Improving the IWFG Algorithm for Calculating Incremental Hypervolume

Wesley Cox & Lyndon While
School of Computer Science & Software Engineering
The University of Western Australia
Perth, Australia 6009
Email: {wesley.cox, lyndon.while}@uwa.edu.au

***Abstract*—We describe an optimised version of the incremental hypervolume algorithm IWFG that achieves new levels of performance for this class of algorithm. The principal changes are the use of an adaptive slicing scheme that works well both for points that need to be fully-evaluated, and for those that need only a small amount of evaluation; and the incorporation of an existing heuristic for ordering objectives independently for each point. The new algorithm can process in substantially less than a second sets containing a thousand points in 10–13 objectives, with much typical data; it is therefore a significant contribution to optimisers that use incremental hypervolume for selection, archiving, or diversity.**

I. Introduction

Hypervolume[1] (the S-metric[2] or the Lebesgue measure[3]) is a popular metric for comparing the performance of multi-objective optimisers. The hypervolume of a set of solutions is the size of the portion of objective space that is dominated by those solutions collectively. Hypervolume captures in one scalar both the closeness of the solutions to the optimal set and their spread across objective space. Hypervolume also has nicer mathematical properties than other metrics[4], [5]. Hypervolume's main drawback is that it is expensive to calculate in larger numbers of dimensions; secondary drawbacks include its sensitivity to the relative scaling of the objectives, to the presence or absence of extremal points, and to the placement of the reference point against which other points are compared.

Hypervolume is also increasingly used in-line in multi-objective evolutionary algorithms, either to promote diversity[6], or as part of an archiving mechanism[7], [8], or as part of the selection process[9], [10]. Here the requirement is usually to determine which point in a set contributes least to the hypervolume of the set, so that that point can be discarded. Calculating hypervolume in-line obviously puts much more stress on the performance issue.

The two fastest published algorithms for determining the least-contributing point in a set are IHSO and IWFG. IHSO[11] is based on the widely-studied HSO algorithm[12], [13], [14]: it derives its speed from a variety of heuristics to minimise the number of slices that have to be evaluated for each point, and from a best-first queuing scheme which means that only the least-contributing point has to be completely evaluated. IWFG[15] is based on the newer (and faster) WFG algorithm[16]: it uses the same queuing scheme as IHSO, but it evaluates individual slices of hypervolume using WFG instead of HSO. The relative performance of these two algorithms has been shown to be uneven on different types of fronts[15].

The principal contribution of this paper is a new version of IWFG that is uniformly (and significantly) faster than IHSO. We analyse the behaviour of IWFG to design a new optimised slicing scheme which adapts well for both the smallest point in the set, which must be completely evaluated, and the other points, for which we want to minimise evaluation. We also incorporate a heuristic from IHSO to order the objectives for each point before evaluation. Experiments show that the new algorithm is much faster than IHSO on fronts with more than about six objectives, and it can process in substantially less than a second sets containing a thousand points in 10–13 objectives, with much typical data. It is a significant contribution to optimisers that make use of incremental hypervolume.

The paper is structured as follows. Section II gives background material in multi-objective optimisation and hypervolume, and Section III describes the relevant details of WFG, IHSO, and IWFG. Section IV gives an analysis of IWFG leading to various design improvements, and Section V describes a comprehensive experimental evaluation of the new algorithm. Section VI concludes the paper and suggests future work.

II. Background Material

A. Multi-objective Optimisation

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 wrt 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* wrt 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 wrt the set of all possible vectors. Pareto optimal vectors have the property that improving 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 approximate the Pareto optimal set.

Precise definitions of these terms can be found in [17].

978-1-5090-0623-6/16/$31.00

B. Hypervolume

Given a set of solutions S returned by a multi-objective optimiser, the *hypervolume*[1], [2], [3] of S is the size of the part of objective space that is dominated collectively by the solutions in S, relative to a reference point in the space. If a set S has a greater hypervolume than a set S', S is taken to be a better set of solutions than S'.

The *exclusive hypervolume* (or *incremental hypervolume*) of a point p relative to an *underlying set* S is the size of the part of objective space that is dominated by p but is not dominated by any member of S. Exclusive hypervolume can be trivially defined in terms of hypervolume:

$$ExcHyp(p, S) \quad = \quad Hyp(S \cup \{p\}) - Hyp(S) \qquad (1)$$

We also use the term *inclusive hypervolume* of a point p for the size of the part of objective space dominated by p alone:

$$IncHyp(p) \quad = \quad Hyp(\{p\}) \qquad (2)$$

All of these concepts are illustrated in Fig. 1.

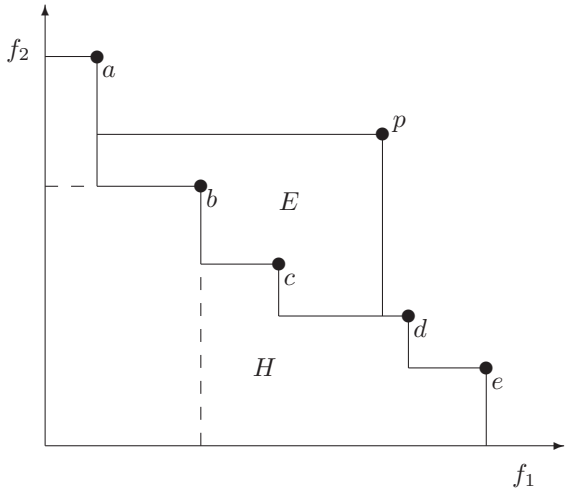


Fig. 1: Maximising in both objectives relative to the origin, the hypervolume of $\{a, b, c, d, e\}$ is the solid-bordered shape labeled H; the exclusive hypervolume of p relative to $\{a, b, c, d, e\}$ is the shape labeled E; and the inclusive hypervolume of any point is the rectangle bounded by that point and the origin (e.g. for b, the dash-bordered rectangle). (Adapted from [16].)

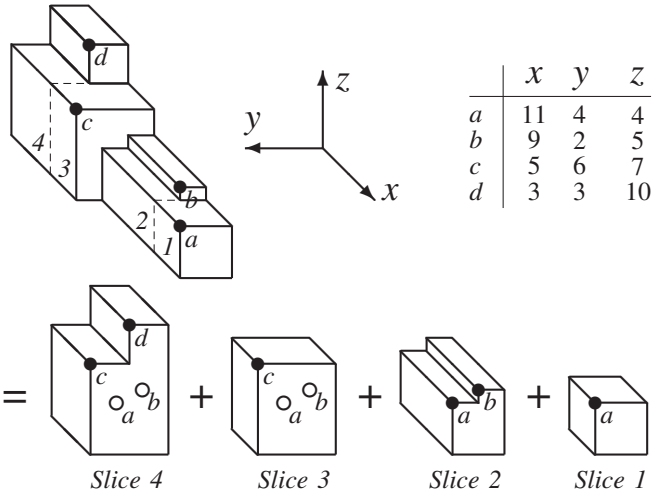
C. Previous Algorithms for Calculating Exact Hypervolumes

Several algorithms have been proposed for calculating hypervolumes and exclusive hypervolumes exactly.

The inclusion-exclusion algorithm for calculating the size of a set union has been adapted for hypervolume calculation[18], but its complexity is $O(n2^m)$ for m points in n objectives, so it is unusable in practice.

LebMeasure[19] processes the points one at a time, calculating the exclusive hypervolume of one point relative to the rest of the set, then discarding that point and processing the others. LebMeasure also has been shown to be very slow[14].

HSO (Hypervolume by Slicing Objectives)[12], [13], [14] slices the nD-hypervolume into separate $n-1$D-hypervolumes through the values in one of the objectives, then it sums the hypervolumes of those slices. Fig. 2 illustrates the principle. HSO's worst-case complexity is $O(m^{n-1})$[14], but good



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

Fig. 2: One step in HSO for calculating $Hyp(\{a, b, c, d\})$. Objective x is processed, leaving four slices in y and z, each with a constant thickness in x. Unfilled circles indicate points which are dominated in y and z. (Reproduced from [11].)

heuristics have been described for re-ordering objectives[20] that deliver much better performance for typical data. Fonseca *et al.* describe a highly-optimised version of HSO[21], including an optimal algorithm for the 3D case[22].

IHSO (Incremental HSO)[11] customises HSO for incremental calculations. Its principal feature is a "best-first" queuing mechanism to determine efficiently the least-contributing point in a set. IHSO is discussed in more detail in Section III. IIHSO (Iterated IHSO)[23] combines IHSO with the point-wise processing of LebMeasure, in metric calculations.

HOY (Hypervolume by Overmars and Yap)[24] adapts an algorithm for solving Klee's measure problem[25] to hypervolume calculation. Objective space is divided into regions, each containing the points that overlap that region. HOY's worst-case complexity ($O(m \log m + m^{n/2})$) is the best of any algorithm, but it performs poorly on realistically-sized fronts[26], [23], [16].

WFG (Walking Fish Group)[16] combines the point-wise processing of LebMeasure with a new bounding trick for calculating exclusive hypervolume quickly, resulting in one of the fastest algorithms yet reported. IWFG (Incremental WFG)[15] customises WFG for incremental calculations. Both of these algorithms are discussed in more detail in Section III.

QHV (Quick Hypervolume)[27] is the fastest exact hypervolume algorithm yet reported. It uses a divide-and-conquer approach reminiscent of Quicksort, dividing up the space using a pivot point and analysing each of the individual sub-spaces recursively. It derives its performance from various clever

implementation tricks, particularly in the data structures used.

An alternative to calculating hypervolume exactly is to use an approximation algorithm (for example [28], [29], [26], [30]). Using such algorithms introduces a trade-off between precision and performance, and much improvement has been made on understanding this trade-off[31], [32], [33]. However, we do not compare with approximation algorithms here.

III. WFG, IHSO, and IWFG

WFG and IHSO form the basis of the algorithm IWFG, so we will discuss the salient features of all three in more detail.

A. WFG

WFG[16] combines the point-wise processing used in LebMeasure[19] with a new technique for calculating the exclusive hypervolume of a point[34], [35]. The exclusive hypervolume of a point p relative to a set S is the difference between the inclusive hypervolume of p and the hypervolume of S after each point has been limited to be no better than p in any objective. Fig. 3 illustrates the principle. Experiments

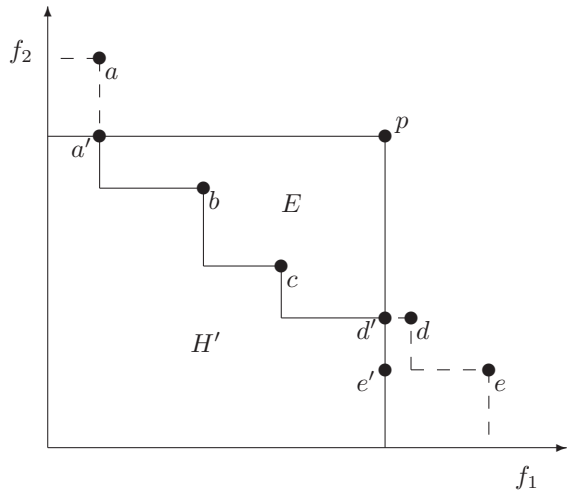


Fig. 3: Maximising in both objectives, the exclusive hypervolume of p relative to $\{a, b, c, d, e\}$ (i.e. E) = the inclusive hypervolume of p (i.e. the rectangle with p at the top-right corner) minus the hypervolume of $\{a', b, c, d', e'\}$ (i.e. H'). e' is dominated and can be discarded. (Reproduced from [16].)

show that this limiting operation produces a lot of points that are dominated within the new set, so subsequent calculations are very fast, and WFG is one of the fastest known exact hypervolume algorithms[16].

B. IHSO

IHSO[11] uses the slice-by-slice technique used in HSO, combined with heuristics to efficiently order the objectives and various techniques to eliminate slices where possible. In particular, slices "above" a point p and slices where p is dominated in the remaining objectives both contribute nothing to p's exclusive hypervolume.

Central to the performance of IHSO is a priority queuing scheme that tries to minimise the amount of work that must be done on each point. The volume dominated by each point is divided into small chunks, and only the currently-smallest point is processed at any given time. IHSO terminates when the currently-smallest point has been completely processed. Fig. 4 outlines the scheme. The granularity of the evaluation was set empirically to minimise the work done.

```
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. 4: Outline of the best-first queuing scheme in IHSO. (Reproduced from [11].)

C. IWFG

IWFG[15] combines the fast bounding trick of WFG with the priority queuing scheme of IHSO, but this creates a problem: because the trick involves subtracting one volume from another, we cannot simply defer either calculation. The queuing scheme assumes that the partial volume of each point increases monotonically as it is processed, otherwise the designation of a "smallest point" has no reliable meaning.

In order to defer calculations, we need an algorithm that divides up the space into separate volumes to be summed. So IWFG does a single HSO-style slicing step at the top level, it calculates the highest non-zero slice for each point immediately to initialise the queue, and it defers the others. All slices needed are processed using WFG, to maximise performance. Fig. 5 gives pseudo-code for IWFG.

```
iwfg(pl):
  sort the points worsening in any objective
  for each point p in pl:
    s = the highest slice relevant to p
    partial[p] = exclhv(p, s)
  sm = the point with the smallest partial
  while sm has not been completely processed
    s = the next slice for sm
    partial [sm] += exclhv(sm, s)
    sm = the point with the smallest partial
  return sm
```

Fig. 5: Pseudo-code for IWFG. (Reproduced from [15].)

IV. Analysing and Improving IWFG

The performance graphs in [15] show that the performance of the initial version of IWFG varies in non-obvious ways for different types of data. We describe here an analysis of the algorithm's behaviour, to design improvements.

A. Behaviour of IWFG

The queuing scheme used in IWFG means that (usually) only the least-contributing point p_s is completely evaluated. Its performance depends on two aspects.

1) How much time is spent evaluating p_s. If we could identify p_s in advance, we would evaluate its entire

volume using WFG, because WFG is faster[1]: of course this is impossible, so the volume is sliced using IHSO, then the slices are evaluated using WFG. The more p_s is sliced, the slower will be its evaluation.

2) How much time is spent evaluating other points. For each point other than p_s, we want to evaluate only enough volume to establish that this point is not the smallest. Ideally we would like to evaluate in one go using WFG the smallest chunk whose hypervolume is bigger than that of p_s.

Unfortunately, we can see that these will often conflict. In future work, [15] introduced the idea of slicing a point's hypervolume multiple times: let us formalise and analyse that concept. We will use D_k (Depth k) to denote a version of IWFG that slices each point's hypervolume k times, then evaluates each volume using WFG. For example, given a front in 7D, D_3 would slice three times and evaluate (and sum) a bunch of 4D volumes for each point, as illustrated in Fig. 6. At the extremes, D_0 means evaluating each point

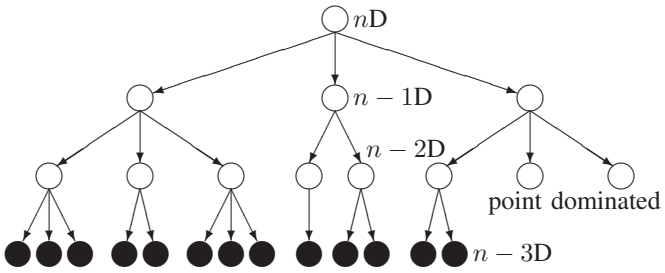


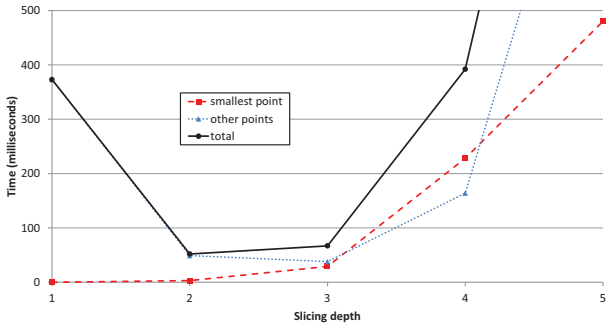
Fig. 6: The D_3 scheme of IWFG, showing the slicing applied to one point. Open circles represent volumes that are sliced using IHSO; filled circles represent volumes that are evaluated in one go using WFG. For real data, the breadth would be much higher. Some branches may terminate early due to sliced points being dominated (as shown), but all branches "try" to slice three times.

completely using WFG only, and D_∞ means using HSO only, i.e. essentially IHSO. Also note that the original version of IWFG from [15] is equivalent to D_1. Now we can observe the following.

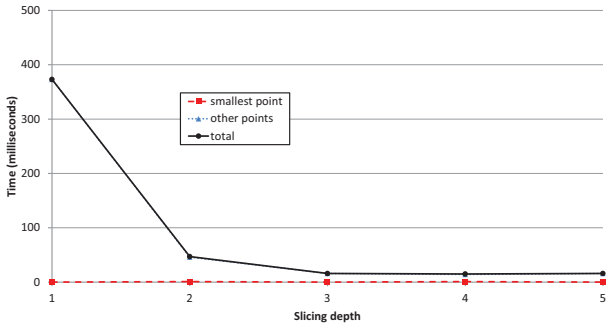
1) The time spent evaluating p_s will be minimised by minimising k, i.e. by using WFG as much as possible.
2) The time spent evaluating the other points will generally be minimised by choosing a higher value of k: high enough that one volume is enough to exceed the volume of p_s (although no higher than that).

Data from typical runs of IWFG substantiates these assertions. Fig. 7a shows that with D_k, the time spent evaluating the smallest point increases significantly with k; while the time spent evaluating the other points decreases with k initially as less evaluation is required, then it too starts to increase as the use of WFG drops away. The cost of queue manipulations is included in the cost for the non-smallest points.

[1]Technically, we wouldn't need to evaluate it at all!



(a) Time breakdown with D_k.



(b) Time breakdown with SD_k.

Fig. 7: Time spent on various components of IWFG vs the degree of slicing used, for randomly-generated data with 200 points in 10D. Each line plots the average processing time for twenty distinct fronts. Where the time for the smallest point is very small, the other two lines coincide.

B. Optimising IWFG

What we need is a slicing scheme that adapts for different points, allowing most of them to slice deeply to minimise their work, but without slowing down the smallest point. We will use SD_k (Starting Depth k) to denote a version of IWFG that does the following:

- slice the point's volume at the top level,
- then slice the first non-zero chunk at the second level,
- then slice the first non-zero chunk at the third level,
- down to the first non-zero chunk at the k^{th} level,
- and evaluate each volume using WFG.

Fig. 8 illustrates the principle for SD_3. The effect is that

1) most of the smallest point is evaluated in big chunks using WFG, speeding up that calculation;

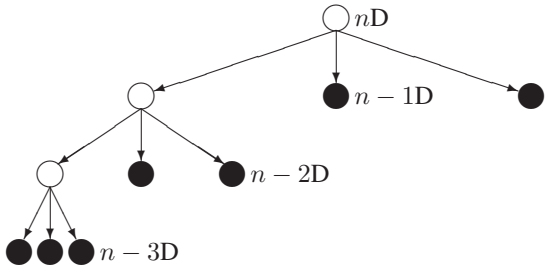


Fig. 8: The SD_3 scheme of IWFG, showing the slicing applied to one point. Open circles represent volumes that are sliced using IHSO; filled circles represent volumes that are evaluated in one go using WFG. Only the left-most branch "tries" to slice three times. The branching factor at the root will depend on the number of points: it will often be in the hundreds, or even in the thousands.

2) several small chunks are available for the non-smallest points to evaluate, which is enough to minimise the work for most of them.

Assuming the worst case where no points are ever dominated, we can quantify this effect for the smallest point:

- for m points with D_k, Fig. 6 shows that $O(m^k)$ chunks are evaluated using WFG;
- whereas with SD_k, Fig. 8 shows that $O(mk)$ chunks are evaluated using WFG.

Even though the individual chunks will be bigger, the increased use of WFG will significantly reduce the evaluation time for the smallest point. This is confirmed in Fig. 7b, which shows that with SD_k, the time spent evaluating the smallest point varies very little with k; while the time spent evaluating the other points decreases initially and stays low, due to the domination of WFG in the calculations.

Section V gives detailed performance data for optimised IWFG with the new slicing scheme.

C. Reordering objectives in IWFG

[11] reports that IHSO is greatly accelerated by using the *Rank* heuristic to select the order used to process the objectives for each point p. *Rank* suggests processing first the objective in which p is best compared to the other points, which gives two advantages.

- There will be fewer empty slices above p before it returns some volume.
- p is relatively poor in remaining objectives, so it is more likely to be dominated early in the recursive process.

Fig. 9 gives an example.

Given that IWFG also uses HSO-style slicing steps, we might reasonably expect that the same heuristic will also benefit IWFG, especially the SD variant, because of its asymmetrical expansion. The performance graphs reported in Section V examine this effect. We note also that WFG has been shown to be relatively insensitive to the order of the objectives[36], so it makes sense to use a heuristic like *Rank* that is tuned to optimising slicing.

43	59	19	40
66	58	24	10
78	46	1	50
100	24	22	60
115	12	20	20
126	1	2	30

Fig. 9: Consider these six points in 4D, maximising every objective. For the first point *Rank* would suggest processing first the second objective (where that point is the best), then the fourth objective (where it is the third-best), then the third objective, then the first. For the last point it would suggest processing the first objective, then the fourth, then the third, then the second. In reality, both points would be dominated early on.

This now allows us to give complete pseudo-code for the SD_k version of IWFG, in Fig. 10.

```
iwfg(pl, k):
  for each point p in pl:
    sort the objectives according to Rank
    sls[p] = the slices for p at the top level
    repeat k-1 times:
      sls[p] = slice(head(sls[p])) ++ tail(sls[p])
    partial[p] = exclhv(p, head(sls[p]))
    sls[p] = tail(sls[p])
  sm = the point with the smallest partial
  while sm has not been completely processed
    partial [sm] += exclhv(sm, head(sls[sm]))
    sls[sm] = tail(sls[sm])
    sm = the point with the smallest partial
  return sm
```

Fig. 10: Pseudo-code for the SD_k version of IWFG.

V. Experimental Performance

We performed a series of experiments to investigate the performance of the new version of IWFG, and to compare its performance with IHSO.

A. Data

We used two types of data.

- Randomly-generated fronts, with random values $x \in [0.1, 10]$ 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.
- The discontinuous and spherical fronts from the DTLZ test suite[37]. We omit the degenerate front because it runs extremely quickly with all reasonable algorithms, and the linear front because its performance is very similar to the spherical front. 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.

The data used in the experiments is available on request.

We ran all fronts both as maximisation problems relative to the origin, and as minimisation problems with various reference points given on the relevant figures. In each run, the algorithm takes a set of points and determines which point has the smallest exclusive hypervolume relative to the rest of the set.

All timings were performed on a machine with a quad core 3.20GHz i7 processor and 8GB of RAM, running Red Hat Enterprise Linux 6.6. All programs were written in C and compiled with gcc v4.4.7 –O3 –march=core2.

B. Results

Figs. 11–13 plot the performance of four algorithms.

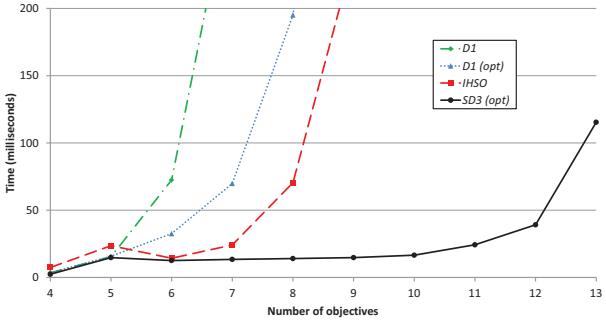
- IHSO, as defined in [11] ($IHSO$).
- IWFG, as defined in [15] (D_1).
- IWFG with the *Rank* heuristic ($D_1(opt)$).
- IWFG with *Rank* and with adaptive slicing ($SD_3(opt)$).

The graphs show clearly that for most front types, optimised IWFG and IHSO perform similarly in 4–6D; but in higher dimensions, IWFG beats IHSO by a large margin in all data.

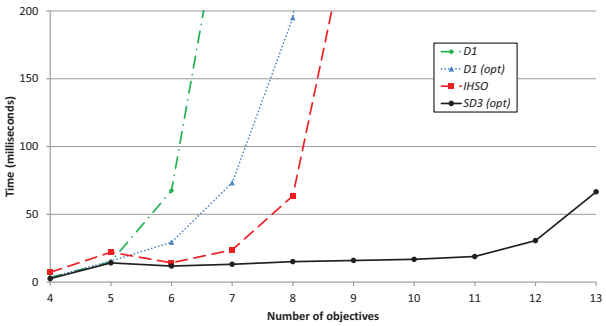
We also observe the following.

- Both the use of the *Rank* heuristic and the use of adaptive slicing contribute significantly to the performance of the new algorithm. The former is apparent from comparing $D_1(opt)$ with D_1: the latter from comparing $SD_3(opt)$ with $D_1(opt)$.
- The relatively poor performance on the discontinuous front is interesting. It might be explained partly by the choice of reference points, which has been shown to be important[11]: otherwise presumably this data has different domination characteristics to the other front-types, or it responds less well to *Rank*. We plan to investigate this.
- We have used SD_3 for this comparison, as it gives consistent performance over this range of data. However for data with more points or with more objectives, it is likely that higher values of k would perform better. Ultimately we will probably need some adaptive scheme for selecting k based on the size of the data.

Table I shows the approximate sizes of fronts that IWFG can process in 100 milliseconds, for each front-type. The table shows that IWFG can process substantial fronts up to 10–13D in all of the types of data tested here.



(a) Maximising wrt $(0, 0, \ldots, 0)$.



(b) Minimising wrt $(10, 10, \ldots, 10)$.

Fig. 11: The performance of IWFG and IHSO on the randomly-generated fronts. Each line plots the average processing time for twenty distinct fronts, each with 1,000 points.

VI. Conclusion

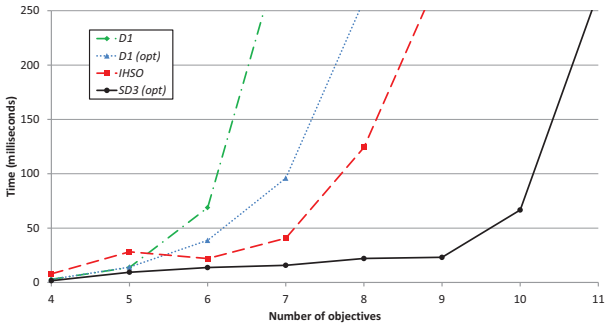
We have described a new version of the algorithm IWFG for determining which point in a set contributes least to the hypervolume of the set. Our analysis of IWFG reveals that its performance is sensitive to the degree of slicing employed at the top-level for each point, and that the smallest point needs to be treated differently to other points. We have developed an adaptive slicing scheme that allows a high degree of use of WFG in evaluating the smallest point, while simultaneously giving other points a good chance of evaluating only a small proportion of their hypervolume. Together with the use of the *Rank* heuristic for ordering objectives for each point, this scheme gives very good performance for even large fronts.

Empirical results show that the new version of IWFG is significantly faster than IHSO in all benchmark data with more than about six objectives, and that it can process in substantially less than a second sets containing a thousand points in 10–13 objectives, with much typical data.

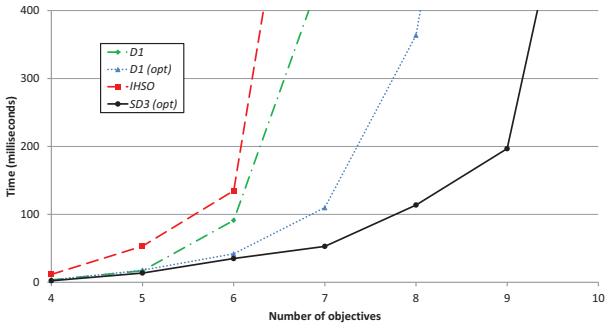
We plan to continue our analysis of IWFG to reveal other opportunities for improving its performance. Possibilities include new heuristics specifically designed for this algorithm, and sharing of evaluation between different points.

References

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

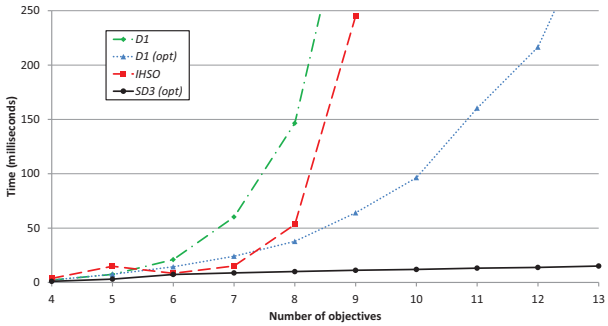


(a) Maximising wrt $(0, 0, \ldots, 0)$.

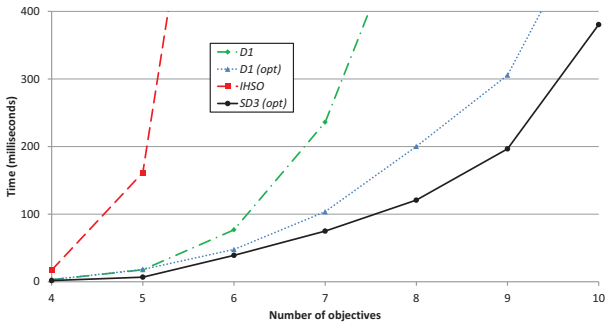


(b) Minimising wrt $(1, 1, \ldots, 1, 2n)$.

Fig. 12: The performance of IWFG and IHSO on the DTLZ discontinuous front. Each line plots the average processing time for twenty distinct fronts, each with 1,000 points.



(a) Maximising wrt $(0, 0, \ldots, 0)$.



(b) Minimising wrt $(1, 1, \ldots, 1)$.

Fig. 13: The performance of IWFG and IHSO on the DTLZ spherical front. Each line plots the average processing time for twenty distinct fronts, each with 1,000 points.

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

[3] M. Laumanns, E. Zitzler, and L. Thiele, "A unified model for multi-objective evolutionary algorithms with elitism," in *Congress on Evolutionary Computation*. IEEE, 2000, pp. 46–53.

[4] 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.

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

[6] S. Huband, P. Hingston, L. While, and L. Barone, "An evolution strategy with probabilistic mutation for multi-objective optimization," in *Congress on Evolutionary Computation*. IEEE, 2003, pp. 2284–2291.

[7] J. Knowles and D. Corne, "Properties of an adaptive archiving algorithm for storing nondominated vectors," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 100–116, April 2003.

[8] J. Knowles, D. Corne, and M. Fleischer, "Bounded archiving using the Lebesgue measure," in *Congress on Evolutionary Computation*. IEEE, 2003, pp. 2490–2497.

[9] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search," in *Parallel Problem Solving from Nature VIII*, ser. Lecture Notes on Computer Science, vol. 3242. Springer-Verlag, 2004, pp. 832–842.

[10] M. Emmerich, N. Beume, and B. Naujoks, "An EMO algorithm using the hypervolume measure as selection criterion," in *Evolutionary Multi-objective Optimisation*, ser. Lecture Notes on Computer Science, vol. 3410. Springer-Verlag, 2005, pp. 62–76.

[11] L. Bradstreet, L. While, and L. Barone, "A fast incremental hypervolume algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp. 714–723, December 2008.

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

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

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

[15] L. While and L. Bradstreet, "Applying the WFG algorithm to calculate incremental hypervolumes," in *Congress on Evolutionary Computation*. IEEE, 2012, pp. 3203–3210.

[16] L. While, L. Bradstreet, and L. Barone, "A fast way of calculating

TABLE I: Sizes of fronts that IWFG can process in 100 milliseconds. n is the number of objectives. gt denotes where the algorithm can process over a thousand points.

	Random		Discontinuous		Spherical	
n	max	min	max	min	max	min
4	gt	gt	gt	gt	gt	gt
5						
6						
7				950		950
8				600		700
9				400		550
10			700	200		350
11			150	150		250
12			100	50		150
13	850					

exact hypervolumes," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 1, pp. 86–95, February 2012.

[17] *Handbook of Evolutionary Computation*. Institute of Physics Publishing and Oxford University Press, 1997.

[18] 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.

[19] M. Fleischer, "The measure of Pareto optima: Applications to multi-objective metaheuristics," in *Evolutionary Multi-objective Optimisation*, ser. Lecture Notes on Computer Science, vol. 2632. Springer-Verlag, 2003, pp. 519–533.

[20] 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*. IEEE, 2005, pp. 2225–2232.

[21] C. M. Fonseca, L. Paquete, and M. López-Ibáñez, "An improved dimension-sweep algorithm for the hypervolume indicator," in *Congress on Evolutionary Computation*. IEEE, 2006, pp. 3973–3979.

[22] N. Beume, C. M. Fonseca, M. López-Ibáñez, L. Paquete, and J. Vahrenhold, "On the complexity of computing the hypervolume indicator," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1075–1082, October 2009.

[23] L. Bradstreet, L. While, and L. Barone, "A fast many-objective hypervolume algorithm using iterated incremental calculations," in *Congress on Evolutionary Computation*. IEEE, 2010, pp. 179–186.

[24] N. Beume, "S-metric calculation by considering dominated hypervolume as Klee's measure problem," *Evolutionary Computation*, vol. 17, no. 4, pp. 477–492, 2009.

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

[26] K. Bringmann and T. Friedrich, "Approximating the least hypervolume contributor: NP-hard in general, but fast in practice," in *Evolutionary Multi-objective Optimisation*, ser. Lecture Notes on Computer Science, vol. 5467. Springer-Verlag, 2009, pp. 6–20.

[27] L. Russo and A. Francisco, "Quick hypervolume," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 481–502, August 2014.

[28] R. Everson, J. Fieldsend, and S. Singh, "Full elite sets for multi-objective optimisation," in *Proceedings of the 5th International Conference on Adaptive Computing in Design and Manufacture*, 2002, pp. 87–100.

[29] J. Bader, K. Deb, and E. Zitzler, "Faster hypervolume-based search using Monte Carlo sampling," in *Multiple Criteria Decision Making for Sustainable Energy and Transportation Systems*, ser. Lecture Notes in Economics and Mathematical Systems, vol. 634. Springer-Verlag, 2010, pp. 313–326.

[30] H. Ishibuchi, N. Tsukamoto, Y. Sakane, and Y. Nojima, "Hypervolume Approximation Using Achievement Scalarizing Functions for Evolutionary Many-Objective Optimization," in *Congress on Evolutionary Computation*. IEEE, 2009, pp. 530–537.

[31] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler, "Theory of the Hypervolume Indicator: Optimal μ-Distributions and the Choice Of The Reference Point," in *Foundations of Genetic Algorithms*. ACM, 2009, pp. 87–102.

[32] K. Bringmann and T. Friedrich, "Don't be Greedy when Calculating Hypervolume Contributions," in *Foundations of Genetic Algorithms*. ACM, 2009, pp. 103–112.

[33] T. Friedrich, C. Horoba, and F. Neumann, "Multiplicative approximations and the hypervolume indicator," in *Genetic and Evolutionary Computation Conference*. ACM, 2009, pp. 571–578.

[34] K. Bringmann and T. Friedrich, "Approximating the least hypervolume contributor: NP-hard in general, but fast in practice," *CoRR*, vol. abs/0812.2636, 2008.

[35] L. Bradstreet, L. While, and L. Barone, "A new way of calculating exact exclusive hypervolumes," The University of Western Australia, School of Computer Science & Software Engineering, Technical Report UWA-CSSE-09-002, 2009.

[36] W. Cox, "Improving the performance of the IWFG algorithm for calculating incremental hypervolumes," Honours dissertation, School of Computer Science & Software Engineering, The University of Western Australia, 2015.

[37] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multi-objective optimization test problems," in *Congress on Evolutionary Computation*. IEEE, 2002, pp. 825–830.