to identify a good order for processing the objectives of a given front. We show that both heuristics make a substantial difference to the performance of HSO for randomly-generated and benchmark data in 5–9 objectives, and that they both enable HSO to reliably avoid the worst-case performance for those fronts. The enhanced HSO will enable the use of hypervolume with larger populations in more objectives.

1 Introduction

Multi-objective optimisation problems abound, and many evolutionary algorithms have been proposed to derive good solutions for such problems, e.g. [1, 2, 3, 4, 5]. However, the question of what metrics to use in comparing the performance of these algorithms remains difficult[6, 7, 1]. One metric that has been favoured by many people is *hypervolume*[8], also known as the S-metric[9] or the Lebesgue measure[10]. 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[11, 12]. Hypervolume has some non-ideal properties too: it requires the (sometimes arbitrary) definition of a reference point on which its calculations are based, and it is sensitive to the relative scaling of the objectives, and to the presence or absence of extremal points in a front.

While *et al.*[13] have shown that the fastest known algorithm for calculating hypervolume exactly is HSO (Hypervolume by Slicing Objectives)[14, 15]. 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 it still easily outperforms all other known algorithms for calculating hypervolume exactly[13, 16].

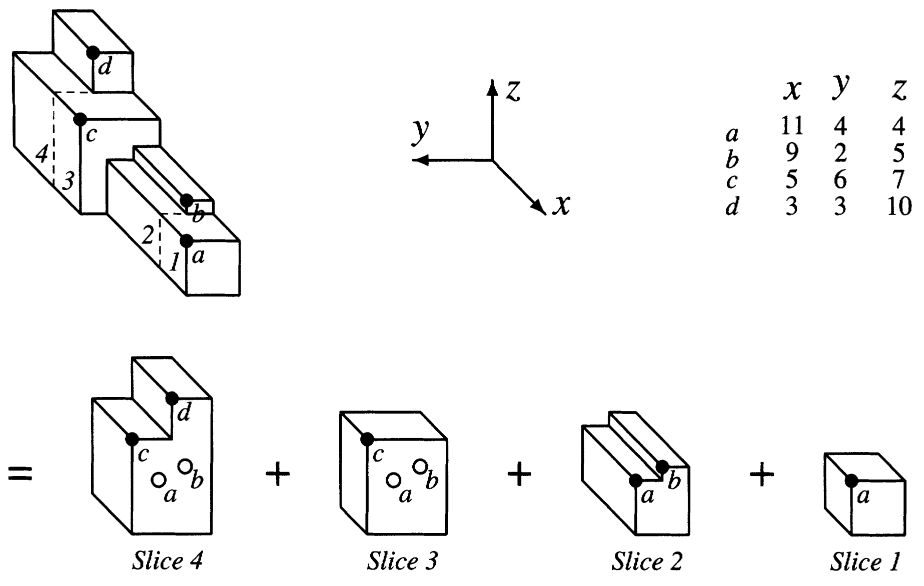
However, the performance of HSO for a given front depends on the order in which it processes the objectives in that front. The number of points contributing hypervolume to each $n-1$D-slice depends on how many points are dominated within that slice: more dominated points implies a smaller set of points to process, which implies less work for that slice. The principal contribution of this paper is the presentation and evaluation of two alternative heuristics that each enhance HSO by trying to select a good order in which to process the objectives for a given front. We present performance data for basic and enhanced HSO showing that both heuristics make a substantial difference to the typical performance of the algorithm. The enhanced HSO will enable the use of hypervolume with larger populations in more objectives.

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 the operation of HSO, and Section 4 discusses its complexity and performance and shows how heuristics might help. Section 5 defines our two (alternative) heuristics, and Section 6 gives empirical data for a range of randomly-generated and benchmark fronts in 5–9 objectives showing how the heuristics improve the performance of HSO. Section 7 concludes the paper and outlines some future work.

2 Fundamentals

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.



	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.

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 [17].

3 The HSO Algorithm

Given m mutually non-dominating points in n objectives, the HSO algorithm is based on the idea of processing the set of 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. Because the points are sorted, they can be allocated to the slices easily. The top slice can contain only the point with the best value in the first objective; the second slice can contain only the points with the two best values; the third slice can contain only the points with the three best 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 re-sorted in the next objective, and newly-dominated points within each slice are discarded.

Figure 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 most natural base case for HSO is when the points are reduced to one objective, 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 the points are reduced to two objectives, which is an easy and fast special case.

Figure 2 gives pseudo-code for HSO. Note that, for exposition purposes, the function hso builds explicitly a set containing the slices to be processed after each iteration. We can improve the performance of the algorithm by processing these slices on-the-fly, as they are generated.

4 The Complexity and Performance of HSO

While *et al.*[13] give a recurrence relation that captures the worst-case complexity of HSO:

$$f(m,1) = 1 \quad (1)$$

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

```
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.

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.

Furthermore, While *et al.*[13] solve this recurrence relation to give the following identity:

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

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 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 Figure 3, in a maximisation problem.

5	⋯	5	1
4	⋯	4	2
3	⋯	3	3
2	⋯	2	4
1	⋯	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.

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 Figure 3, the earlier the last objective is processed, the faster the hypervolume will be calculated.

Thus it seems that enhancing HSO with a mechanism to help the algorithm to identify a good order in which to process the objectives in a given front could make a substantial difference to the real performance of the algorithm.

5 Heuristics

We present and evaluate two alternative heuristics that attempt to derive a good order for HSO to process the objectives in a given front.

5.1 Maximising the number of dominated points

A good order for the objectives is one in which many partial points are dominated by other points early in the process. One obvious tactic then is to calculate for each objective how many points will be dominated immediately if that objective is processed, and to process first the objective that will generate the most dominated points. We call this heuristic MDP: Maximising the number of Dominated Points.

We can apply this idea in two ways.

- We can simply calculate the heuristic once, then sort the objectives in decreasing order of numbers of dominated points.
- Alternatively, we can calculate the heuristic once, eliminate the best objective, then re-calculate the heuristic to identify the next objective, and so on, until all the objectives have been ordered.

Our experience shows that applying the heuristic iteratively works better, especially for large numbers of objectives, but that diminishing returns apply to some extent. We therefore iterate until four objectives remain, at which point we order those four objectives according to the last calculation.

The complexity of MDP is easy to calculate: at each iteration, for each objective, we (nominally) compare each point with every other point for domination. Thus for m points in n objectives, each iteration of MDP has complexity $O(m^2n^2)$, and applying MDP iteratively has complexity $O(m^2n^3)$, While this may sound expensive, remember that HSO is exponential in n in the worst case, so a good polynomial-time heuristic is likely to pay large dividends.

5.2 Minimising the amount of worst-case work

For each objective, MDP effectively counts the number of points that will contribute to the bottom $n-1$D-slice of the hypervolume. However, in some cases, this number might be misleading: it is theoretically possible to generate m slices where the first $m-1$ slices contain respectively $1, 2, \ldots, m-1$ points, but the bottom slice contains only 1 point. Example data that exhibits this behaviour is given in Figure 4, for a maximisation problem.

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

Figure 4: A pathological example for MDP. MDP will choose to process the first objective, but processing the second objective would be faster.

We can avoid this possibility with a slightly more involved heuristic that calculates explicitly for each objective the number of non-dominated partial points in each slice, estimates the amount of work required to process each slice, and sums these values to estimate the amount of work required if HSO processes that objective first. This heuristic effectively models the recurrence relation in (2), by summing the work required to process each slice individually. For each slice, we use the worst-case complexity of HSO given in (3) to estimate the work required to process that slice. Thus we call this heuristic MWW: Minimising the Worst-case Work.

Again, we can apply this idea once only, or iteratively, and again, our experience shows that iteration works better. As with MDP, we apply MWW iteratively until four objectives remain, at which point we order those four objectives according to the last calculation.

The complexity of MWW is similar to that of MDP. For each objective, we sort the points in that objective, then we build incrementally the sets of points in each slice, much as in the functions `slice` and `insert` in Figure 2. This leads to the worst-case complexity for each iteration being $O(n(m \log m + m^2 n))$, which again simplifies to $O(m^2n^2)$. The need to maintain an explicit set of non-dominated points during the calculation of MWW may make it more expensive than MDP in some cases, although any difference is likely to be small.

6 Empirical Performance Data

We evaluated the performance of the two heuristics vs. basic HSO on two different types of data: randomly-generated fronts, and samples taken from the four distinct Pareto optimal fronts of the problems in the well-known DTLZ test suite[18].

We evaluated the heuristics (mostly) on data in 5–9 objectives, so to estimate the best-, average-, and worst-case timings for each front using basic HSO, we used the following procedure.

For $n \leq 5$: we evaluated all $n!$ permutations of the objectives.

For $n > 5$: we sampled the $n!$ permutations in two ways, and we combined all of the results in the calculations.

- We evaluated all $n(n-1)$ permutations of the first two objectives (with the remaining objectives randomised).
- Additionally, we evaluated 120 randomly-chosen permutations.

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. The data used in the experiments are available at

`http://wfg.csse.uwa.edu.au/Hypervolume`

6.1 Benchmark data

We evaluated the heuristics on the four distinct fronts from the 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. Each hypervolume was calculated as a minimisation problem in every objective, relative to the point $1 \cdots 1$.

Tables 1(a)–1(c) and Figures 5(a)–5(d) and 6 show the resulting comparisons. Each row of each table is based on runs with ten different fronts, and it gives the following data.

- For HSO:

  wrst is the longest time for any run on any front.

  awst is the average of the longest time for each front.

		basic HSO					HSO+MDP			HSO+MWW		
n	m	$wrst$	$awst$	$avrg$	$abst$	$best$	$wrst$	$avrg$	$best$	$wrst$	$avrg$	$best$
5	800	71.45	61.39	**11.91**	1.18	1.02	1.64	**1.30**	1.13	1.67	**1.31**	1.13
6	230	84.68	68.79	**10.50**	0.43	0.33	1.33	**0.74**	0.34	0.67	**0.46**	0.34
7	110	81.45	73.18	**10.50**	0.25	0.16	4.44	**1.15**	0.17	0.50	**0.25**	0.13
8	65	74.07	65.43	**10.29**	0.21	0.15	4.21	**1.00**	0.14	0.25	**0.17**	0.11
9	45	66.68	56.61	**10.25**	0.19	0.06	0.80	**0.30**	0.07	0.25	**0.13**	0.05

(a) The spherical DTLZ front.

		basic HSO					HSO+MDP			HSO+MWW		
n	m	$wrst$	$awst$	$avrg$	$abst$	$best$	$wrst$	$avrg$	$best$	$wrst$	$avrg$	$best$
5	800	65.16	52.31	**11.40**	1.06	0.91	1.37	**1.18**	1.03	1.37	**1.21**	1.04
6	220	65.74	55.58	**9.93**	0.41	0.26	1.67	**0.77**	0.40	0.80	**0.46**	0.28
7	110	83.76	73.60	**10.95**	0.23	0.15	2.19	**0.57**	0.16	0.35	**0.25**	0.15
8	65	74.81	69.93	**10.96**	0.25	0.09	1.32	**0.52**	0.11	0.50	**0.20**	0.08
9	45	66.59	59.00	**10.14**	0.22	0.10	1.26	**0.48**	0.08	0.39	**0.19**	0.06

(b) The linear DTLZ front.

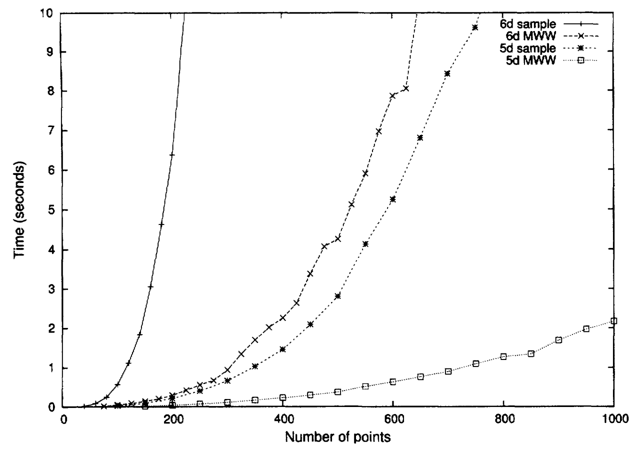
		basic HSO					HSO+MDP			HSO+MWW		
n	m	$wrst$	$awst$	$avrg$	$abst$	$best$	$wrst$	$avrg$	$best$	$wrst$	$avrg$	$best$
5	1000	20.68	16.44	**9.90**	3.40	3.14	4.17	**3.78**	3.52	3.96	**3.66**	3.37
6	250	28.64	20.75	**10.71**	3.38	2.79	4.49	**3.88**	3.43	5.49	**4.12**	3.42
7	110	29.36	25.57	**13.00**	4.28	3.10	7.61	**5.77**	3.35	7.54	**5.16**	3.82
8	60	24.20	20.11	**10.29**	3.54	2.36	6.51	**4.67**	3.01	7.17	**4.40**	3.02
9	40	24.69	19.38	**9.84**	3.82	2.49	6.80	**5.00**	2.37	5.93	**4.38**	2.24

(c) The discontinuous DTLZ front.

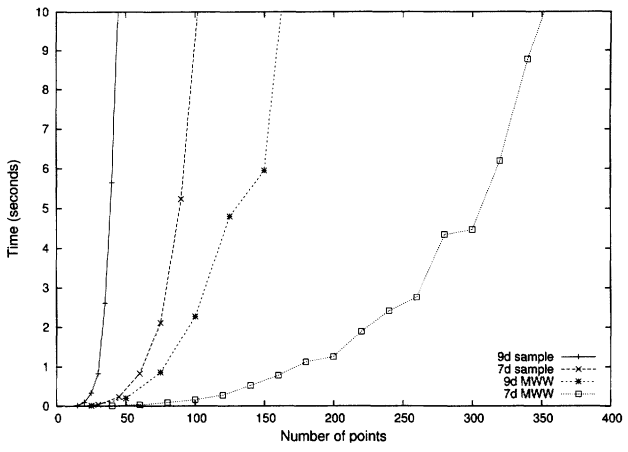
		basic HSO					HSO+MDP			HSO+MWW		
n	m	$wrst$	$awst$	$avrg$	$abst$	$best$	$wrst$	$avrg$	$best$	$wrst$	$avrg$	$best$
5	1200	23.20	17.46	**11.39**	7.97	7.02	16.37	**9.87**	7.66	10.38	**8.32**	7.15
6	300	22.97	18.92	**11.16**	6.24	3.73	12.25	**9.11**	3.65	12.55	**7.03**	3.71
7	130	30.89	24.96	**13.74**	7.50	6.18	15.77	**10.47**	6.39	13.82	**8.17**	6.17
8	70	37.44	26.99	**12.39**	4.47	2.63	6.65	**5.31**	3.80	8.07	**5.26**	2.58
9	45	32.63	22.31	**9.73**	3.59	1.39	9.39	**4.76**	1.42	8.16	**4.12**	1.62

(d) Randomly-generated fronts.

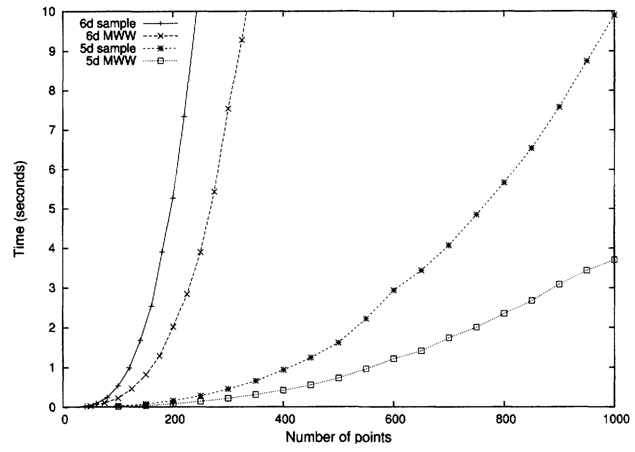
Table 1: Comparison of the performance of HSO, HSO+MDP, and HSO+MWW on various fronts. Each datum is based on ten different data sets: the figures for basic HSO are calculated using the sampling procedure described in Section 6. For each value of n, m is chosen so that the HSO $avrg \approx 10$s.



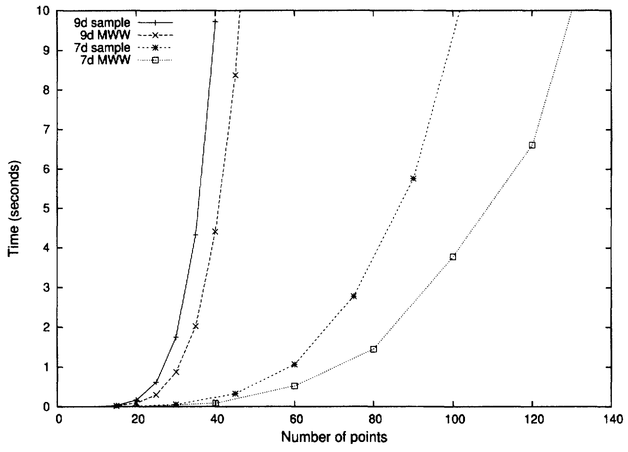
(a) The spherical DTLZ front in 5 and 6 objectives.



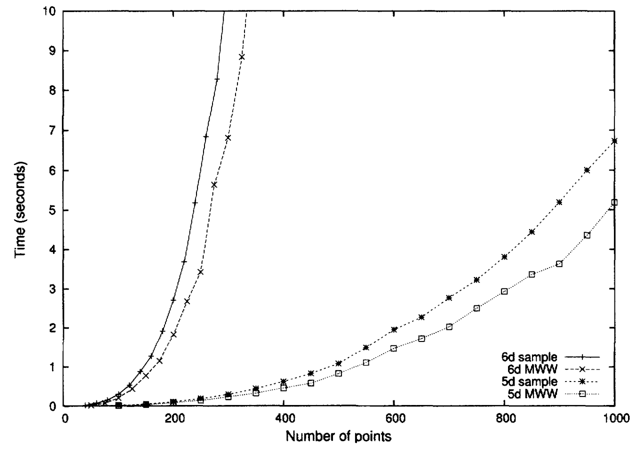
(b) The spherical DTLZ front in 7 and 9 objectives.



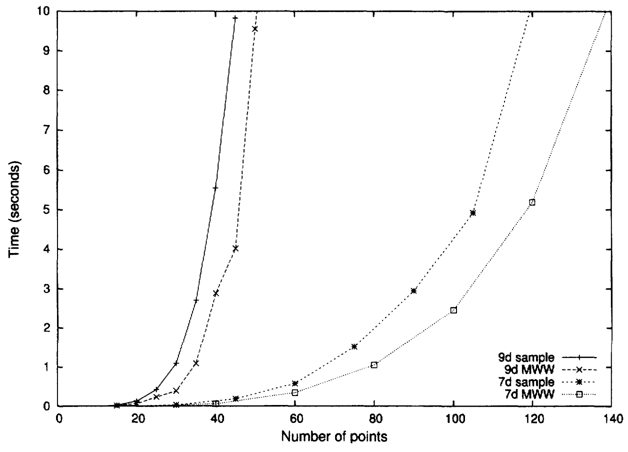
(c) The discontinuous DTLZ front in 5 and 6 objectives.



(d) The discontinuous DTLZ front in 7 and 9 objectives.



(e) Randomly-generated fronts in 5 and 6 objectives.



(f) Randomly-generated fronts in 7 and 9 objectives.

Figure 5: Comparison of the performance of HSO and HSO+MWW on various fronts. Each datum is based on ten different data sets: the figures for basic HSO are calculated using the sampling procedure described in Section 6. The plot for the linear DTLZ front is similar to that for the spherical front and is excluded for space reasons.

avrg is the average time for all of the runs.

abst is the average of the shortest time for each front.

best is the shortest time for any run on any front.

- For each heuristic:

wrst is the longest time for any run on any front.

avrg is the average time for all of the runs.

best is the shortest time for any run on any front.

As observed previously by While *et al.*[13], the best-case objective order for the degenerate front gives performance that is polynomial in the number of objectives, so for that front, we plot only the performance of HSO+MWW. Each other plot compares the performance of HSO+MWW with the average performance of HSO over the sample of permutations of the objectives.

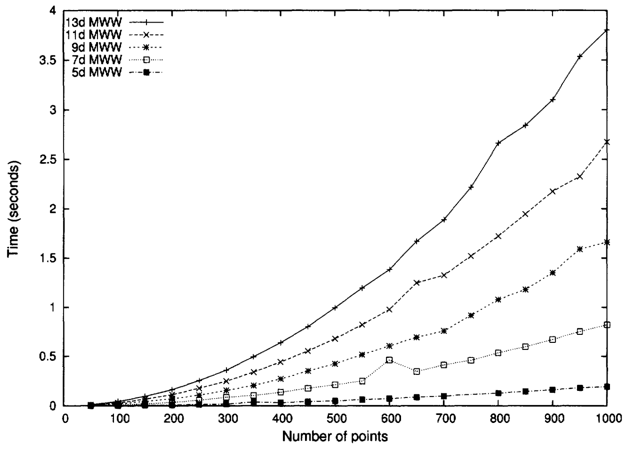


Figure 6: The performance of HSO+MWW on the degenerate front. Each datum is based on ten different data sets.

6.2 Randomly-generated data

We generated sets of m mutually non-dominating 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 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 kept adding points until $|S| = m$. Each hypervolume was calculated as a maximisation problem in every objective, relative to the origin.

Table 1(d) and Figures 5(e)–5(f) show the resulting comparison.

6.3 Discussion

For each heuristic in each row of each table, we make the following comparisons.

- We compare *avrg* for the heuristic with the range *awst*...*avrg*...*abst* for basic HSO, to determine how much improvement the heuristic delivers in typical cases.
- We compare *wrst* for the heuristic with *wrst* and *awst* for basic HSO, to determine how well the heuristic avoids the worst-case ordering.
- We compare *best* for the heuristic with *abst* and *best* for basic HSO, to determine how close the heuristic gets to the best-case ordering. (Note that the best cases for the heuristics sometimes beat the best case for basic HSO: this is due to the incomplete nature of the sampling used for the basic HSO figures.)

We make the following observations.

- For all of the DTLZ fronts, both heuristics deliver major performance gains, and MWW in particular delivers performance that is not far from optimal. The performance gains for the spherical and linear fronts in particular are spectacular: speed-up factors of 10–80 in the average cases. The performance gain for the discontinuous front is somewhat less (speed-up factors of 2–3): no doubt this is due to some property of the front itself.
- Random fronts may be the worst-case form of data for the heuristics, but both heuristics still always outperform basic HSO in the average case, with speed-up factors up to 2.5.
- Both heuristics avoid the worst-case objective ordering in all cases: in fact, the worst-case for the heuristics is nearly always better than the average case for basic HSO, usually by a substantial amount.
- The performance gain increases both with increasing number of objectives, and with increasing number of points.
- MWW generally out-performs MDP.
- The graphs however highlight the fact that exponential performance makes life tough: although the heuristics deliver useful speed-ups for processing fronts of a given size, they do not always greatly improve the sizes of fronts that can be processed in a given time.

The question arises what size of fronts the enhanced algorithm can process in various times. Table 2 shows this data for HSO+MWW on the spherical front. We chose ten seconds as indicative of the performance required to use hypervolume in off-line metric calculations after the EA is complete, and one second as indicative of the performance required to use hypervolume in an on-line diversity or archiving mechanism during the execution of the EA.

n	10 seconds	1 second
5	1,900	700
6	650	320
7	350	170
8	240	110
9	160	80
10	110	60
11	80	50
12	70	40
13	50	30

Table 2: Sizes of fronts in various numbers of objectives that HSO+MWW can process in the times indicated, for spherical DTLZ data.

We also performed some minor experimentation to estimate the cost of calculating the heuristics themselves. Our experiments indicate that these calculations usually take less than 1% of the run-time of the enhanced algorithm, and that they never exceed about 6% of the run-time, even with populations up to 2,000. This is of course to be expected, because of the exponential complexity of HSO itself.

7 Conclusions and Future Work

We have described two alternative heuristics that each improve the performance of the HSO algorithm for calculating hypervolume, itself the fastest algorithm described to date. Each heuristic works by re-ordering the objectives in a front to reduce the sizes of the sets of points that have to be processed during the execution of the algorithm. Both heuristics deliver significant improvement to the performance of HSO, with reductions in the run-time of the algorithm of 25–98%. The enhanced HSO will enable the use of hypervolume with larger populations in more objectives.

We intend to speed-up the calculation of our heuristics, e.g. by minimising the cost of dominance-checking, although we do not expect this to deliver serious further improvements. We also intend to pursue other avenues for making HSO faster, such as reducing the amount of repeated work that results from processing slices independently.

We also intend to design an incremental version of HSO, for use as a diversity or archiving mechanism in an evolutionary algorithm.

Acknowledgments

We thank Simon Huband for discussions on hypervolume and HSO, and for providing the raw DTLZ data.

This work was supported partly by The University of Western Australia Research Grants Scheme, and also partly by an ARC Linkage grant.

Bibliography

[1] S. Huband, P. Hingston, L. While, and L. Barone, "An evolution strategy with probabilistic mutation for multi-objective optimization," in *CEC 2003*, H. Abbass and B. Verma, Eds., vol. 4. IEEE, 2003, pp. 2284–2291.

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

[3] R. C. Purshouse and P. J. Fleming, "The MultiObjective Genetic Algorithm applied to benchmark problems — an analysis," The University of Sheffield, UK, Research Report 796, 2001.

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

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

[6] T. Okabe, Y. Jin, and B. Sendhoff, "A critical survey of performance indices for multi-objective optimisation," in *CEC 2003*, H. Abbass and B. Verma, Eds., vol. 2. IEEE, 2003, pp. 878–885.

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

[8] R. Purshouse, "On the evolutionary optimisation of many objectives," Ph.D. dissertation, The University of Sheffield, Sheffield, UK, 2003.

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

[10] M. Laumanns, E. Zitzler, and L. Thiele, "A unified model for multi-objective evolutionary algorithms with elitism," in *CEC 2000*, vol. 1. IEEE, 2000, pp. 46–53.

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

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

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

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

[15] J. Knowles, "Local-search and hybrid evolutionary algorithms for pareto optimisation," Ph.D. dissertation, The University of Reading, 2002.

[16] L. While, "A new analysis of the Lebmeasure algorithm for calculating hypervolume," in *EMO 2005*, ser. LNCS, C. Coello Coello *et al.*, Ed., vol. 3410. Springer-Verlag, 2005, pp. 326–340.

[17] T. Bäck, D. Fogel, and Z. Michalewicz, Eds., *Handbook of Evolutionary Computation.* Iop Institute of Physics, 1997.

[18] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multi-objective optimization test problems," in *CEC 2002*, D. B. Fogel *et al.*, Ed., vol. 1. IEEE, 2002, pp. 825–830.