A New Analysis of the LebMeasure Algorithm for Calculating Hypervolume

Lyndon While

The University of Western Australia, WA 6009, Australia
lyndon@csse.uwa.edu.au

Abstract. We present a new analysis of the LebMeasure algorithm for calculating hypervolume. We prove that although it is polynomial in the number of points, LebMeasure is exponential in the number of objectives in the worst case, not polynomial as has been claimed previously. This result has important implications for anyone planning to use hypervolume, either as a metric to compare optimisation algorithms, or as part of a diversity mechanism in an evolutionary algorithm.

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

1 Introduction

Multi-objective optimisation problems abound, and many evolutionary algorithms have been proposed to derive good solutions for such problems, for example [4, 11, 1, 5]. However, the question of what metrics to use in comparing the performance of these algorithms remains difficult[7, 9, 4]. One metric that has been favoured by many people is *hypervolume*[4, 8], also known as the S-metric[10] or the Lebesgue measure[3]. 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. It also has nicer mathematical properties than many other metrics[2, 12], although it can be sensitive to scaling and to the presence or absence of extremal values.

Unfortunately, hypervolume is very expensive to calculate for problems with more than 2–3 objectives. A recently published algorithm that seemed to solve this problem is the LebMeasure algorithm[2, 3]. The original analysis of LebMeasure claimed that this algorithm has complexity $O(m^3n^2)$ for m points in n objectives, thus allowing the efficient calculation of hypervolume for large sets of points in large numbers of objectives. However, we prove in this paper that although LebMeasure is polynomial in the number of points, it is exponential in the number of objectives in the worst-case, and thus its performance deterio-

C. A. Coello Coello et al. (Eds.): EMO 2005, LNCS 3410, pp. 326–340, 2005.

rates sharply as the number of objectives increases. We present three strands of evidence.

Empirical Data. We give some pathological patterns of sets of points that clearly exhibit exponential slowdown under LebMeasure as the number of objectives increases.

A Lower-Bound on the Complexity. For one pattern of sets of two points, we describe their slowdown with a recurrence relation in the number of objectives n, and we prove that this recurrence relation is equal to 2^{n-1}, thus proving that LebMeasure is exponential in the number of objectives in the worst case.

An Upper-Bound on the Complexity. It is hard to establish an exact complexity for LebMeasure in the general case, because it is difficult to be certain which sets of points will suffer the biggest slowdown. However, we can establish a likely upper-bound on the worst-case complexity by considering illegal sets of points that are likely to perform worse than any legal sets of points[1]. We do this for one pattern of sets of points, defining a recurrence relation for their slowdown in m and n, and proving that this recurrence relation is $O(m^n)$.

Taken together, this evidence demonstrates that in the worst case, LebMeasure is exponential in the number of objectives, and thus that it cannot provide a general solution to the performance problem (currently) inherent in calculating hypervolume.

The rest of this paper is structured as follows. Section 2 describes LebMeasure and its behaviour. Section 3 gives an empirical analysis of some patterns of sets of points which exhibit exponential growth (in terms of the number of objectives) under LebMeasure. Section 4 gives a recurrence relation for one pattern of sets of points, and proves that the time to process this pattern grows exponentially with the number of objectives. Section 5 discusses the possibility of placing an upper-bound on the complexity of LebMeasure, and Section 6 concludes the paper and outlines some future work.

2 The LebMeasure Algorithm

2.1 The Definition of Hypervolume

Given a set S containing m points in n objectives, the hypervolume of S is the size of the portion of objective space that is dominated by at least one point in S. We say that a point $\overline{x}$ dominates a point $\overline{y}$ iff $\overline{x}$ is at least as good as $\overline{y}$ in every objective *and* $\overline{x}$ is strictly better than $\overline{y}$ in at least one objective. The hypervolume of S is calculated relative to a reference point which is worse than (or equal to) every point in S in every objective. Given two sets of points S and S' containing solutions to a given problem, if S occupies a greater hypervolume than S', then S is taken to be a better solution to the problem.

[1] A set of points is legal only if its members are mutually non-dominating.

2.2 The Behaviour of LebMeasure

LebMeasure works by processing the points in a set one at a time. It calculates the volume that is dominated exclusively by the first point, then it discards that point and moves on to the subsequent points, until all points have been processed and all volumes have been summed. This is particularly efficient when the volume dominated exclusively by a point $\overline{x}$ is "hyper-cuboid", but where this is not the case, LebMeasure lops-off the largest hyper-cuboid volume that is dominated exclusively by $\overline{x}$, and replaces $\overline{x}$ with a set of up to n "spawned" points that dominate the remainder of $\overline{x}$'s exclusive hypervolume. A spawn is discarded if it dominates no exclusive hypervolume, either because it has one or more objective values equal to the reference point, or because it is dominated by an unprocessed point.

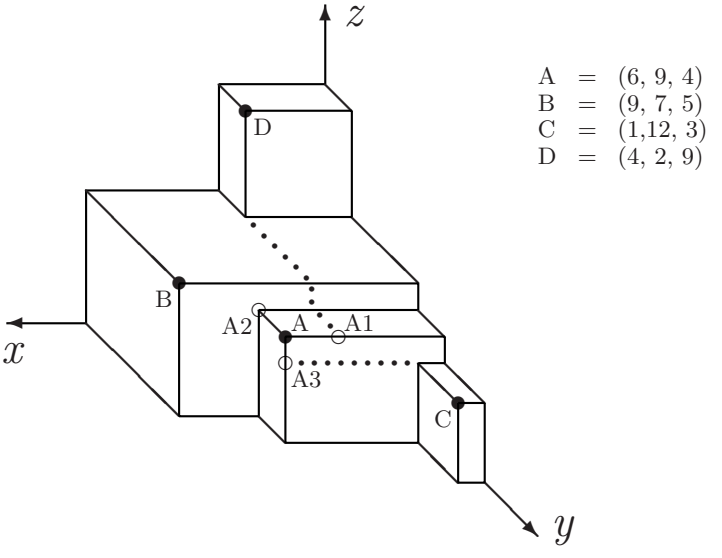

Fig. 1. Spawning in LebMeasure. The blocks denote the hypervolume dominated by {A, B, C, D} in a maximisation problem relative to the origin. The filled circles represent the four points, and the empty circles represent the three potential spawns of A. Each spawn is generated by reducing one objective to the largest smaller value from the other points. It is clear that A2 is dominated by B, so A is replaced by A1 and A3

Consider as an example the set of points shown in Fig. 1, with Point `A` to be processed first. The largest hyper-cuboid dominated exclusively by `A` is bounded at the opposite corner by `(4,7,3)`. Thus the three potential spawns of `A` are

```
A1 = (4, 9, 4)      A2 = (6, 7, 4)      A3 = (6, 9, 3)
```

However, `A2` is dominated by `B` (from the main list of points), so only `A1` and `A3` dominate exclusive hypervolume of their own, and only those two are added to the main list to be processed.

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

Fig. 2 gives pseudo-code for LebMeasure.

```
hypervolume (ps):
   pl = a stack containing the points from ps in some order,
        each paired with n, the number of objectives
   hypervolume = 0
   while pl /= emptylist
     (p, z) = head (pl)
     pl = tail (pl)
     a = oppositeCorner (p, pl)
     hypervolume = hypervolume + volBetween (p, a)
     ql = spawns (p, z, a, pl)
     prepend ql to pl
   return hypervolume

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

oppositeCorner (p, pl) returns the point that bounds the
largest hyper-cuboid that is dominated exclusively by the
point p relative to the points on pl

volBetween (p, q) returns the hypervolume between the points p and q

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

Fig. 2. Pseudo-code for LebMeasure. Each point on the main stack is paired with the highest index that it can use to generate a non-dominated spawn

2.3 The Complexity of LebMeasure

[2, 3] claim that the worst-case complexity of LebMeasure is $O(m^3n^2)$. This claim is based largely on an observation about the spawning behaviour of the algorithm. Consider the situation in the above example, after `A` has been processed:

```
A1 = (4,  9, 4)
A3 = (6,  9, 3)
```

```
B  = (9,  7, 5)
C  = (1, 12, 3)
D  = (4,  2, 9)
```

The three potential spawns of `A1` are

```
A11 = (1, 9, 4)     A12 = (4, 7, 4)     A13 = (4, 9, 3)
```

Both `A12` and `A13` are dominated (respectively by `B` and `A3`). Moreover, *this is guaranteed to be the case*. For example, `A13` is generated by reducing the first and third objectives from `A`, *but* `A3` is still unprocessed, and that point is generated by reducing only the third objective from `A`. Similarly, `A12` is guaranteed to be dominated by `A2` (or in this example, by the point that dominated `A2`: dominance is transitive).

Thus these two points are guaranteed to be dominated, and LebMeasure need never generate them in the first place. This is a major optimisation for the algorithm. However, we shall see that this optimisation is not enough to change the time complexity of LebMeasure: it provides only a constant-factor speed-up.

The optimisation does however change the space complexity of LebMeasure. As described in [2, 3], the size of the stack of points is now bounded by $m+n-1$, because the number of potential spawns generated from a point `P` is bounded by the lowest-numbered objective that was reduced during the formation of `P`.

3 Empirical Evidence of the Complexity of LebMeasure

It is clear that the complexity of LebMeasure is at least equal to the number of points processed *that actually contribute to the result* (including spawns, spawns of spawns, etc.): this is the number of times that the `while` loop in Figure 2 is traversed. We can thus estimate a lower bound on the complexity by counting the numbers of contributing points for some concrete sets of points. We consider two pathological sets of points that together suggest strongly that LebMeasure is exponential in the number of objectives.

Consider first the sets of points described by the pattern shown in Fig. 3. The numbers of points processed for this pattern are as given in Table 1. The

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

Fig. 3. A pathological example for LebMeasure. This pattern describes sets of five points in n objectives, $n \geq 2$. All columns except the first are identical. The pattern can be generalised for other numbers of points

Table 1. The numbers of points processed for the pattern from Fig. 3, equal to m^{n-1}

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

numbers clearly show growth that is exponential in the number of objectives, indicating that this pattern generates m^{n-1} points.

However, this pattern raises the question of the order in which the points are processed. If the points from Fig. 3 are reversed before being presented to LebMeasure, then no non-dominated spawns are generated and the number of points is m, for all values of n. The order shown in Fig. 3 is in fact the worst-case ordering for this pattern.

Thus one way to improve the overall performance of LebMeasure is to develop heuristics to select the best order in which to present the points to the algorithm. However, we illustrate now that there are patterns of points that exhibit exponential complexity under LebMeasure even in their best-case ordering. Consider the sets of points described by the pattern shown in Fig. 4.

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

Finally, [6] proposes the use of LebMeasure in an evolutionary algorithm that calculates hypervolume incrementally to promote diversity. Structurally, LebMeasure is well-suited to this task, because it calculates explicitly the hypervolume dominated exclusively by its first point. However, even evaluating the hypervolume dominated by a single point can be exponential in LebMeasure, as

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

Fig. 4. A second pathological example for LebMeasure. This pattern describes sets of five points in n objectives, $n \geq 2$. The first and last columns are fixed, and if $n > 2$, the second column is fixed and each other column is a simple rotation of the previous column. The pattern can be generalised for other numbers of points

Table 2. The best-case numbers of points processed for the pattern from Fig. 4, approximately equal to (but not bounded by) $m(m!)^{(n-2)div\ m}((n-2)mod\ m)!$. The values of m reported were limited by the numbers of permutations that must be tested

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

shown by the numbers in Table 3. Note that in this context, where we need to evaluate the hypervolume contribution of a single point, changing the order of the other points makes no difference to the algorithm's performance.

Thus a simple empirical study of LebMeasure suggests that it is exponential in the number of objectives.

Table 3. The numbers of points processed for the first point of the pattern from Fig. 3, equal to $m^{n-1}-(m-1)^{n-1}$, i.e. $O(m^{n-2})$

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

4 A Lower Bound on the Complexity of LebMeasure

To prove a lower bound on the worst-case complexity of LebMeasure, we give a recurrence *hcs* for the number of points processed for the two-point pathological example in Fig. 5.

$1\,2\cdots 2$
$2\,1\cdots 1$

Fig. 5. A third pathological example for LebMeasure. This is the pattern from Fig. 3, with two points in n objectives

It is useful to introduce some terminology at this point. Given a point $\overline{x}$: the *descendants* of $\overline{x}$ are the contributing spawns of $\overline{x}$, and their contributing spawns, and so on; and the *relatives* of $\overline{x}$ are the points derived from the same original point as $\overline{x}$.

$$h(1,k) = 1 \tag{1}$$

$$h(n,k) = 1 + \sum_{i=0}^{k-1} h(n-1,i) \tag{2}$$

$$hcs(n) = h(n-1,n-1) + 1 \tag{3}$$

$h(n,k)$ returns the number of points processed for a point (or spawn) which has n 2s, of which we can reduce k and still generate spawns that aren't dominated by their relatives. The value of k in the call for a point $\overline{x}$ depends on the ancestry of $\overline{x}$, in particular on which 2s were reduced during the formation of $\overline{x}$: basically k is the number of contiguous 2s in the lower indices of $\overline{x}$'s objective values.

(1) says that if a point has only one 2, it can't generate any non-dominated spawns: they would be dominated by the second original point (but count one for itself). (2) says that if a point can reduce k 2s, it can generate k spawns by reducing any one of them (and count one for itself). Each spawn will have $n-1$ 2s, and each will be able to reduce the 2s with lower indices than the one just reduced.

For example, the point $(2,2,2,1,2,1)$ corresponds to $h(4,3)$, and it generates three spawns: $(1,2,2,1,2,1)$ corresponds to $h(3,0)$; $(2,1,2,1,2,1)$ corresponds to $h(3,1)$; and $(2,2,1,1,2,1)$ corresponds to $h(3,2)$.

$hcs(n)$ returns the number of points processed for the points from Fig. 5 in n objectives. (3) calls h for the first point, but obviously the second point generates no spawns.

It is easy to see by expansion that

$$hcs(n) = 2^{n-1}$$

We now give an inductive proof of this equality.

Lemma 1:

$$h(k,k) = \sum_{i=1}^{k} h(i,i-1) \tag{4}$$

Proof: The proof is by induction on k. The base case is $k = 1$:

$$h(1,1) = h(1,0) = \sum_{i=1}^{1} h(i,i-1), \text{ by (1)}$$

Inductive case: Suppose the formula holds for $k = j$, i.e.

$$h(j, j) = \sum_{i=1}^{j} h(i, i-1) \tag{5}$$

$$\begin{aligned} h(j+1, j+1) &= 1 + \sum_{i=0}^{j} h(j, i), \text{ by (2)} \\ &= 1 + h(j, j) + \sum_{i=0}^{j-1} h(j, i) \\ &= 1 + \sum_{i=1}^{j} h(i, i-1) + \sum_{i=0}^{j-1} h(j, i), \text{ by (5)} \\ &= h(j+1, j) + \sum_{i=1}^{j} h(i, i-1), \text{ by (2)} \\ &= \sum_{i=1}^{j+1} h(i, i-1) \end{aligned}$$

Hence the formula holds for $k = j + 1$, and by induction, it holds for all k.

Lemma 2:

$$h(n, k) = 2^k, \text{if } n > k \tag{6}$$

Proof: The proof is by induction on n. The base case is $n = 1$:

$$h(1, 0) = 1 = 2^0, \text{ by (1)}$$

Inductive case: Suppose the formula holds for $n = j$, i.e.

$$h(j, k) = 2^k, \text{if } j > k \tag{7}$$

$$\begin{aligned} h(j+1, k) &= 1 + \sum_{i=0}^{k-1} h(j, i), \text{ by (2)} \\ &= 1 + \sum_{i=0}^{k-1} 2^i, \text{ by (7), given } j > k \\ &= 1 + 2^k - 1, \text{ sum of a geometric series} \\ &= 2^k \end{aligned}$$

Hence the formula holds for $n = j + 1$, and by induction, it holds for all n.

Theorem 1:

$$hcs(n) = 2^{n-1}$$

Proof:

$$\begin{aligned} hcs(n) &= h(n-1, n-1) + 1, \text{ by (3)} \\ &= \sum_{i=1}^{n-1} h(i, i-1) + 1, \text{ by (4)} \\ &= \sum_{i=1}^{n-1} 2^{i-1} + 1, \text{ by (6)} \\ &= \frac{1}{2} \sum_{i=1}^{n-1} 2^i + 1 \\ &= \frac{1}{2}(2^n - 2) + 1, \text{ sum of a geometric series} \\ &= 2^{n-1} \end{aligned}$$

Thus the two-point pathological example in Fig. 5 has complexity that is exponential in the number of objectives n, and we can say immediately that LebMeasure is exponential in the number of objectives in the worst case.

5 An Upper-Bound on the Complexity of LebMeasure

To develop an upper-bound on the general worst-case complexity, consider the "spawning tree" for three objectives, illustrated in Fig. 6. The root of the tree is an original point, and where a line connects two points, the lower point is a spawn of the higher point: thus the tree contains all of the descendants of the point at the root. The points are processed in depth-first order, left-to-right across the tree. The key feature of the tree is that if Objective i is reduced at any time during the formation of a point $\overline{x}$, then any potential spawn of $\overline{x}$ where Objective j, $j > i$, is reduced is guaranteed to be dominated by an unprocessed relative of $\overline{x}$. This limits the size of the tree to some extent.

We can derive a recurrence q that counts the points spawned in this tree.

$$q(0, y, z) = 1 \tag{8}$$

$$q(x, y, z) = 1 + q(x-1, y, z) + \sum_{i=1}^{y-1} q(z, i, z) \tag{9}$$

$$p(m, n) = \sum_{i=0}^{m-1} q(i, n, i) \tag{10}$$

$q(x, y, z)$ returns the number of descendants of a point, where

- Objective y is the highest-indexed objective that can be changed,
- there are up to x points smaller than the current one in Objective y, and
- there are up to z points smaller than the current one in Objectives $1 \ldots y-1$.

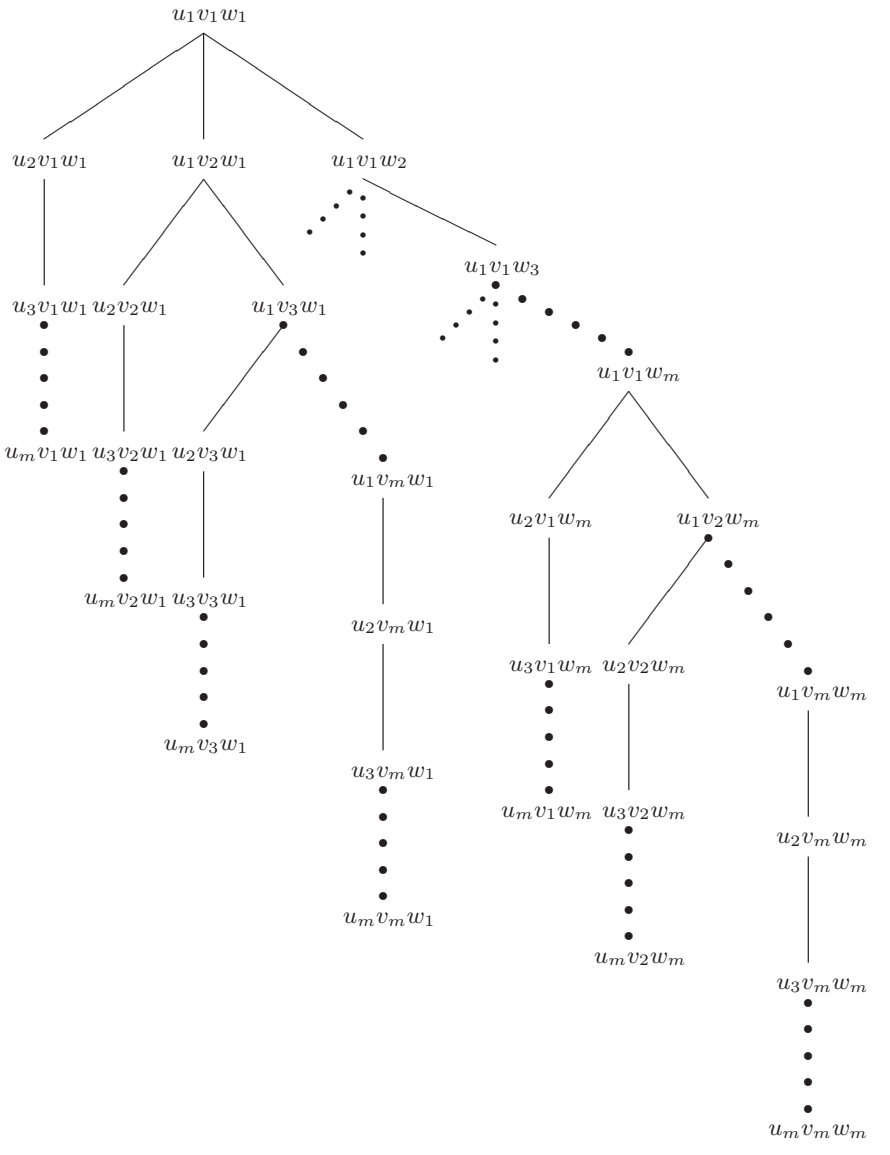

Fig. 6. The "spawning tree" for LebMeasure for m points in three objectives. The original point at the root of the tree generates all of the points in the tree, in the worst case. In each objective x, the values occur in the worsening order $x_1, x_2, \ldots, x_m$. The key feature is that when the i^{th} objective has been reduced, no "higher" objective can subsequently be reduced, because the resulting spawn would certainly be dominated (explicitly: if v has been reduced, w can't be reduced; and if u has been reduced, neither v nor w can be reduced). The tree generalises naturally to other numbers of objectives

(8) says that if there are no points smaller in the current objective (i.e. the current objective equals the reference point), just count one for the point itself. (9) says that the current point can generate y spawns, either by:

- reducing Objective y, so the first argument is decremented, or
- reducing any one of Objectives $1 \ldots y-1$, so the first argument is restored,

and count one for the point itself.

$p(m, n)$ returns the worst-case number of points processed for m points in n objectives. (10) calls q separately for each original point: in each call, i is the number of original points yet to be processed.

Note that this recurrence models patterns of sets of the form shown in Fig. 7, which are illegal, because the points are not mutually-non-dominating. Hence the

m	m	$\cdots$	m
m-1	m-1	$\cdots$	m-1
$\vdots$	$\vdots$		$\vdots$
1	1	$\cdots$	1

Fig. 7. A (theoretical but illegal) pathological example for LebMeasure. This pattern describes sets of m points in n objectives, $n \geq 2$. All columns are identical

solution to p will be only an upper-bound on the general complexity. However, it is difficult to derive an exact general complexity, because it is difficult to know which sets of points will suffer the worst performance[2].

It is easy to see by expansion that

$$p(m, n) = \sum_{i=1}^{m} i^n$$

We now give an inductive proof of this equality.

Lemma 3:

$$q(x, y, z) = 1 + x(1 + \sum_{i=1}^{y-1} q(z, i, z)) \tag{11}$$

Proof: The proof is by induction on x. The base case is $x = 0$:

$$q(0, y, z) = 1 = 1 + 0(1 + \sum_{i=1}^{y-1} q(z, i, z)), \text{ by (8)}$$

[2] It is also unnecessary to some extent: the lower-bound is the more significant result.

Inductive case: Suppose the formula holds for $x = j$, i.e.

$$q(j, y, z) = 1 + j(1 + \sum_{i=1}^{y-1} q(z, i, z)) \tag{12}$$

$$\begin{aligned} q(j+1, y, z) &= 1 + q(j, y, z) + \sum_{i=1}^{y-1} q(z, i, z), \text{ by (9)} \\ &= 1 + 1 + j(1 + \sum_{i=1}^{y-1} q(z, i, z)) + \sum_{i=1}^{y-1} q(z, i, z), \text{ by (12)} \\ &= 1 + (j+1)(1 + \sum_{i=1}^{y-1} q(z, i, z)) \end{aligned}$$

Hence the formula holds for $x = j + 1$, and by induction, it holds for all n.

Lemma 4:

$$q(x, n, x) = (x+1)^n \tag{13}$$

Proof: The proof is by strong induction on n. The base case is $n = 1$:

$$\begin{aligned} q(x, 1, x) &= 1 + q(x-1, 1, x) + \sum_{i=1}^{0} q(x, i, x), \text{ by (9)} \\ &= 1 + q(x-1, 1, x) \\ &= 1 + 1 + (x-1)(1 + \sum_{i=1}^{0} q(x, i, x)), \text{ by (11)} \\ &= (x+1)^1 \end{aligned}$$

Inductive case: Suppose the formula holds for all $1 \leq n \leq j$, i.e.

$$q(x, n, x) = (x+1)^n, \text{if } n \leq j \tag{14}$$

$$\begin{aligned} q(x, j+1, x) &= 1 + q(x-1, j+1, x) + \sum_{i=1}^{j} q(x, i, x), \text{ by (9)} \\ &= 1 + 1 + (x-1)(1 + \sum_{i=1}^{j} q(x, i, x)) + \sum_{i=1}^{j} q(x, i, x), \text{ by (11)} \\ &= x + 1 + x \sum_{i=1}^{j} q(x, i, x) \\ &= x + 1 + x \sum_{i=1}^{j} (x+1)^i, \text{ by (14), given } i \leq j \end{aligned}$$

$$= x + 1 + x\frac{(x+1)^{j+1} - (x+1)}{x}, \text{ sum of a geometric series}$$
$$= (x+1)^{j+1}$$

Hence the formula holds for $n = j + 1$, and by induction, it holds for all n.

Theorem 2:

$$p(m,n) = \sum_{i=1}^{m} i^n$$

Proof:

$$p(m,n) = \sum_{i=0}^{m-1} q(i,n,i), \text{ by (10)}$$
$$= \sum_{i=0}^{m-1} (i+1)^n, \text{ by (13)}$$
$$= \sum_{i=1}^{m} i^n$$

Thus the illegal example in Fig. 7 has complexity that is $O(m^n)$. We expect that this would be worse than any legal set of points.

6 Conclusions and Future Work

We have proved that, for a set of m points in n objectives, while LebMeasure is polynomial in m, it is exponential in n:

- we have given data that exhibits exponential slowdown as n increases;
- we have proved that the pattern of sets of points in Fig. 5 generates 2^{n-1} contributing points, giving a lower-bound on the worst-case complexity;
- we have proved that the pattern of illegal sets of points in Fig. 7 generates $O(m^n)$ contributing points, which is likely to be worse than any legal set.

Thus LebMeasure (as it stands) cannot provide a general solution to the performance problem (currently) inherent in calculating hypervolume.

As discussed in Section 3, the performance of LebMeasure can be improved (sometimes dramatically) by presenting the points to the algorithm in the best order (although the worst-case complexity is still exponential). Points with many poor objective values generate few non-dominated spawns, and should be processed first. We have developed some initial heuristics to exploit this.

Another promising line of research is in re-ordering the objective values in the points before calculating their hypervolume. This too can deliver significant performance improvement for some sets of points.

Finally, we speculate that no polynomial-time algorithm exists that can calculate hypervolume exactly, due to the irregularity of the shapes in the worst case. We would love to be able to prove this!

Acknowledgments

We would like to thank Phil Hingston, Mark Fleischer, and Simon Huband for helpful discussions on LebMeasure and its behaviour.

References

1. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE TEC*, 6(2):182–97, 2002.
2. M. Fleischer. The measure of Pareto optima: Applications to multi-objective metaheuristics. Technical Report ISR TR 2002-32, University of Maryland, 2002.
3. M. Fleischer. The measure of Pareto optima: Applications to multi-objective metaheuristics. *EMO 2003*, vol 2632 of *LNCS*, pp 519–33. Springer-Verlag, 2003.
4. S. Huband, P. Hingston, L. While, and L. Barone. An evolution strategy with probabilistic mutation for multi-objective optimization. *CEC'03*, vol 4, pp 2284–91. IEEE, 2003.
5. J. Knowles and D. Corne. M-PAES: A memetic algorithm for multi-objective optimization. In *CEC'00*, vol 1, pp 325–32. IEEE, 2000.
6. J. Knowles, D. Corne, and M. Fleischer. Bounded archiving using the Lebesgue measure. *CEC'03*, vol 4, pp 2490–7. IEEE, 2003.
7. T. Okabe, Y. Jin, and B. Sendhoff. A critical survey of performance indices for multi-objective optimisation. *CEC'03*, vol 2, pp 878–85. IEEE, 2003.
8. R. Purshouse. *On the evolutionary optimisation of many objectives*. PhD thesis, The University of Sheffield, 2003.
9. J. Wu and S. Azarm. Metrics for quality assessment of a multi-objective design optimization solution set. *Journal of Mechanical Design*, 123:18–25, 2001.
10. E. Zitzler. *Evolutionary algorithms for multi-objective optimization: Methods and applications*. PhD thesis, Swiss Federal Inst of Technology (ETH) Zurich, 1999.
11. E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm for multi-objective optimization. *EUROGEN 2001*, pp 95–100. Int Center for Numerical Methods in Engineering, Barcelona, 2001.
12. E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance assessment of multi-objective optimizers: An analysis and review. *IEEE TEC*, 7(2):117–32, 2003.