Quick Hypervolume

Luís M. S. Russo and Alexandre P. Francisco

***Abstract*—In this paper, we present a new algorithm for calculating exact hypervolumes. Given a set of d-dimensional points, it computes the hypervolume of the dominated space. Determining this value is an important subroutine of multiobjective evolutionary algorithms. We analyze the quick hypervolume (QHV) algorithm theoretically and experimentally. The theoretical results are a significant contribution to the current state of the art. Moreover, the experimental performance is also very competitive, compared with existing exact hypervolume algorithms.**

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

I. Introduction

IN THIS PAPER, we focus on problems that optimize several objectives at the same time. Most of the time these objectives conflict with each other, meaning that optimizing one objective implies a loss of performance in another. An illustrative example is the problem of sensor placement to detect outbreaks on networks as quickly as possible, such as the spreading of information on online social networks or of viruses in water distribution networks [1]. For the sake of simplicity, let us consider just two objectives: detection time and detection likelihood. Since in most practical cases, we are not allowed to place sensors on all network nodes, because of either performance or budget issues, we must carefully place sensors on a few nodes taking into account our objectives. On one hand, we would like to detect as many events as possible and, thus, we would like to place the sensors on nodes that allow us to reach all other nodes. On the other hand, we would like to detect events as soon as possible and, thus, we would like to place the sensors on nodes close to the origin of the outbreaks, which are usually distinct from former ones. Even in this simple setting, it becomes clear that deciding on sensor placement is not simple and that we must deal with a multiobjective optimization problem. Note that, in general, the problem of sensor placement involves many more objectives, becoming much harder.

As the number of objectives and of items under analysis increases, the complexity of the problem increases considerably. Namely, the time the problem takes to be solved, due to the large number of possible choices. We seem to have intuitive knowledge of this complexity. From a psychological point of view, this may have the negative impact of increasing anxiety [2]. Interestingly, as the amount of choice increases so do the artifacts people use for coping with complexity.

Multiobjective evolutionary algorithms (MOEAs) [3] solve multiobjective optimization problems, which occur in a wide range of problems including, scheduling, economics, finance, automatic cell planning, traveling salesman, etc. Updated surveys on these algorithms are readily available [4], [5]. There is a class of MOEAs in which we are particularly interested because they use indicators to guide their decisions, in particular, they may use the hypervolume [6]–[9].

We study the complexity of the algorithms that compute hypervolumes, specifically the space and time performance. We obtain the following results:

1) Section III describes QHV, a new divide and conquer algorithm for computing hypervolumes. The algorithm is fairly simple, although it requires some implementation details, explained in Section IV-A.
2) Section IV-B includes a theoretical analysis of QHV. Assuming the points are uniformly distributed on a hypersphere or hyperplane, the analysis shows that QHV takes $O(dn^{1.1}\log^{d-2} n)$ time to solve a hypervolume problem, with n points in d dimensions. This bound holds for the average case and with high probability. The power in n can be made as close to 1 as desired, our actual bound is $O(dn^{1+\epsilon}\log^{d-2} n)$, and $\epsilon > 0$ can be chosen arbitrarily small.
3) We study this performance experimentally. Our QHV prototype is extremely competitive against state-of-the-art hypervolume algorithms (see Section IV-C).

The paper includes a small review of related literature in Section V. The paper ends with a brief discussion and conclusions in Section VI.

Let us move on to the hypervolume problem.

II. Problem

Given a set of d-dimensional points, we seek to compute the hypervolume of the dominated space. This section describes the hypervolume calculation problem. Fig. 1 shows a set of points and the respective 2-D hypervolume, commonly referred to as area.

The region of space under consideration is delimited by a rectangle with opposing vertexes z and o, that are close to 0 and 1, respectively. We consider only rectangles that are parallel to the axis.

Manuscript received July 12, 2012; revised November 29, 2012 and March 22, 2013; accepted June 2, 2013. Date of publication September 11, 2013; date of current version July 29, 2014. This work was supported by the National Funds through FCT-Fundação para a Ciência e a Tecnologia, under Grant PEst-OE/EEI/LA0021/2013, Grant TAGS PTDC/EIA-EIA/112283/2009, Grant NetDyn PTDC/EIA-EIA/118533/2010, and Grant HELIX, PTDC/EEA-ELC/113999/2009.

The authors are with INESC-ID/KDBIO and Departmento de Engenharia Informática, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisbon 1049-001, Portugal (e-mail: lsr@kdbio.inesc-id.pt; aplf@kdbio.inesc-id.pt).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2013.2281525

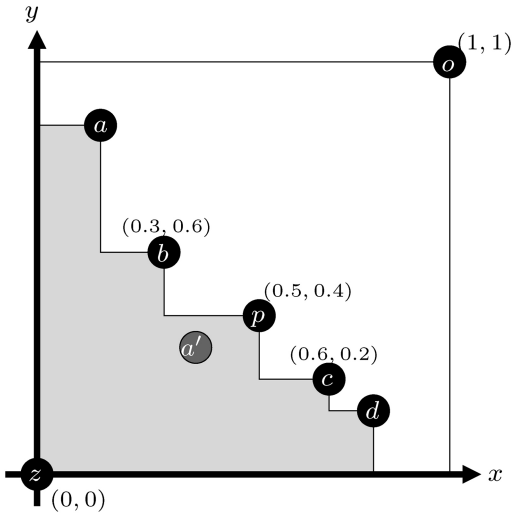

Fig. 1. Area of a set of 2-D points.

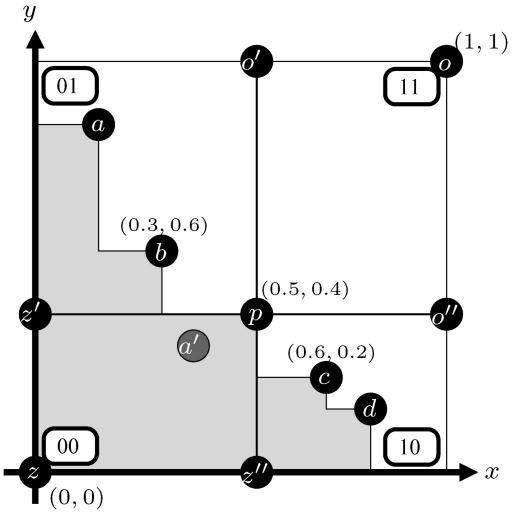

Fig. 2. Pivot divide and conquer for 2-D points. The quadrants are labeled by binary numbers.

We say that point p dominates[1] point a' because a' is contained in the rectangle of vertexes z and p. Notice that we cannot state that d dominates a', since the rectangle with vertexes z and d does not contain a'.

Given a set of points, in our example $\{a, a', b, p, c, d\}$, we want to compute the dominated area, shown in gray in Fig. 1. In general, we use hyper-rectangles, instead of rectangles, to define dominance between points. The point z is always a vertex of these hyper-rectangles. The opposing vertex is a point p, from the set. Therefore, the problem is to compute the hypervolume occupied by the hyper-rectangles that use the points in the set, but without counting the dominated space twice, just like in the 2-D example.

The coordinates can be any reals in $[0, 1]$. Our algorithm uses a subroutine to eliminate dominated points, so we do not insist on having a set of nondominated points.

The mathematically inclined reader may prefer formal definitions. Consider the d-dimensional space $[0, 1]^d \subseteq (\mathbb{R}^+)^d$. Let us assume that z is the point where all coordinates are 0. If $p = (p_1, \ldots, p_d) \in [0, 1]^d$ and $a = (a_1, \ldots, p_d) \in [0, 1]^d$ we say that p dominates a, denoted $a \preceq p$ if $a \neq p$ and $a_i \leq p_i$ for all $i \in \{1, \ldots, d\}$. The hyper-rectangle associated with p consists of the set $\mathcal{H}(p) = \{q \in [0, 1]^d | q \preceq p\}$. Given a set of points $\mathcal{S} \subseteq [0, 1]^d$, the hypervolume problem consists of computing $\lambda(\cup_{r \in \mathcal{S}} \mathcal{H}(r))$, where λ represents the Lebesgue measure of the set. For computational reasons, $\mathcal{S}$ should be finite, we use n to represent its size.

III. Pivot Divide and Conquer

In this section, we describe the QHV algorithm, by working our way from 2-D to higher dimensions and gradually introducing the necessary concepts. Pivot divide and conquer is the technique used by QuickSort [10]. The process consists of the following three steps.

1) Select a special pivot point. This point is processed and excluded from the recursion.
2) Divide the space according to the pivot, more precisely classify points into the possible space regions.
3) Recursively solve each of the subproblems in the smaller regions of space, and add up the hypervolumes.

[1]A point does not dominate another point that has the same coordinates, this is the sole exception to the rectangle criterion we gave.

A. 2-D Case

Fig. 2 shows an example of this process in 2-D. First, we choose point p to be the pivot. Second, we divide the rectangle, of vertexes z and o, according to p. Third, we recursively compute the area of the points in quadrants 01 and 10.

Let us now focus on the details of each step. In the first step, it is necessary to choose a pivot p that is not dominated by any other point p'. Naively comparing all pairs of points would yield $O(n^2)$ time. Instead we compute the area of the rectangles with vertexes z and each point p'. We choose p to be the point with the largest dominated area. Hence, p is nondominated by any other p', meaning that the quadrant 11 will contain no other point.

The space is divided into quadrants, according to the pivot. The quadrants are labeled in binary. The first bit is used for the x-coordinate and the second bit for the y-coordinate. A 0 indicates that, for the given coordinate, the points of the quadrant are between z and p. Otherwise, the points of the quadrant are between p and o and we use a 1. In our example with $p = (0.5, 0.4)$, point $c = (0.6, 0.2)$ is in the quadrant 10, because $0.5 < 0.6$ and $0.4 > 0.2$. We can classify a point, by comparing it with p.

The conquer procedure is called recursively for quadrants 10 and 01. Each recursive call uses its own bounding rectangle and the respective list of points. Assume that the points z' and o' are the vertexes of the bounding rectangle of quadrant 01 and likewise points z'', o'' bound quadrant 10. The total area is obtained by adding the results of the smaller subproblems with the area of the pivot. In our example this yields

$$\begin{aligned}\mathrm{QHV}(z, o, \{a, a', b, p, c, d\}) = {} & \mathrm{QHV}(z', o', \{a, b\}) \\ & + \mathrm{QHV}(z'', o'', \{c, d\}) \\ & + 0.5 \times 0.4.\end{aligned}$$

Notice that point a' is discarded because it belongs to quadrant 00, and, therefore, is dominated by p. This is one way by which QHV identifies dominated points.

For 2-D, this procedure behaves as follows.

Theorem 1: The QHV algorithm determines the area dominated by n points of a 2-D space in $O(n \log n)$ expected time.

The analysis is essentially the same as for QuickSort. Notice that the selection of the pivot requires $O(n)$ time, instead of the $O(1)$ of QuickSort. This does not change the asymptotic complexity because QuickSort spends this time in the division.

This result is not competitive when the points are already sorted by a coordinate. In that case, the respective area can be computed in $O(n)$ time and we explain how in Section V. Still, if the points are given in random order, the sorting step requires $O(n \log n)$ time, thus, the overall time is the same as for QHV.

B. Higher Dimensions

For the sake of discussion, let us consider, for a moment, that p was not a point of the original set. This means that we were partitioning the space according to an external pivot. The relation in the previous equation would now be inaccurate, because of the 0.5×0.4 term. This term is related to the 00 quadrant, for which the contributing area is now different. To determine the correct area, we need to project the points b and c to that quadrant. Point b is projected to $b' = (0.3, 0.4)$, point c is projected to $c' = (0.5, 0.2)$ (see Fig. 3). In this case, projecting point b means finding the point of the 00 quadrant that is dominated by b and that dominates all other points of that quadrant that are also dominated by b. Operationally determining the coordinates of the projected points consists of replacing some of the coordinates with the values of the pivot. For point b, we replaced the y-coordinate and for point c, we replaced the x-coordinate. Another way to understand how projections work is to consider set intersection. Let A be the rectangle with vertexes z and b and B the rectangle with vertexes z and p. The projection b' is the upper right vertex of $A \cap B$. This notion of projection cannot, however, be used when we are given the quadrant number instead of the vertex p.

In this hypothetical scenario, the last term in the equation could be replaced by QHV$(z, p, \{a', b', c'\})$. In 2-D, it is redundant to project the points a and d, but the same is not necessarily true in higher dimensions. One way to realize this claim is to look at Fig. 3 as the top view of a 3-D space, meaning that the points may have distinct z coordinates. In this case, the division lines, the ones that intercept at p, are 2-D planes.

Point projections are indeed necessary for higher dimensions. Points from quadrants 01 and 10 can be projected into the 00 quadrant. During the course of the execution no point will be classified into the 11 quadrant, because such a point would dominate p and, hence, violate the pivot selection criteria. Still if there existed a fictitious point f in the 11 quadrant, it could be projected into the 01 and 10 quadrants, and by transitivity into 00, the resulting point would be p, no matter what point of 11 is projected.

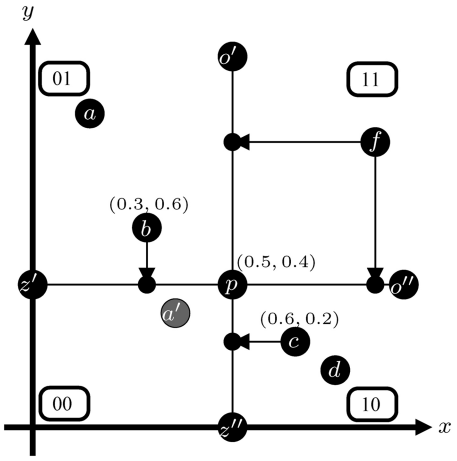

Fig. 3. Projection of points b and c to the 00 quadrant and of point f to quadrants 01 and 10.

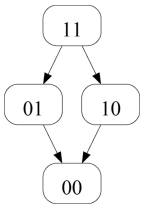

Fig. 4. 2-D Hasse diagram.

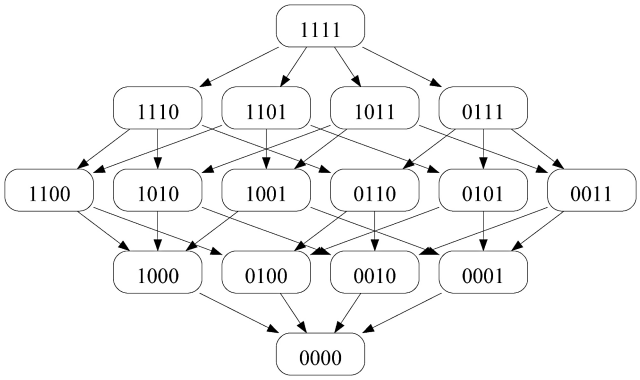

Fig. 5. 4-D Hasse diagram.

An important observation is that a point from the 01 quadrant cannot be, meaningfully, projected into the 10 quadrant, and vice versa. These properties can be represented graphically by a Hasse diagram (see Fig. 4). The diagram represents the quadrants as vertexes and the fact that points from a quadrant can be projected into another by arrows. This diagram can be generalized into higher dimensions. Fig. 5 shows the Hasse diagram of a 4-D space.

Although this latter diagram looks intricate the underlying logic is simple. Each of the nodes represents a region of space, referred to as hyperoctants instead of quadrants. The hyperoctant 0000 is the one that contains the bounding vertex z, likewise 1111 contains o. Hyperoctants 1110 and 1010 are adjacent in the diagram, because 1010 is obtained by flipping the second bit from 1 to 0. Adjacent hyperoctants differ by exactly 1 bit. This means that the points of 1110 can be projected

into 1010, by replacing the second coordinate. Moreover, by also replacing the first coordinate the points of 1110 can be projected into 0010. In general the points of a hyperoctant h can be projected into h' iff $h \,\&\, h' = h'$, where & is the bitwise AND operation. For example the points of 1110 cannot be projected into 0101, since $1110 \,\&\, 0101 = 0100 \neq 0101$. The & operation exposes that, for the last coordinate, a point of 1110 is before the pivot whereas a point of 0101 is after the pivot. Hence, the last coordinate invalidates projections from 1110 to 0101.

In d dimensions QHV is generalized as follows.

1) Pivot selection is essentially the same procedure as before, computing hypervolumes, instead of areas.
2) Dividing the problem into subproblems consists in classifying and projecting points. Point classification is essentially the same as before. Comparing p' with the pivot involves comparing all d dimensions, and now there are 2^d possible hyperoctants. Moreover, each hyperoctant is also assigned all possible point projections. For instance hyperoctant 0001 includes the projections of all the points assigned to 0011, 0101, 0111, 1001, 1011, and 1101. Therefore, a point, by means of projections, has influence on several subproblems.
3) The conquer step is performed recursively, but now it is slightly more complex. There are no points in 1111, or in general in hyperoctant $\top$, therefore, there is no recursive call into this region. There is also no recursive call to 0000, in general, we represent this hyperoctant as $\perp$. For $\perp$ we use the hypervolume of the pivot.[2] If there are any points in this hyperoctant they are dominated by p and, therefore, discarded. For any other hyperoctant h, we compute a recursive call.

In the recursive call, it is also necessary to compute the appropriate bounding vertexes z' and o'. Vertex o' is obtained as the projection of o into the respective hyperoctant. Vertex z' is a dual projection of z, meaning that the coordinates are projected away from z, instead of toward it.

IV. Analysis

In this section, we study the performance of the QHV algorithm in terms of space and time requirements. We start by discussing some engineering considerations related to the implementation of QHV. Then, we move on to the theoretical analysis, which includes an average case bound and a high probability bound. We finish the section with an experimental validation.

A. Implementation

To obtain an efficient implementation of QHV it is necessary to be mindful of several design issues. The space requirements of QHV vary significantly between the best case, the worst case and the average case, as we will see in Section IV-B. Therefore, we use dynamic memory when storing the list of point indexes. In fact all the indexes of the successive recursive calls are stored in an array inside the splitter structure S. The size of this array can grow beyond $O(n)$. Still, the lists inside this array are handled in a stack-like fashion, meaning that the lists are removed from the array once their recursive call finishes. Removing means that the space is considered available to be overwritten by other lists. Naive management of memory, by allocating list every time, can easily consume 25% of execution time.

Next, we show our C implementation of QHV, the reader that is unfamiliar with C, or uninterested in the implementation details, can still safely follow the text. Even though the implementation is in C it was written for readability.

The implementation follows the three steps we mentioned in Section III. A recursive call of `quickHVolumeR` receives the bounding hyper-rectangle, given in points `z` and `o`; the number of points `n`; an array with the `n` point indexes `idx`, for this call; and a pointer to the array containing all points `PS`.

An implementation of QHV in C is the following:

```
static double
quickHVolumeR(point * z,
              point * o,
              int n,
              int *idx, point * PS)
{
  splitter S; /* The splitting structure */
  int i, j;   /* Counters */
  point p;    /* Pivot */
  point pp;   /* Temporary point */
  point newo; /* The new o point */
  point newz; /* The new z point */
  double hv;  /* HyperVolume */
  double max; /* Current maximum */

  max = -DBL_MAX; /* Initialization */

  ///1. Select Pivot
  i = 0;
  while(i < n)
    {
      intercept(&pp, o, &(PS[idx[i]]));
      hv = objective(z, &pp, o);
      if(hv > max)
        { /* New Pivot Candidate */
          max = hv;
          j = i;
        }
      i++;
    }

  /// Separate Pivot
  intercept(&p, o, &(PS[idx[j]]));
  hv = HV(z, &p);
  drop(idx, &n, j);

  /// 2. Divide Points
  S = newSplitter();
  i = 0;
  while(i < n)
    {
      intercept(&pp, o, &(PS[idx[i]]));
      newSets(classify(&p, &pp));
      while(hasNextSet())
        push(S, idx[i], nextSet());
      i++;
    }
```

[2]Hypothetical external pivot was used only to illustrate projections.

```
  split(S);

  /// 3. Conquer
  while(hasNext(S))
    {
      idx = next(S, &j, &n);
      projectZero(&newz, &p, j, z);
      projectOne(&newo, &p, j, o);

      hv += quickHVolumeR(
         &newz, &newo, n, idx, PS);
    }
  freeSplitter(S);

  return hv;
}
```

The selection step chooses the point `p`, with index `j`, as the one that maximizes a given `objective`. Usually the hypervolume, i.e., `HV(z, &pp)`, but other pivot selection functions are viable. We always keep the original points, we do not store projections, instead we pass the `idx` array. Therefore, before using a point, it is necessary to project it into the bounding hyper-rectangle. These simpler projections are computed with the intercept function, that stores the corresponding point in `pp`, the first argument. The second and third arguments input the vertexes of the original hyper-rectangles, `o` and `&(PS[idx[i]])`. After the pivot is selected, its hypervolume, `HV(z, &p)`, initializes the result value, variable `hv`. Instruction `drop` removes `p` from `idx`.

The division step uses the splitter data structure `S`, to classify the points into the respective hyperoctants. Points are inserted into `S` with push, by indicating the point `idx[i]` and the hyperoctant `nextSet()`. The `newSets` function receives a hyperoctant description, such as 1100, and returns, by successive `nextSet` calls, all the hyperoctants that it can be projected to, for example 1000 and 0100.[3] The classify function determines the hyperoctant, to which `pp` belongs, considering `p` as the pivot. The split operation groups the points into hyperoctants.

The conquer step queries the `S` structure for the new lists of points `idx` and then makes a recursive call to `quickHVolumeR`. The next operation returns this list along with its size `&n` and the respective hyperoctant `&j`. The information in `j` is used to obtain the new bounding hyper-rectangle vertexes, `newz` and `newo`, by calling the `projectZero` and `projectOne` functions.

The classification step is fairly subtle. Given a point, we classify it into a certain hyperoctant h. We then use some bit manipulation to generate all the hyperoctants h', such that h can project into h'. The h' hyperoctants are obtained with the `nextSet` operation. Basically, it consists of keeping a counter from 1 to $2^j - 1$ and mapping the bits from this counter to the bits that are set to 1 in h. Here, j is the number of 1's in the representation of h. This is simple to achieve with an array that stores the different bits of h and using bitwise OR.

[3]This denomination comes from the original use of Hasse diagrams, as a way to represent the subsets of a given set, where 1 means that the element is present and the 0 means that it is absent.

The hardest part of classification consists of grouping all the points of the same hyperoctant together. We store the points in a list, with tags, indicating the hyperoctant to which the point should go. The same point can occur multiple times with different tags. Notice that this list may be of size $O(n2^{d-1})$, in the worst case, because a point can project to potentially another $2^{d-1} - 2$ hyperoctants. Assume the list is of size ℓ, sorting this list according to tags yields $O(\ell \log \ell)$ time. We eliminate the $\log \ell$ factor as follows. Assuming that d is small enough, in our tests $d \leq 13$, we can use count sort, which takes $O(2^d + \ell)$ time, and $O(2^d)$ extra space for storing the counters. The size of the list varies considerably, between 2 and $O(2^{d-1}n)$. Clearly, we cannot afford to have an $O(2^d)$ fixed time cost in all sorts, particularly because this procedure occurs most often when ℓ is small. This would be much worse than the $\log \ell$ factor we are trying to reduce. To eliminate this cost, we use an adaptation of the structure by Briggs and Torczon [11]–[13]. This structure can be used to avoid initializing the counters. That is not exactly the way in which we use it. All counters are initialized to 0 once when the algorithm starts. Everytime the count sort algorithm finishes, it cleans the counters, only the ones that were used in that particular sort. Hence, the counter array can be reused without the need for a complete clean up. The structure is used in the accumulating step of the count sort. Scanning all the possible counters would also yield an $O(2^d)$ time cost, instead we use this structure to scan only the counters that were affected by some point of the list. Hence, we obtain $O(\ell)$ sorting time.

An important optimization is removing dominated points. We have already seen that the points that end up in $\perp$ are dominated and therefore removed, by not doing recursion into this hyperoctant. Still some points that are nondominated might have dominated projections. Recall that in the hypothetical external pivot discussion of Section III-A the projection of point d was removed from the recursion into the 00 quadrant, precisely because the projection was dominated by the projection of c. Hence, after grouping the points by hyperoctant, the splitter structure `S` runs an algorithm to remove dominated points. Since the number of points in each list is usually small, particularly when d is large, this is computed naively, by comparing all pairs of points, in $O(dn^2)$ time. This can be improved to $O(n \log^{d-2} n)$ time with the multidimensional divide and conquer algorithm, that we discuss in Section V. In practice, this is not critical for the overall performance, although the following detail was. Suppose that we project a 4-D point from 1110 into 1010 and the respective projection becomes dominated in this latter hyperoctant, then it is not necessary to further project this point into 1000 or 0010 because the resulting points would still be dominated. This procedure is further explored in Lemma 1. The C implementation of QHV that we presented does not reflect this process. The code generates all possible projections, by means of the `nextSet` function. In our prototype, we proceed in a slightly different way. The hyperoctants are processed by height, from highest to lowest, where the height is the number of 1's in the hyperoctant representation. For example, in 4-D, we first process 1110, 1101, 1011, and 0111, these are the hyperoctants of height 3. From

these hyperoctants, we project the nondominated points to the respective hyperoctants of height 2, and from those we project to the hyperoctants of height 1. This means that the count sort, the projection generation and determining dominated points is computed interleaved by decreasing height. This procedure is computed with the split command. Therefore, in our prototype, there is no call to `nextSet` in the `quickHVolumeR`, but for exposition purposes the original version is less obscure. Moreover, this issue is crucial for the analysis we present in Section IV-B.

A crucial design issue is the algorithm for small subproblems. The QHV algorithm can be used until there is only one point in the recursive call. This would be inefficient, for small subproblems, it is better to use another algorithm, which can have worse asymptotic complexity, but is more efficient for small problems. Once again our theoretical analysis, in Section IV-B, reflects this issue, because the theoretical bounds hold only for values of n larger than some n_0. Moreover, the time that it takes to solve such small problems is an important multiplicative factor of the whole algorithm. We also take advantage of computer architecture features, such as bit manipulation and the SSE3 instruction set. This is crucial as QHV can spend 90%, of the time, on the small subproblems.

In QuickSort, this is a known issue and, usually, when the subproblems are of size 16, it is better to use insertion sort. In the QHV prototype, we implemented two alternatives. For small dimensions or few points, we use a cache oblivious HSO [14]. For higher dimensions, we use inclusion exclusion (IEX). A description of these algorithms is given in Section V, for now it is only relevant that their time complexity is $O(n^{d-1})$ for HSO and $O(d2^n)$ for IEX. Clearly IEX scales very badly when the number of points increases. The size of the subproblems varies, but for higher dimensions it is close to 10. A rough analysis, of such problems, for 9-D shows that HSO requires less than 100 million ops, whereas IEX requires at most 10 thousand ops. Hence, the choice of IEX over HSO, in high dimensions. This does not mean that HSO is 10 000 times slower than IEX, because the bound of HSO might be loose or have a small constant, but it is reasonable to expect that in this case HSO is indeed much slower than IEX. To decide which algorithm we should use, we could use their asymptotic bounds to estimate the number of operations each algorithm requires and select the algorithm that requires less operations. Still this would not account for the underlying constants. Therefore, we executed a series of tests, varying n from 1 to 10 and d from 3 to 15, to determine, for that configuration, which algorithm was fastest most of the time. We hard coded resulting selection into our prototype, which, therefore, chooses between IEX and HSO, by simultaneously considering d and the number of points of the subproblem.

Another fundamental issue is the pivot selection criteria. It is important to choose a pivot that divides the points into hyperoctants, such that most points end up in the lower hyperoctants, instead of the higher hyperoctants. In this sense, it is also better to choose a pivot that is nondominated. The number of points that end up in higher hyperoctants depends largely on the Pareto front, and therefore is mostly uncontrollable. Interestingly, the shape of the front seems to be related to the intrinsic complexity of the calculus. We observed that for the same number of points, the shape of the frontier affected simultaneously the performance of QHV and of the WFG [18] algorithm. Other algorithms are less sensitive to this phenomena, namely, HSO [14], mainly because they are already slower.

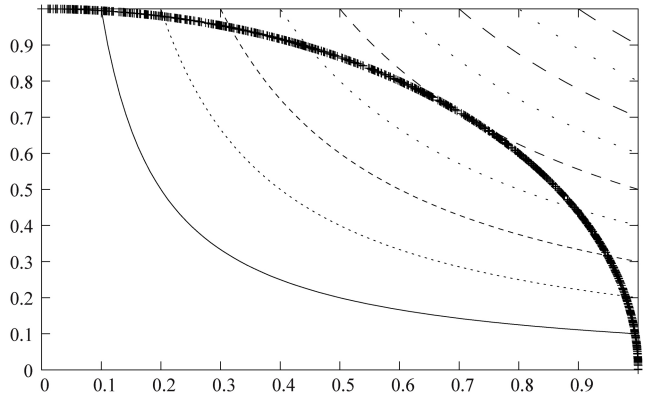

Fig. 6. Spherical Pareto front and the contour lines of points with the same area xy.

Fortunately standard test fronts are usually well behaved, in which case we obtain the best performance. The performance of the algorithm degrades significantly as the fronts become less regular. We observe such phenomena in the experimental results by comparing spherical fronts with random fronts.

To obtain nondominated pivots, we use the hypervolume of the point. On well behaved fronts this criteria is enough to obtain a good distribution of points into hyperoctants. Fig. 6 illustrates this issue, it shows a spherical Pareto front and the contour lines of points with the same area xy. The resulting pivot is the first point to intercept the lines, i.e., the one that intercepts line $xy = 0.5$.

B. Theoretical Analysis

1) *Extreme Cases:* Storing n points, with d dimensions, requires $\Theta(dn)$ space. On top of this space QHV stores the lists of points for the recursive calls. The lists contain the indexes, instead of the points themselves, therefore the size of the lists is independent of d. To organize the points into hyperoctants, we need to store a counter per hyperoctant, therefore we always need $\Theta(2^d)$ extra space. This extra space does not, necessarily, become a term in the time performance, it can be avoided by using a hash to store the nonzero counters. Whenever we measure space recall that recursion is computed with the de facto standard stack.

We use $T(n)$ to represent the time spent by QHV to process n points. The easier cases to analyze are the best and worst case. In the best case QHV processes n points in $\Theta(dn \log^{d-1} n)$ time. This occurs if in the division step the points distribute uniformly among the hyperoctants adjacent to $\perp$. In this case, the recursion equation is $T(n+1) = \Theta(n(d+\log^{d-2} n)) + dT(n/d)$, hence, the time bound. The extra space is $O(n)$. The $\Theta(n \log^{d-2} n)$ time is due to the algorithm that removes dominated points, we explain this algorithm in Section V. In fact for the best case, we could discard

the dominated verification. Assume that the points distribute nicely, in that case the bound is $\Theta(dn \log n)$.

In the worst case, the points distribute among the hyperoctants adjacent to $2^d - 1$. We use the even more pessimistic relation $T(n+1) = \Theta(n(d+\log^{d-2} n)) + 2^d T(n)$, which assumes subproblems of n points in every hyperoctant. This yields a time bound of $O(n(d + \log^{d-2} n)2^{nd})$, by adding a geometric series. In fact, in the worst case, we can also discard the dominated verification and the bound is $O(nd2^{nd})$. The extra space bound is much smaller than the time bound. The size of the point lists in consecutive recursive calls decreases by at least 1. Therefore, the overall list size is $1+2+\ldots+n = O(n^2)$. Still the overall space requirements are larger because each point in the list may project into $2^d - 2$ hyperoctants, and these projections must be stored simultaneously in memory. Therefore, in the worst case, the extra space requirements is $O(2^d n^2)$.

2) *Average Case:* We start by reintroducing the techniques that are used to analyze quicksort. We assume a class of problems and let $T(n)$ represent the time that QHV takes to solve a random problem from this class. Therefore, $T(n)$ is a random variable, instead of a deterministic value, in fact it is a sequence of random variables. Notice that we do not have to worry about problems that are smaller than a given constant n_0, the time that QHV takes to solve one such problem is necessarily bounded by a constant. Therefore, the problem class describes a type of problem, of which there are infinitely many problems. Moreover, a class contains all the subproblems that QHV generates when it partitions the original problem.

To discover the nature of $T(n)$, we study the recursion tree of QHV. A recursion tree is useful to understand what happens when a recurrence is iterated. It is a diagram of the calls a recursive algorithm, such as QHV, uses to solve a given problem. The tree contains a root, with the original problem, and the children of the root contain the subproblems that result from dividing the original problem. Those nodes also contain their subproblems as children, and so on.

We need to pay special attention to the size of the problems in each node of the recursion tree. It is usual to sum the size of a subproblem, i.e., the number of points that it was initially assigned. For a node v, we denote this value as n'_v. We refer to these subproblems as the original subproblems. However, this would yield a harder analysis. Instead we sum the size of the clean problems. Recall that in Section IV-A, we mentioned that each subproblem is initially processed by an algorithm that removes dominated points, including projections. The number of points that are left in a subproblem, after this pruning, is denoted by n_v. We refer to these subproblems as the clean subproblems. These values are important because these points are the points that will be divided by future recursive calls. In a random tree we denote these values as

$$S(n) = \sum_{v \text{ node of the recursion tree}} n_v.$$

Therefore, $S(n)$ is also a random variable. Likewise we denote the sum of the original problems by $S'(n)$. The following Lemma explains why we can restrict our attention to $S(n)$, thus simplifying the analysis.

Lemma 1: Consider a d-dimensional hyper-volume problem with n points. Let $T(n)$ represent the amount of time QHV requires to solve it. Let $S(n)$ and $S'(n)$ be the sum of the clean problems, and of the original problems, respectively. The following relations hold:

1) $T(n) \leq S'(n) \cdot O(\log^{d-2} n) + S(n) \cdot O(d)$;
2) $S'(n) \leq (d+1) \cdot S(n)$;
3) $T(n) \leq S(n) \cdot O(d \log^{d-2} n)$.

Proof: Point 1 is essentially due to the algorithm that eliminates dominated points, which is affected by an $O(\log^{d-2} n)$ factor. This algorithm is applied to every original problem. It is valid to use the distributive property, since no problem can contain more than the initial n points. We also need to account the time of selecting the pivot and dividing the points into the hyperoctants. This involves classifying the points, which takes $O(d)$ time per point. This process is applied to the clean problems.

Point 2 is a more intricate property. Let v be any node of the recursion tree, assume its corresponding hyperoctant is h. Now let $v_1, \ldots, v_d$ be sibling nodes, i.e., all these nodes are obtained by dividing the same problem. Let $h_1, \ldots, h_d$ be the associated hyperoctants. We assume that $h_1, \ldots, h_d$ are immediate ancestors of h in the Hasse diagram. Immediate ancestors means that they are just above h, for example 1101 is an immediate ancestor of 1001, 0101, and 1100, but not of 0001 or any other hyperoctant. Note that there might be d or less such nodes, but not more. Therefore, we have the following relation:

$$n'_v \leq n_v + n_{v_1} + \ldots + n_{v_d}.$$

Since each hyperoctant projects to at most d other hyperoctants, we obtain the following:

$$\begin{aligned} S'(n) &\leq \sum_v (n_v + n_{v_1} + \ldots + n_{v_d}) \\ &\leq (d+1) \cdot \sum_v n_v \\ &= (d+1) \cdot S(n). \end{aligned}$$

Point 3 follows from the previous points, 1 and 2. ∎

Because of this Lemma the remaining analysis focuses on bounding $S(n)$. The resulting bounds are then multiplied by $O(d \log^{d-2} n)$. Hence, it is also reasonable to assume that this factor contains a c_0 constant, such that any problem smaller than n_0 can be solved in $O(c_0)$ time.

To gain some intuition we analyze a good average case. Let us focus first on 2-D, and prove Theorem 1 (see Section III-A). In 2-D, our typical good case verifies the same relation as the best case and, therefore, we aim to prove that $E[T(n)]$ is $\Theta(dn \log n)$. To prove that this bound holds we need to find a large class of regular partitions for which we can bound $E[S(n)]$ by a model, $M(n) = c \cdot n \log n$. We use a Bernoulli indicator random variable P, which assumes value 1 if the partition is regular and 0 otherwise. The probability that the partition is regular, $Pr(P = 1)$, is denoted by p. The class of regular partitions will be gradually refined during the analysis. Still, this class is chosen in a way that probability p is big enough, in fact we assume that QHV behaves in an optimistic

fashion, meaning that if the random partition it obtains at a given point is not regular, the optimistic QHV throws it away and randomly selects another. If there are enough regular partitions, we can use conditional expected values to bound $E[T(n)]$.

Lemma 2: Consider a class of hypervolume problems, such that, a random problem with n points, from this class, has a regular partition with probability at least p. Moreover, determining whether a partition is regular requires at most $c' \cdot n$ time. The expected size of the clean tree of QHV algorithm respects the bound $E[S(n)] \leq M(n)$, provided that

$$E[S(n)|P = 1] \leq M(n) - \left(\frac{1}{p} - 1\right) c'n.$$

Proof: Consider the following derivation:

$$\begin{aligned} E[S(n)] &= p \cdot E[S(n)|P = 1] \\ &\quad + (1 - p) \cdot E[S(n)|P = 0] && (1) \\ E[S(n)] &= p \cdot E[S(n)|P = 1] \\ &\quad + (1 - p) \cdot (E[S(n)] + c'n) && (2) \\ pE[S(n)] &\leq p \cdot E[S(n)|P = 1] + (1 - p)c'n && (3) \\ E[S(n)] &\leq E[S(n)|P = 1] + \left(\frac{1}{p} - 1\right) c'n && (4) \\ E[S(n)] &\leq M(n). && (5) \end{aligned}$$

Equation (1) is a general property of conditional expectations. To obtain (2) we assume $E[S(n)|P = 0] \leq E[S(n)] + c' \cdot n$. This expresses the fact that if we select an irregular partition, the optimistic QHV throws it away and asks for another one, but we still have to sum the cost of verifying if the partition is regular. Equation (5) follows from the previous equation and the hypothesis. ■

Notice that Lemma 2 considers a constant c', which is related to the amount of time it takes to verify whether a partition is regular. This time depends on what is considered a regular partition. Let us analyze the 2-D case. The partitions in 2-D are simple, because the subproblems do not share points. One subproblem gets $f \cdot n$ points and the other subproblem gets $(1 - f) \cdot n$ points. We can, therefore, consider a partition as regular if it is a $3/4$ and $1/4$, or even more balanced, i.e., no subproblem can contain more that $3n/4$ points.

Hence, it is reasonable to assume that we can verify in at most $c' \cdot n$ time whether a partition is regular. We only need to determine how many points exist in each subproblem. Notice that in 2-D, point projections can be ignored. To simplify the exposition, we omit the constant c' by assuming $c' = 1$, this a constant scaling technique meaning that the associated constants need to be multiplied by c' if $c' > 1$, but it does not alter the asymptotic bounds, so we omit it for simplicity.

Proof of Theorem 1: We aim to prove that $E[S(n)] \leq c \cdot n \log n = M(n)$. We choose $c = 2/(2-(3/4)\log 3)$, about 2.5. Recall the constant scaling assumption, without it we should have $c \approx 2.5c'$.

This property is established by induction on n. The base cases are trivial. The induction step uses Lemma 2 with the model $M(n)$. We consider a partition as regular if it is $3/4$, $1/4$, or even more balanced. These partitions occur for at least half the possible pivots, therefore $p = 1/2$. For this value the expression $(1/p) - 1$ is 1. The value of $E[S(n)|P = 1]$ can be complex to compute, therefore, we bound it with the $3/4$, $1/4$ partitions. We can the use the induction hypothesis to produce the following deduction:

$$\begin{aligned} E[S(n)|P = 1] &\leq E[S(n/4) + S(3n/4)] + n \\ &= E[S(n/4)] + E[S(3n/4)] + n \\ &\leq M(n/4) + M(3n/4) + n \\ &= c \cdot (n/4)\log(n/4) \\ &\quad + c \cdot (3n/4)\log(3n/4) + n \\ &= c \cdot n \log n - c(2 - (3/4)\log 3) \cdot n + n \\ &= c \cdot n \log n - n. \end{aligned}$$

This relation satisfies the hypothesis of Lemma 2, from which we conclude that $E[S(n)] \leq M(n) = c \cdot n \log n$, thus establishing the induction step.

We can now use Lemma 1 to prove that $E[T(n)] \leq E[S(n)] \cdot O(d \log^{d-2} n) \leq O(dn \log^{d-1} n)$. ■

The partitions of QHV are not always as simple. For higher dimensions they can be complex. There are several subproblems, which, moreover, can share points among them. Let $F_1 n, \ldots, F_{2^d-1} n$ be a random partition, where the F_i's are random variables. Notice that, because of Lemma 1, we are only interested in clean problems, therefore, $F_1 n, \ldots, F_{2^d-1} n$ represent the size of the clean problems. We simplify the notation by not writing $F_1^{(n)} n$, the associated n value can usually be inferred from the context. The probability that we obtain a partition $f_1 n, \ldots, f_{2^d-1} n$ is written as

$$Pr\left(F_1 n = f_1 n, \ldots, F_{2^d-1} n = f_{2^d-1} n\right).$$

We use these distributions to study $E[T(n)]$ in higher dimensions. Our analysis is structured as follows: the next subsection presents a bound on $E[T(n)]$, and a high probability bound, of $T(n)$, based on abstract properties; the last subsection proposes a distribution of $F_1 n, \ldots, F_{2^d-1} n$, which appropriately models the partitions of QHV. We compute the corresponding values on that distribution and conclude that the expected time of QHV is bounded by $O(dn^{1+\epsilon} \log^{d-2} n)$, even with high probability, for any ϵ as small as desired, $0 < \epsilon < 1$.

3) *General Bounds:* In general, for higher dimensions, a point may occur in more than one subproblem. In 2-D this does not happen, adding up the points in each subproblem yields n. In other words, we used f and $1 - f$, therefore $f + 1 - f = 1$. For example in the previous result we had $3/4 + 1/4 = 1$. In higher dimensions, we allow for the subproblems to share points. Therefore, a partition $f_1, \ldots, f_{2^d-1}$ yields $\sum_{i=1}^{k} f_i \geq 1$. This sum measures how much the subproblems overlap, which is crucial for QHV.

We now need to decide which model $M(n)$ to use. To gain some intuition consider an ideal partition int 2^d hyperoctants. In this case the resulting restriction is

$$E[S(n)|P = 1] = 2^d E[S(n/2^d)] + n.$$

If at least half the partitions were this ideally well distributed a model $M(n) = n \log n$ would be enough. The previous equation becomes $E[S(n)|P = 1] = n \log n - dn$ for this model, which satisfies the hypotheses of Lemma 2. The resulting bound would be $E[S(n)] = O(n \log n)$.

A slightly more realistic model would be $M(n) = n^r$, with $r > 1$. If we use this model, (6) becomes $E[S(n)|P = 1] = (2^{d(1-r)} + n^{1-r}) \cdot n^r - n$, where the expression associated with n^r is smaller than 1 for all $n \geq n_0$ and for some n_0. The resulting bound is therefore $E[T(n)] = O(dn^r \log^{d-2} n)$. We only need to determine a value of r that holds for realistic partitions. We use the following notion.

Definition 1: Consider a class of d-dimensional hypervolume problems. Let $F_1 n, \ldots, F_{2^d-1} n$ be the size of the subproblems obtained when a problem of size n is divided by a random partition, where the F_i's are random variables. A random partition has overlap r if the following relation holds, for some $\delta > 0$

$$\sum_{i=1}^{2^d-2} (F_i n)^r < n^r (1 - \delta).$$

A class has overlap r if at least half its partition have overlap r. The overlap power o of a class is the following limit:

$$\lim_{\substack{0<\delta\to 0 \\ n_0\to+\infty}} \inf_{n_0 \leq n} \left\{ r : Pr\left(\sum_{i=1}^{2^d-2} (F_i n)^r < n^r (1 - \delta) \right) \geq 1/2 \right\}.$$

There is no guarantee that o exists, but if it does, it is a useful value. Notice that $o \geq 1$, for any class. Even, and particularly, if there is overlap, we still need to satisfy $\sum_{i=1}^{2^d-2} f_i \geq 1$, for any partition. Therefore, $r > 1$, although the resulting limit might be 1. If we select the partitions that have overlap r, close to o, as regular, then we can establish a bound on the expected time.

Theorem 2: Consider a class of d-dimensional hypervolume problems, with a finite overlap power o. For any positive $\epsilon > 0$, the optimistic QHV algorithm solves an n point problem, from this class, in $O(dn^{o+\epsilon} \log^{d-2} n)$ expected time.

Proof: Consider an $\epsilon > 0$ and let $r = o + \epsilon$. According to the definition of overlap power we have that there is an n_0 and a δ_1 such that, for any $n \geq n_0$ and $\delta \leq \delta_1$ we have that

$$Pr\left(\sum_{i=1}^{2^d-2} (F_i n)^r < n^r (1 - \delta) \right) \geq 1/2.$$

We define a partition as regular if one of the following equivalent conditions holds:

$$\sum_{i=1}^{2^d-2} c \cdot (f_i n)^r + n \leq c \cdot n^r - n \tag{6}$$

$$c \sum_{i=1}^{2^d-2} (f_i n)^r \leq c \cdot n^r - 2 \cdot n$$

$$\sum_{i=1}^{2^d-2} (f_i n)^r \leq n^r - (2/c) \cdot n$$

$$\sum_{i=1}^{2^d-2} (f_i n)^r \leq n^r \left(1 - \frac{2}{cn^{r-1}}\right)$$

$$\sum_{i=1}^{2^d-2} (f_i n)^r \leq n^r \left(1 - \frac{1}{n^{r-1}}\right).$$

To make the $2/c$ factor disappear, in the last step, we choose $c = 2$. Notice that the subtracted term, on the last equation, converges to 0 because $r - 1 = o + \epsilon - 1 \geq \epsilon > 0$. Therefore, either $1/n^{r-1} \leq \delta_1$, for any $n \geq n_0$, or we choose another n_0, such that $n_0 \geq 1/\sqrt[r-1]{\delta_1}$, which implies $1/n^{r-1} \leq \delta_1$.

Let us now establish that $E[S(n)] \leq M(n) = c \cdot n^r$, like in Theorem 1, by induction on n. The base cases are trivial, we need only establish the result for $n \geq n_0$.

Like before we use the induction hypothesis to deduce that

$$\begin{aligned} E[S(n)|P = 1] &\leq E[\sum_{i=1}^{2^d-2} S(f_i n)] + n \\ &= \sum_{i=1}^{2^d-2} E[S(f_i n)] + n \\ &\leq \sum_{i=1}^{2^d-2} M(f_i n) + n \\ &= \sum_{i=1}^{2^d-2} c \cdot (f_i n)^r + n \\ &\leq c \cdot n^r - n. \end{aligned}$$

In the last step we used the definition of regular partition, i.e., (6). This last relation satisfies the hypothesis of Lemma 2 with $p = 1/2$ and $M(n) = c \cdot n^r$, from which we can conclude that $E[S(n)] \leq M(n) = c \cdot n^r$. Recall that the definition of overlap power implies that at least half the partitions are regular.

We again use Lemma 1 to prove that $E[T(n)] \leq E[S(n)] \cdot O(d \log^{d-2} n) \leq O(dn^r \log^{d-2} n)$. ∎

Let us now further analyze the 2-D case. In this case, there is no overlap among subproblems. The overlap power is 1, for any problem class. To compute this value, we can consider several partitions, but the crucial one is 3/4, 1/4. We need to check that for any $\epsilon > 0$ there is a $\delta > 0$, small enough, such that

$$\left(\frac{3}{4}\right)^{1+\epsilon} + \left(\frac{1}{4}\right)^{1+\epsilon} + \delta < 1.$$

This is necessarily true and therefore the previous Theorem states that in 2-D QHV obtains $O(dn^{1+\epsilon})$ performance for any $\epsilon > 0$. This is coherent with Theorem 1.

This result does not immediately imply the high probability bound. To obtain such a result we require more information regarding the distribution of the partitions. Still we can already prove a crucial property of the recursion tree.

Lemma 3: Consider a class of d-dimensional hypervolume problems, with a finite overlap power o. For any positive number $\epsilon > 0$, the recursion tree of the optimistic QHV algorithm contains at most $n^{o+\epsilon}$ leaves.

Proof: In fact, we prove a slightly more general result. Any subtree of the recursion tree contains at most $n^{o+\epsilon}$ leaves. Let $\epsilon > 0$ be any positive number, and $r = o + \epsilon$.

Recall that n_v represents the size of the clean problem, associated with node v of the recursion tree. Therefore, for any sub-tree, we can compute the following values:

$$\ell = \sum_{v \text{ is a leaf of the sub-tree.}} n_v^r$$

We will now prove, by induction, that $\ell \leq n^r$. The base case is the subtree which contains only the root, this sum yields precisely n^r.

In the general step, we expand the subtrees. The induction hypothesis states that a given subtree respects the bound, i.e., $\ell \leq n^r$. We expand the subtree by including the children (not necessarily all) of a leaf v, but only for one leaf. We denote by ℓ' the new sum. If the leaf in question has an irregular partition this means that it contains only one child v', and that $n_v = n_{v'}$. Therefore, the value of ℓ does not change from one subtree to the other, i.e., $\ell' = \ell \leq n^r$. Otherwise, the leaf is regular. We know that the sum of the new subtree is $\ell' \leq \ell - n_v^r + \sum_{i=1}^{2^d-2}(f_i n_v)^r$, where the f_i coefficients indicate the partition of the problem in v. Since the partition is regular we have that $\sum_{i=1}^{2^d-2}(f_i n_v)^r < n_v^r$ and therefore $\ell' < \ell \leq n^r$.

To conclude, observe that each leaf contains a nonempty problem, otherwise it would not be in the recursion tree. Hence, the number of leaves g, of a given subtree, is smaller than the sum we considered, i.e., $g \leq \ell \leq n^r$. ■

Notice that this result is not an average case result, the bound on the number of leaves holds every time. A bound on the number of leaves is essentially a bound on the width of the tree. Hence, to bound the total size, and therefore the total work, we need to study the height of the tree. To bound the height of the tree, we need that the regular partitions contain more structure. In general, we expect the F_i values to be small, but have not imposed a bound. The next value is used precisely to obtain such a bound, without increasing the overlap power.

Definition 2: Consider a class of d-dimensional hypervolume problems, with a finite overlap power o. The critical balance f_o of this class is given by the following expression:

$$\lim_{\substack{o<r\to o \\ 0<\delta\to 0 \\ n_0\to+\infty}} \inf_{n_0\leq n} \left\{ f : Pr\left(\sum_{i=1}^{2^d-2} (F_i n)^r < n^r(1-\delta) \text{ and } \forall_i F_i \leq f \right) \geq 1/2 \right\}.$$

We can now use the critical balance to establish a high probability bound on the average case of the optimistic QHV.

Theorem 3: Consider a class of d-dimensional hypervolume problems, with a finite overlap power o and critical balance f_o. Let ϵ and ϵ' be positive numbers $\epsilon, \epsilon' > 0$, such that $e(f_o + \epsilon')^{o+1+\epsilon} > 1$. The optimistic QHV algorithm solves an n point hypervolume problem, from this class, in $O((dn^{o+\epsilon}\log^{d-2} n)/(-\log(f_o+\epsilon')))$ time, with at least $1-1/n$ probability. Formally

$$Pr\left(T(n) < \frac{n^{o+\epsilon}}{-\log(f_o+\epsilon')} \cdot O(d\log^{d-2} n) \right) \geq 1 - \frac{1}{n}$$

where $T(n)$ is the time.

The proof is given in Appendix B. It uses the approach in Probability and Computing [15, exercise 4.20]. Our main novelty is Lemma 3.

Now that we obtained general results, we move on to model the distribution of the partitions in QHV. Therefore, we obtain bounds for the overlap power and critical balance of certain classes of problems.

4) *Surface Partitions:* We now have a bound on $E[T(n)]$, and a high probability bound of $T(n)$. These bounds are dependent on the distribution of the partitions $F_1 n, \ldots, F_{2^d-1} n$, namely, on overlap power and critical balance. We will now focus on obtaining these values for surface partitions, i.e., partitions that result from points that are uniformly distributed on a hyperplane or the surface of a hypersphere. Therefore, bounding the performance of QHV on these classes of problems.

Notice that for 2-D we should not model the partitions with binomial distributions, i.e., the F_i variables do not follow a binomial distribution. This happens because the pivot is chosen uniformly at random from the available points. Therefore, there is no bias toward partitions closer to 1/2, 1/2, as would occur in the case of the binomial distribution. In general for d dimensions, we do not model the partition distribution exactly, instead we focus on the overlap power. We bound the probability that the subproblems grow too big.

Let $E_1 n, \ldots, E_{2^d-2} n$ be random variables that model a random partition, without overlaps, i.e., these variables model the fraction of points that is originally assigned to each hyperoctant, i.e., before the projections. We also need to model the points that get projected into an hyperoctant, in which case we assume a surface property. Moreover, we are only interested in the size of the clean problems.

In 3-D, the fraction of points that are projected into an octant is related to the perimeter of the surface. In general, the points that are projected are associated with a subspace of dimension $d-2$. Therefore, we consider binomial random variables $B_1 n, \ldots, B_d n$, with n trials and success probability of $1/\sqrt[d-1]{n}$. Therefore, as n increases, the number of possible projections increases, but the success probability decreases. The B_i's are independent from each other and from the E_i's. The success probability arises from counting how many points are in a $d-2$ space, assuming that the n points are on $d-1$ space, i.e., $n^{(d-2)/(d-1)}/n$.

Notice that our model for projected points is extremely pessimistic, it assumes that every point can be projected into another hyperoctant, with a fairly high probability $1/\sqrt[d-1]{n}$. Note that as d grows this value converges to 1, meaning that for higher dimensions almost all points occur in the frontier of the subspace. This phenomena is known as the curse of dimensionality. Still for QHV it is not that bad, because for $n \geq n_0$ with a large n_0, the probability can be very close to 0. Thus, the performance degrades with d but we are still able to obtain an $O(dn^{o+\epsilon}\log^{d-2} n)$ bound. Our model for a random partition is $(E_1 + B_1)n, \ldots, (E_{2^d-2} + B_{2^d-2})n$. Note that, again, this model is pessimistic, a more realistic model would assume that the frontier of a problem is related to its size. In which case the model would be $E_1(1+B_1)n, \ldots, E_{2^d-2}(1+B_{2^d-2})n$. The resulting restrictions are easier to solve, therefore, we use the pessimistic model.

There are certainly classes of problems that cannot be modeled this way. Therefore, we restrict our attention to hyperplanes and hyperspheres with uniform point distribution, for which the model is appropriate.

Corollary 1: Consider a class of d-dimensional hypervolume problems of points uniformly distributed on a hyperplane or on the surface of a hypersphere. For any positive number $\epsilon > 0$, the optimistic QHV algorithm solves an n point

hypervolume problem, from this class, in $O(dn^{1+\epsilon}\log^{d-2}n)$ expected time.

Proof: The proof follows by showing that the overlap power of this class is 1 and using Theorem 2.

First, let us focus on the $E_1n+\ldots+E_{2^d-2}n$ partitions. We argue that the partitions where none of the E_i's is larger than $7/8$ occur with at least $3/4$ probability. For the 2-D case, this claim is necessarily true, as the pivot is chosen uniformly at random and the resulting partition is $f,1-f$. For higher dimension this bound still applies, just choose two coordinates and ignore the rest. The remaining coordinates only further divide the points. Consider for example 3-D. Assume that fn was the number of points that, in 2-D, ended up in the quadrant 01 and $(1-f)n$ the points that ended up in quadrant 10. Therefore, if we sum the points that in 3-D where assigned to 010 and 011 we will obtain at most fn points, likewise if we sum the points in 100 and 101 we obtain at most $(1-f)n$ points. In general for any dimension d, the probability that some E_in is larger than $7/8$ is smaller than $1/4$.

To bound the B_i's we assume that they are at most $6/\sqrt[d-1]{n}$, i.e., six times the expected value, and use a Chernoff bound [15, Theorem 4.4] to obtain that

$$Pr\left(B_i \geq \frac{6}{\sqrt[d-1]{n}}\right) \leq 2^{-6n^{(d-2)/(d-1)}}.$$

We now use a union bound to limit the probability that some B_i is larger than $6/\sqrt[d-1]{n}$. Let K the random variable that returns the value of the largest B_i in the partition

$$Pr\left(K \geq \frac{6}{\sqrt[d-1]{n}}\right) \leq 2^d 2^{-6n^{(d-2)/(d-1)}}.$$

Therefore, if $n \geq n_0 \geq ((d+2)/6)^{(d-1)/(d-2)}$ we can bound the previous probability and negate the event

$$Pr\left(K \geq \frac{6}{\sqrt[d-1]{n}}\right) \leq 1/4$$
$$Pr\left(\forall_i B_i < \frac{6}{\sqrt[d-1]{n}}\right) \geq 3/4$$
.

Therefore, the probability that all the B_i's are small is at least $3/4$. If we use these partitions to compute the overlap power we obtain the following equation:

$$\left(\frac{7}{8}+\frac{6}{\sqrt[d-1]{n}}\right)^{1+\epsilon}+\left(\frac{1}{8}+\frac{6}{\sqrt[d-1]{n}}\right)^{1+\epsilon}+\frac{2^d-4}{(6/\sqrt[d-1]{n})^{1+\epsilon}}+\delta<1.$$

Moreover, the value on the left side of the equation is the largest such value that we obtain by considering several partitions, that occur with $(3/4)^2 > 1/2$ probability. Now it is only necessary to guarantee that for any $\epsilon > 0$, there is an n_0, big enough, and a $\delta > 0$, small enough, for which the restriction holds. This is again necessarily true. ■

To obtain the high probability bound, we focus on the critical balance associated with the overlap power we have just determined. We cannot determine the exact critical balance but we can upper bound it, by using the previous partitions.

Corollary 2: Consider a class of d-dimensional hypervolume problems of points uniformly distributed on a plane or on the surface of a hypersphere. For any small positive number ϵ, such that $0 < \epsilon < 1$, the optimistic QHV algorithm solves an n point hypervolume problem, from this class, in $O(dn^{1+\epsilon}\log^{d-2}n)$ time, with at least $1-1/n$ probability. Formally

$$Pr\left(T(n) < n^{1+\epsilon}\cdot O(d\log^{d-2}n)\right) \geq 1-\frac{1}{n}$$

where $T(n)$ is the time.

Proof: The result follows from Theorem 3, by determining an upper bound on the critical balance. For any $\epsilon > 0$ and $n \leq n_0$, for a large enough n_0 we can bound the imbalance of the previous partitions as

$$\left(\frac{1}{8}+\frac{6}{\sqrt[d-1]{n}}\right)^{1+\epsilon} \leq \left(\frac{7}{8}+\frac{6}{\sqrt[d-1]{n}}\right)^{1+\epsilon} \leq 29/32.$$

Therefore, $f_o \leq 29/32$. We use Theorem 3 with this value. To make the $1/(-\log(f_o+\epsilon))$ factor disappear inside the O notation we choose $\epsilon' = 1/32$. This yields $f = 15/16$ and a total factor of $1/-\log(15/16) \approx 10.8$. We only need to check that $ef^{2+\epsilon} \geq 1$. This is true provided $\epsilon \leq \frac{\log e}{log(16/15)} - 2 \approx 13.5$. Therefore, we can bound ϵ by 1, which is strictly more restrictive and cleaner. ■

C. Testing

We now present experimental results for estimating the overlap power and compare our QHV prototype with state-of-the-art alternatives.

1) *Estimating Overlap Power:* Although in Corollary 2 we can choose ϵ as small as we want, in practice this result is limited because it may be that very small ϵ values occur only for extremely big n_0 values. In particular n_0 may be larger than typical datasets, which contain from 10 to 1000 points. Therefore, we study the performance of QHV empirically, by estimating the values of $r(n)$ such that $S'(n) < n^{r(n)}$. Recall that $S'(n)$ is the sum of the original problems on a recursion tree, defined in Lemma 1. In fact, we observe that these values seem to vary linearly with d and we use linear regression to determine $a(n)$ and $b(n)$ such that $S'(n) < n^{a(n)+b(n)d}$, for different n values.

By looking at the restriction in Corollary 1 we obtain a hand waiving explanation for this behavior. Consider that ϵ and δ are fixed. Consider n_0 and d values, such that $6/\sqrt[d-1]{n_0}$ is small enough. Now assume we change d to $d+\Delta$, we want to find an n_0' value such that

$$\frac{6}{\sqrt[d-1]{n_0}} = \frac{6}{\sqrt[d-1+\Delta]{n_0'}}$$

the resulting value is $n_0' = n_0^{1+\Delta/(d-1)}$, which shows a linear relation in the exponent.

We used QHV in a simulation mode that keeps a global variable with the accumulated number of points of every recursive call of QHV. The resulting value is $S'(n)$, therefore it should be $O(n^{1+\epsilon})$. We looked at the values $(\log(S'(n)))/\log n$. These values provide a good estimate of $1+\epsilon$ because $(\log c/\log n)+1+\epsilon \geq 1+\epsilon$. In this inequality, c represents the constant hidden in the O notation, which is assumed to be $c \geq 1$. We can pessimistically assume that $c = 1$, which only makes our estimate of $1+\epsilon$ larger than the real value.

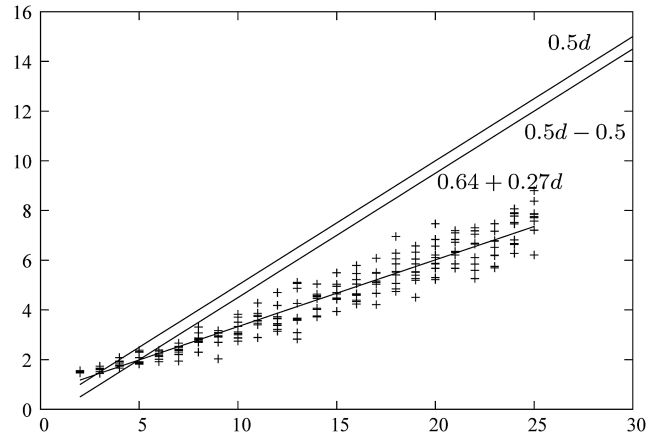

Fig. 7. Spherical front with ten points.

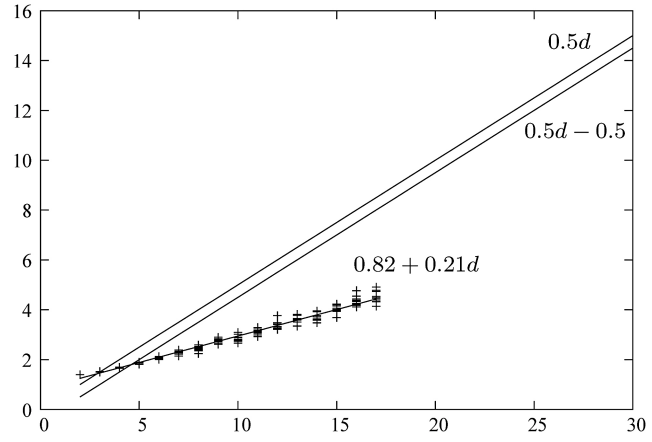

Fig. 8. Spherical front with 100 points.

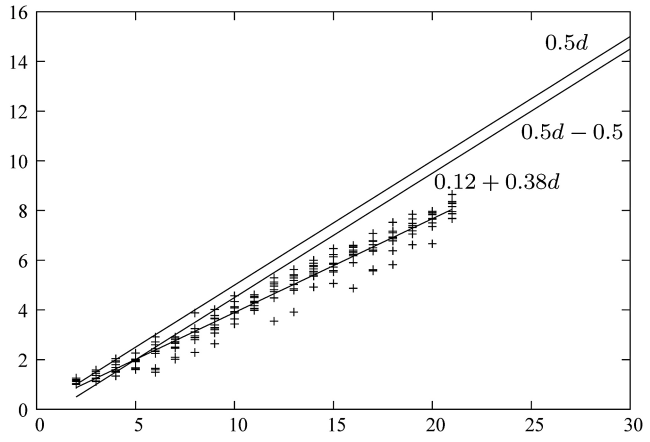

Fig. 9. Random front with ten points.

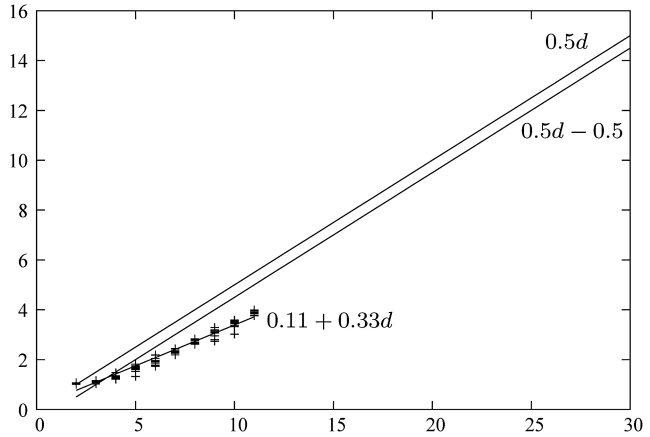

Fig. 10. Random front with 100 points.

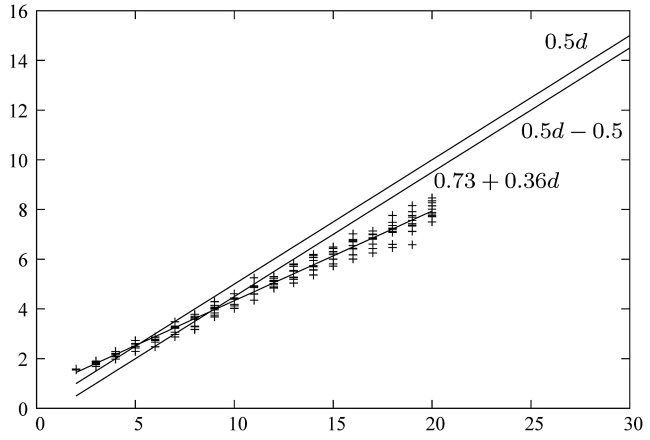

Fig. 11. Linear front with ten points.

We tested ten different fronts for each dimension. The plots are shown in Figs. 7–12. The fronts had ten points and 100 points, respectively. The line $0.5d$ is related to the complexity of the HOY algorithm [16] and the $0.5d - 0.5$ line to the complexity of GBox [17]. We observe, empirically, that the resulting points seem to form a line, i.e, $1+\epsilon = a(n)+d \cdot b(n)$. Therefore, we compute linear regression and obtain empirical expressions for d'. We can see that the resulting values are very sensitive to the number of points and to the shape of the of the front in consideration. Still in all cases the value of $b(n)$ was considerably smaller than 0.5.

2) *Experimental Setting:* Let us now focus on the system time and space performance of the algorithm. In particular, we will show how the techniques proposed in the previous section affect performance. Our implementation of the QHV algorithm is available at http://kdbio.inesc-id.pt/ lsr/QHV/

All results were obtained on a quad-core processor at 3.20GHz, with 256KB of L1 cache, 1MB of L2 cache, 8MB of L3 cache, and 8GB of main memory. The prototypes were compiled with `gcc` 4.7.1. For QHV we used `-msse2` flag to support SSE2 and passed the cache line size into the code `-DCLS = $$(getconf LEVEL1_DCACHE_LINESIZE)`, this was used to align memory to cache lines. The dimension was also determined at compile time, so there is a different binary for each dimension, this allows for better loop unrolling. Each one of the other prototypes contains a sophisticated makefile that produces optimized binaries, it automatically selects the best flags for a given platform. We used the optimized binaries produced by those makefiles, for our platform. Besides the proper flags we included `-s -static -m32 -O9 -march=core2`.

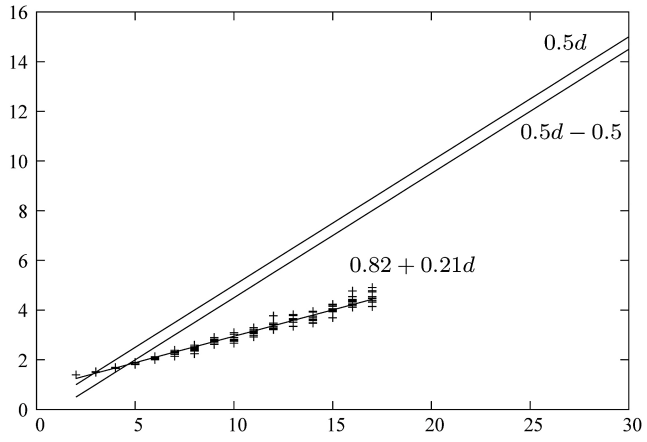

Fig. 12. Linear front with 100 points.

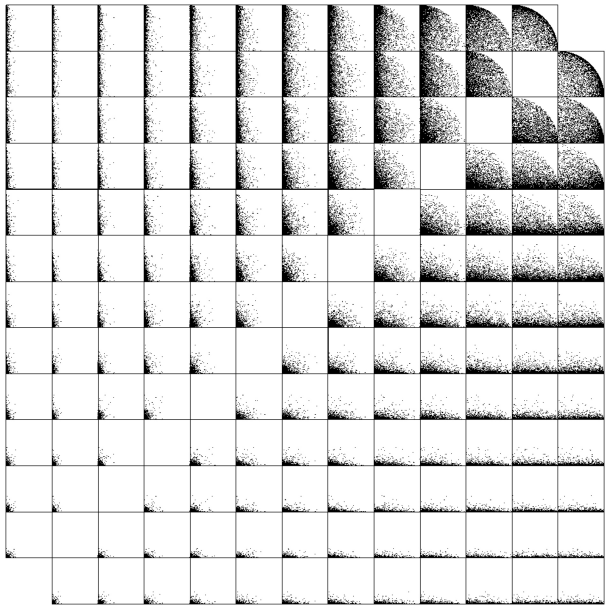

Fig. 13. Matrix of plots for the 13-D spherical points of the WFG dataset. The file contained around 300 points and a similar distribution can be observed in the remaining files of the dataset.

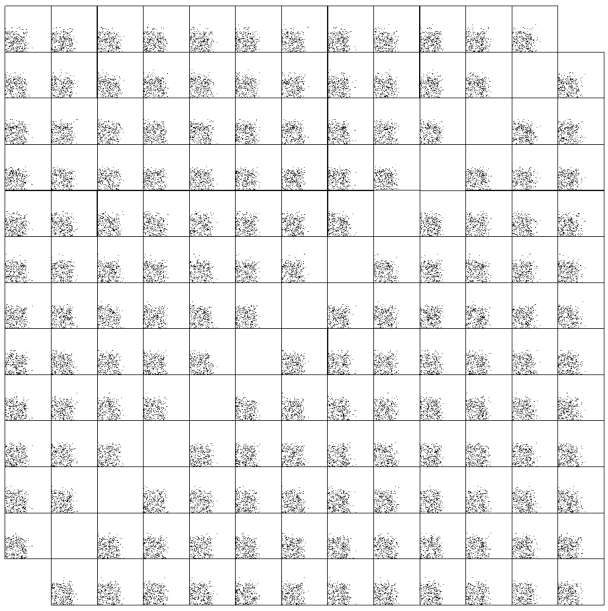

Fig. 14. Matrix of plots for the 13-D spherical points of our dataset. The file contained 300 points.

TABLE I
MEMORY PEAKS OF DIFFERENT ALGORITHMS ON SPHERICAL FRONTS WITH 1000 POINTS

Heap Peaks in KB

	3D	4D	5D	6D	7D	10D	13D
QHV	28.7	37.0	66.3	129.3	242.3	795.1	311.8
WFG	125.1	195.4	273.6	359.6	453.3	781.5	1180.0
FPRAS	51.7	59.5	67.3	75.2	83.0	107.0	131.1
HV	168.2	199.5	230.8	262.1			
HV4D		121.2					

Stack Peaks in KB

QHV	2.4	2.4	4.8	4.9	7.1	12.4	16.3
WFG	9.5	9.5	9.5	9.5	9.5	9.5	9.5
FPRAS	1.4	1.4	1.4	1.4	1.4	1.4	1.4
HV	9.8	9.8	9.8	9.8			
HV4D		9.6					

In the experimental evaluation, we used the benchmarks available from [18] (http://www.wfg.csse.uwa.edu.au/hypervolume/). In particular, we rely on the second benchmark with dimensions ranging from 3 to 13, with spherical, random, discontinuous, and degenerate front datasets. Each dataset contains from 10 to 1000 points, depending on the number of dimensions. We performed 10 runs per dataset, each one with 20 fronts. The spherical points of the WFG dataset are not uniformly randomly chosen on a hypersphere's surface with d dimensions. To illustrate dimensions dependency, Fig. 13 shows a matrix of plots, each plot shows the points, plotted in 2-D, by choosing two coordinates and discarding the rest.

We also generated our own datasets, in such a way that the points were chosen uniformly at random on the hypersphere's surface. More precisely, we generated a random point in $[0, 1]^d$, using the `drand48` function, from `stdlib.h`. These points are uniformly random on the given space, but not necessarily on a hypersphere surface. We projected the points into the surface, by calculating the distance to the origin and dividing every coordinate by this value. Thus, obtaining points at distance 1. The resulting points can be seen in Fig. 14.

3) *Experimental Performance:* Figs. 15–19 show the results concerning the running time of our QHV algorithm; the WFG algorithm [18]; the FPRAS [36] with $\epsilon = 0.01$; the HV algorithm, which is an improved version of FLP (http://iridia.ulb.ac.be/∼manuel/hypervolume); and an optimized version for 4-D, HV4-D, provided by the authors [19]; We use logarithmic scales to cope with the performance gap among different algorithms.

In comparing the performance of the different algorithms, it is important to point out that the FPRAS returns an $\pm\epsilon$ approximation of the hypervolume, not an exact value. Moreover, the approximation may miss this interval with 25% probability. This algorithm is extremely sensitive to ϵ, changing it to 0.1 yields a 100 times speed-up, in practice. Therefore it is not meaningful to claim that the FPRAS is faster or slower in a given dataset, it depends on the precision that is required by the application. We choose a fixed reasonable ϵ to show how

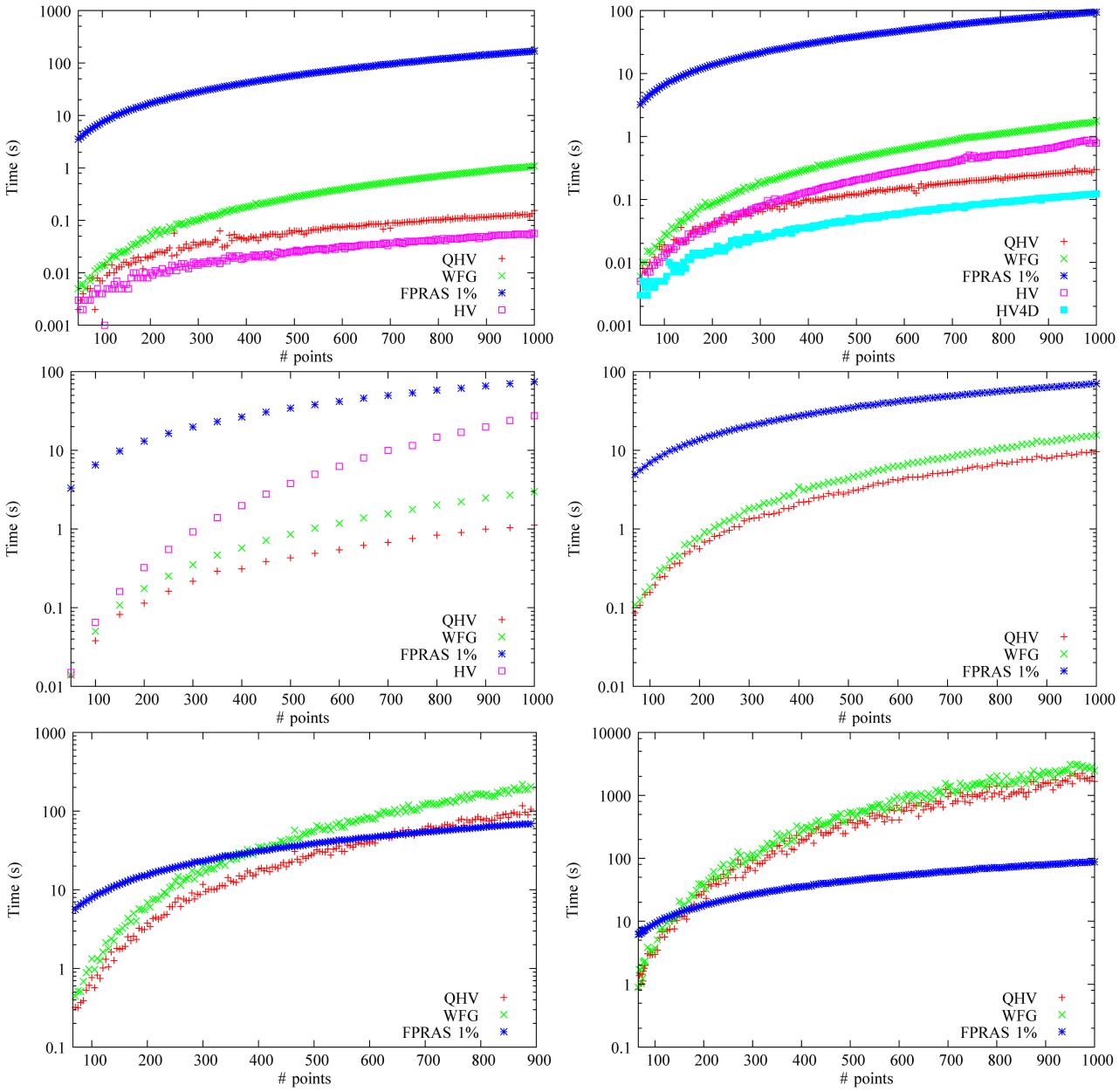

Fig. 15. (Left to right and top to bottom) Running time for 3-D, 4-D, 5-D, 7-D, 10-D, and 13-D spherical fronts.

the performance evolves. The real hypervolume decreases exponentially with d, therefore, the estimate should become inaccurate for higher dimensions. We inspected the resulting values and observed that estimates are actually reasonably accurate.

By far the worst performance of QHV occurs in the degenerated dataset. This is not an abnormal behavior of QHV. Instead, the remaining algorithms behave much better than usual, for that kind of dataset. Like QHV, the FPRAS algorithm also maintains a consistent performance. Notice that for 10-D the WFG algorithm is around 1000 times faster for this dataset than it is for the spherical dataset. Clearly, QHV and the FPRAS are ignoring some intrinsic property of the dataset.

We can observe that HV is the fastest algorithm in 3-D, but the performance degrades quickly. We omitted HV for higher dimensions, because of this slowdown. In 3-D, QHV is the second best algorithm. In 4-D, the fastest algorithm is the HV4D. QHV still obtains competitive performance, usually better than HV for a large number of points, except on the degenerated dataset. For higher dimensions, QHV is the fastest algorithm and the performance becomes similar to WFG for very high dimensions, for example 13-D (see Fig. 15). This is partially a consequence of the nonuniform dependencies on the dataset. To show this fact, we ran a 13-D test on our dataset (see Figure 16). The results still show a large gap between WFG and QHV, where QHV remains faster. In 13-D, we tested fewer points, because the algorithms became 100 times slower.

Note that at 13-D the FPRAS obtains much better performance than QHV and WFG. The FPRAS estimates are in fact fairly accurate.

In Table I, we show the memory peak requirements of the different algorithms. It can be observed that, up to 7-D, QHV requires less space than HV and WFG. The memory requirements of QHV increase with d, because we need to

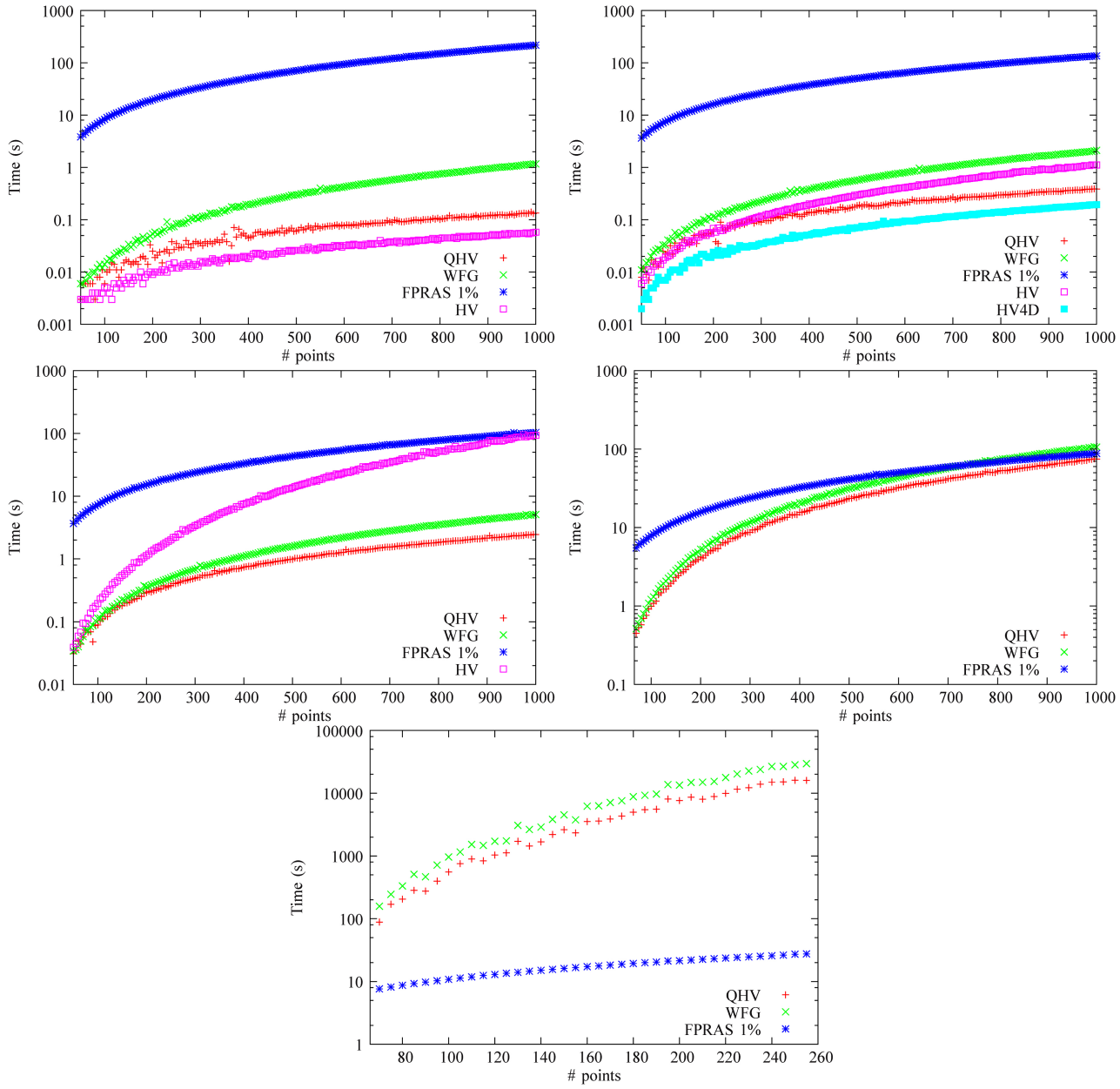

Fig. 16. (Left to right and top to bottom) Running time for 3-D, 4-D, 5-D, 7-D, and 13-D spherical fronts, from our dataset.

store a counter for each hyperoctant. As we mentioned in Section IV, this cost can be avoided by using a hash, which would also have an impact on the time performance. To avoid this effect, we chose not to implement the hash. In fact the space requirements of these algorithms are modest, for example, WFG loads all the point sets in a test into memory, to minimize this effect we reduced the tests so that they contain only one set of points. Although the memory peak increases for QHV the same happens to WFG, in 13-D QHV also requires less space than WFG.

V. Related Work

There has been a large amount of research focused on calculating hypervolumes. A fair analysis of QHV can only be established in this context. We analyze only algorithms that compute the hypervolume exactly; hence, we do not include HypE [20], because it computes the hypervolume partially, to determine a ranking.

The most naive algorithm consists in using the inclusion exclusion principle (IEX) [21]. Given two rectangles A and B, consider A as the rectangle with vertexes z and a, likewise B is the rectangle formed by vertexes z and b. The area of $A \cup B$ verifies the following relation $\mathrm{HV}(A \cup B) = \mathrm{HV}(A) + \mathrm{HV}(B) - \mathrm{HV}(A \cap B)$. Naturally, this relation can be generalized to include several rectangles. Alas, the resulting expression is exponential in the number of rectangles, an algorithm that uses the inclusion exclusion principle requires $\Theta(d2^n)$. This bound is not so bad when n is very small, especially because it has a relatively small dependency on d.

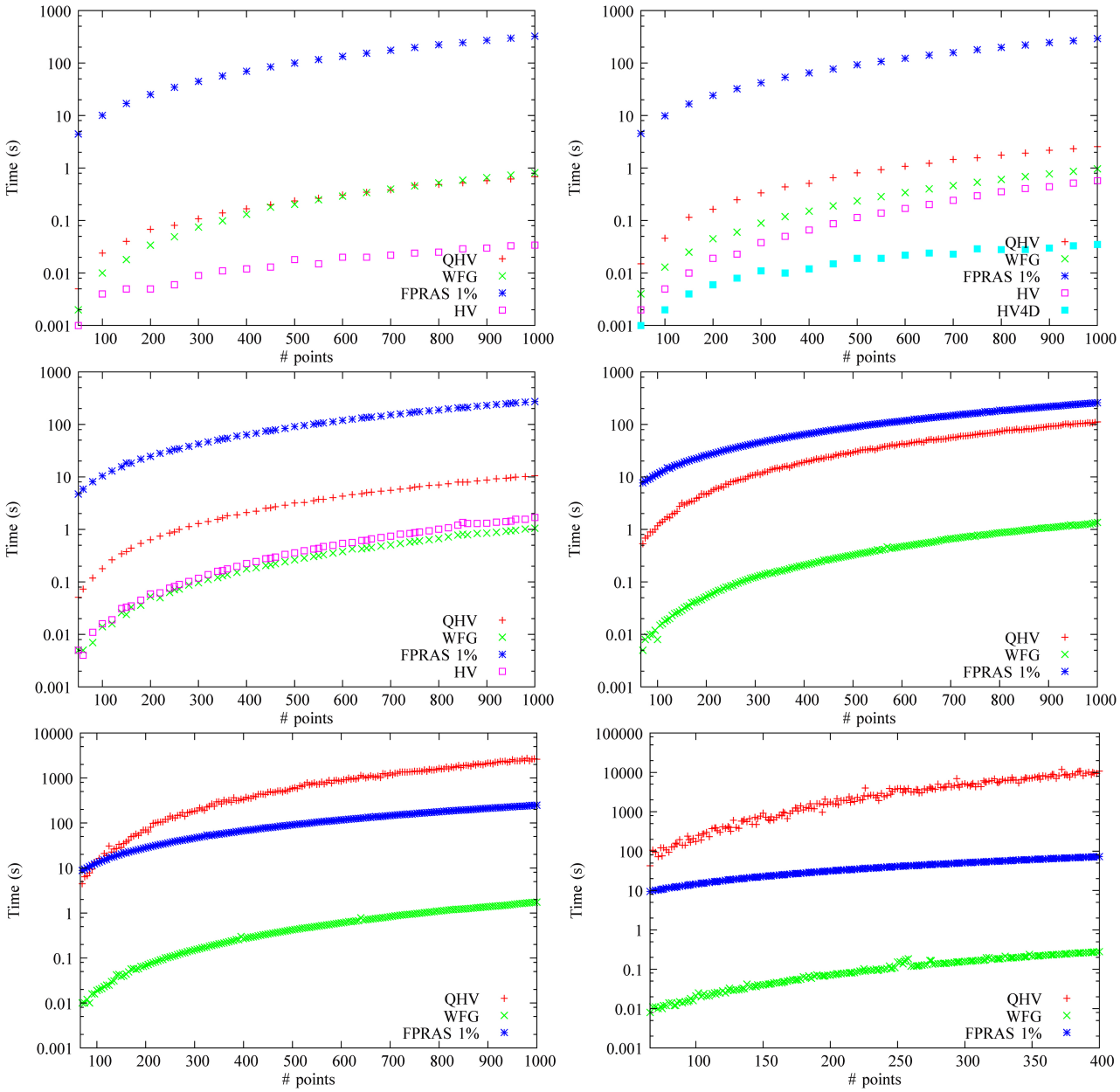

Fig. 17. (Left to right and top to bottom) Running time for 3-D, 4-D, 5-D, 7-D, 10-D, and 13-D degenerated fronts.

Divide and conquer techniques can be tracked through mathematics to centuries B.C. In computer science, the first such algorithm was mergesort by John von Neumann in 1945. In geometry, a general divide and conquer strategy was proposed by Bentley [22]. This approach is somehow different from the one in QHV. Bentley's divide and conquer divides a d-dimensional problem, with n points, into two d-dimensional problems on $n/2$ points. We show an illustration of this procedure in Fig. 20. In the example of the figure, we have two 2-D problems, the division line is chosen such that there are exactly $n/2$ points on each side. As these problems are solved, recursively, the resulting values must be combined in a marriage step. This implies solving a problem with $d-1$ dimensions. In this case, we simply determine the length of the dominated segment in the division line.

For the 2-D case, the resulting algorithm works in $O(n)$ time. Notice that we assume the points are given sorted, which would otherwise require $O(n \log n)$ preprocessing time. For a general dimension d, the Bentley's multidimensional divide and conquer technique [22] requires $O(n \log^{d-2} n)$ time. In particular, this method can be used to remove dominated points, for any number of dimensions d, still it cannot be directly applied to compute hypervolumes when $d > 2$.

This limitation led to the development of efficient algorithms for hypervolume, that use essentially different approaches. The area has been lively with increasingly effective algorithms being proposed regularly, from different research groups. Hence, discovering faster algorithms is becoming harder.

Instead of divide and conquer a common approach for hypervolume computation is a dimension sweep method.

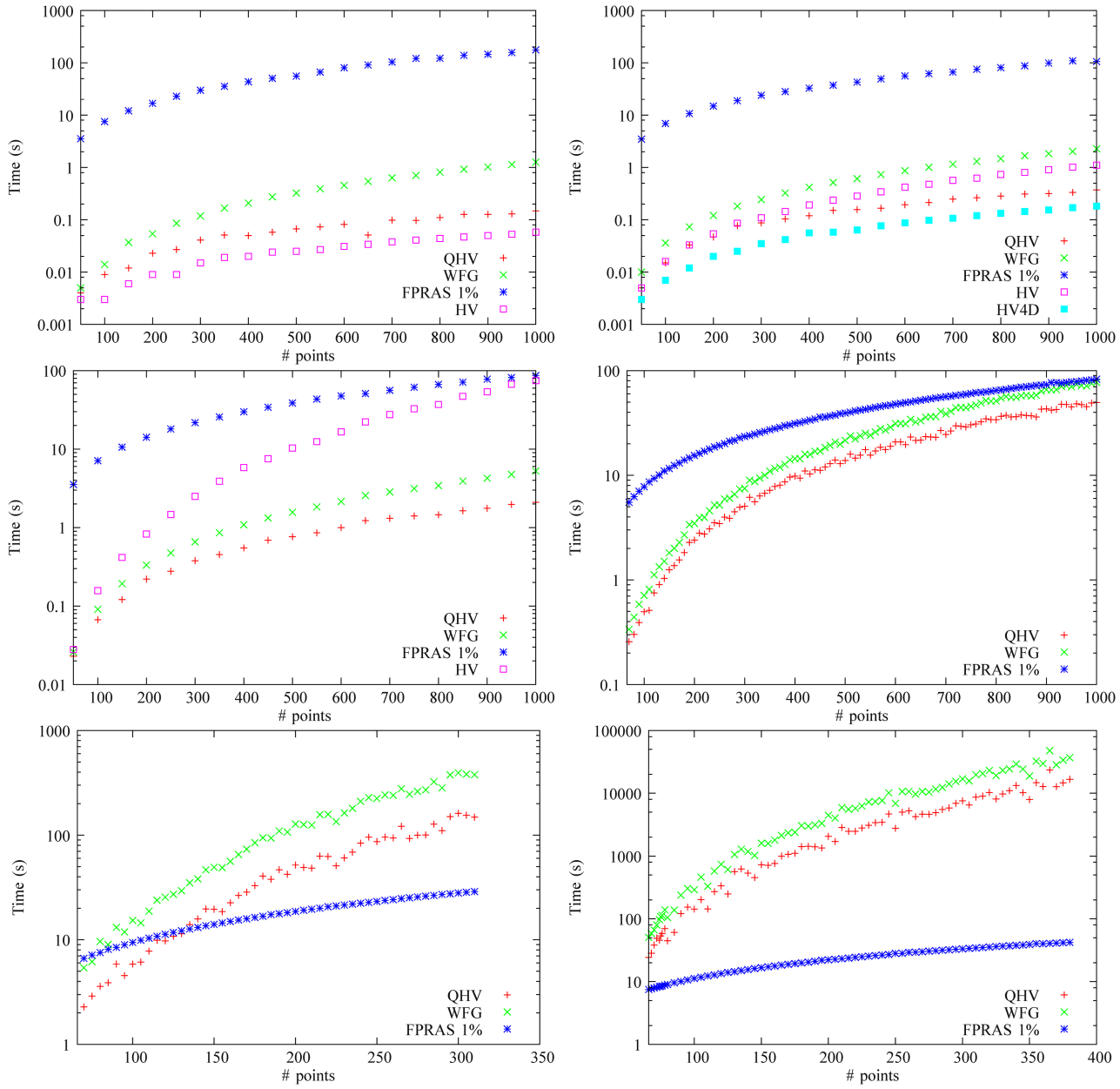

Fig. 18. (Left to right and top to bottom) Running time for 3-D, 4-D, 5-D, 7-D, 10-D, and 13-D random fronts.

The hypervolume by slicing objectives (HSO), proposed by While *et al.* [14], was the first such algorithm. At the time, the algorithm was shown to be more efficient than the LebMeasure algorithm [23]. The HSO algorithm works as follows. Consider that the line in Fig. 20 slides from right to left. Whenever the line is between two consecutive points the height of the segment is always the same. In general, between two consecutive points in the last dimension, there is a subproblem of dimension $d-1$ that is constant. We can multiply the $d-1$ hypervolume of this subproblem by the distance between the points. The $d-1$ problem is recursively solved with HSO. The overall time complexity becomes $O(n^{d-1})$, for $d \geq 2$. Notice that in 2-D we can solve the 1-D subproblems in $O(1)$ time, this happens because the 1-D problem is only the height of the dominated segment and the subproblems increase, monotonously, from right to left. This idea was pushed further by noticing that in 3-D the consecutive 2-D problems can also be updated incrementally. The resulting algorithm, by Paquete *et al.* [24], obtained $O(n \log n)$ time for 3-D problems. The generalized algorithm FLP [25], which also included new pruning improvements was the first to obtain $O(n^{d-2} \log n)$ time, for $d \geq 2$. This algorithm is sensitive to the order in which the dimensions are considered [26].

For higher dimensions, $d \geq 4$, the best theoretical bound was, until recently, the HOY algorithm [16], which is based on the fact that the hypervolume we are calculating is a special case of the measure of the union of hyper-rectangles, the d-dimensional version of Klee's measure [27], [28]. Overmars and Yap [29] obtained an $O((n+n^{d/2}) \log n)$ time algorithm for this problem. Hence, yielding an algorithm for hypervolume,

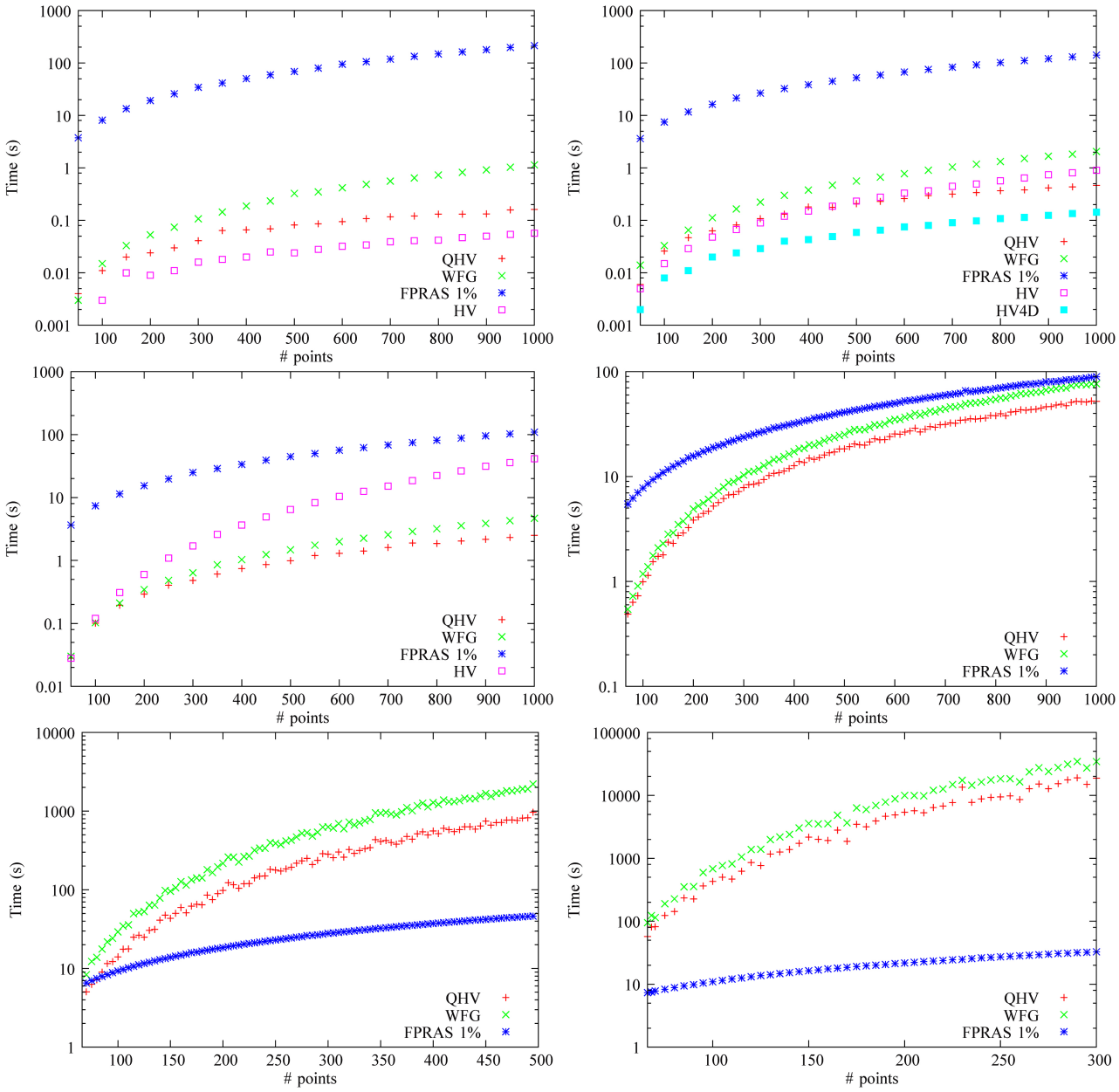

Fig. 19. (Left to right and top to bottom) Running time for 3-D, 4-D, 5-D, 7-D, 10-D, and 13-D discontinuous fronts.

although this was not immediately recognized in the hypervolume literature [25], [30]. Moreover, Beume and Rudolph [16] explored the fact that one of the hyper-rectangle vertexes is common to all rectangles and proposed the HOY algorithm, with $O(n \log n + n^{d/2})$ worst case time. This algorithm uses divide and conquer techniques. Initially, the space is divided into cells, when the cells are small enough they also use IEX. This division is done by dividing only one dimension at a time, similar to the multidimensional divide and conquer. Still the dimension that is divided need not be fixed. This division differs from QHV, that divides all dimensions simultaneously. The divided space is organized in an orthogonal partition tree. The last dimension is not included in the division and is processed with a line sweep. This algorithm is not as sensitive as FPL [13] to the order of the dimensions.

Recently, Guerreiro, Fonseca and Emmerich [19] gave the first improvement to HOY, the HV4-D algorithm obtains $O(n^2)$ time in 4-D. Even more recently, Yildiz and Suri [17] improved the general case to $O(n^{(d-1)/2} \log n)$ time. The GBox algorithm works by reducing the problem to a semidynamic weighted volume with $d-2$ dimensions. They design and make use of a structure to maintain the sum of ordered products.

Concerning these algorithms there is sometimes a large gap between theoretical results and actual practical performance. Moreover, theoretical solutions are sometimes cumbersome to implement, which might justify why we found no prototypes of HOY or GBox. The good theoretical complexity of HOY and GBox does not necessarily guarantee that they are the most efficient algorithms in practice. The average case complexity can be very different from the worst and best cases. For

example, we showed that QHV obtains $O(n \log n)$ time, in the best case, which is the lower bound [32] for hypervolume but this performance is virtually impossible.

Bradstreet *et al.* [31], [33], adapted HSO to compute the hypervolume incrementally, IHSO and IIHSO. The IHSO algorithm computes the contribution of a single point. Given a set of points $\mathcal{S}$ and a point p the exclusive hypervolume of p is $\text{HV}(\{p\} \cup \mathcal{S}) - \text{HV}(\mathcal{S})$. This is an important measure in some evolutionary algorithms [6]–[9]. The IIHSO algorithm computed the hypervolume by adding the exclusive contribution of each point, and removing it from the set. This approach is effective, and IIHSO became the fastest known algorithm on typical benchmark data with many objectives. The IHSO algorithm introduced a best-first queuing mechanism to efficiently determine the least-contributing point in a set. The worst case performance of IIHSO remains $O(n^{d-1})$.

Yang and Ding [34] have recently, and independently from us, proposed another pivot divide and conquer algorithm for hypervolumes [34, p. 5, Algorithm 1]. They obtained better worse case time $O((d/2)^n)$, but worse space $O(dn^2)$. They also selected a point to divide the space, but their division is very different. Contrary to QHV that makes $2^d - 2$ recursive calls, their algorithm uses always d recursive calls. Essentially the division into subproblems is different from QHV. They present only a theoretical analysis and there is no implementation of their algorithm.

Guerreiro [35] also proposed a divide and conquer algorithm for 3-D, the HVDC3-D, which obtains optimal $O(n \log n)$ time. They also use reference points to divide the set of points, but the resulting division is quite different.

The WFG algorithm [18] is inspired by the exclusive volume approach. The worst case performance is $O(2^n)$, which is better than the worst case of QHV. Still, just like QHV, the experimental running time is far from this bound [18]. This algorithm differs from IIHSO in the order in which the points are swept, and on how the contribution of each point is computed. The algorithm is recursive on the number of dimensions. It also uses the optimal $O(n \log n)$ algorithm for 3-D [25]. The experimental results on WFG establish it as the most efficient algorithm in high dimensions, in fact the performance is more resilient to increases in d. Section IV-C also shows a similar result for QHV. In fact, the exclusive hypervolume is used as a way to restrict the number of points involved in the computation. Hence, obtaining the same effect QHV obtains with division.

A slightly different approach to this problem consists in using approximation algorithms. Bringmann and Friedrich [36] have shown that computing the hypervolume is #P-hard, moreover, the same is true for most measures of unions of high-dimensional geometric objects. They presented an efficient fully polynomial-time randomized approximation scheme (FPRAS) for computing the volume of the unions of objects, provided it is possible to: 1) test whether a given point lies inside the object; 2) sample a point uniformly; and 3) calculate the volume of the object in polynomial time. Their algorithm computes an approximation of the hypervolume $\pm\sqrt{\epsilon}$ in $O(dn/\epsilon)$ time, although the result may miss this region with 25% probability. Bringmann and Friedrich [37], [38] have also proposed algorithms to compute the contribution of each individual point.

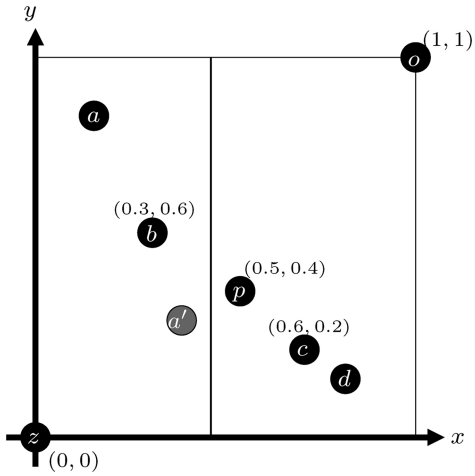

Fig. 20. Area of a set of 2-D points.

VI. Conclusion and Further Work

In this paper, we proposed QHV, a new algorithm for computing hypervolumes. We focused on performance, time, and space complexity. The QHV algorithm uses a divide and conquer strategy, which is different from the usual line sweep approach. The resulting algorithm is fairly simple and efficient. We analyzed QHV theoretically and experimentally.

A. Discussion

Algorithmic results have a tendency to be biased toward theory or practice. Theoretical results usually achieve significant worst case bounds, but the resulting implementations may be nonexistent or cumbersome. On the other hand good practical results are sometimes achieved, but lack a theoretical grounding. Most of the time practical algorithms are hard to analyze because their good performance resides in the average case and not in the worst case bound. Average case analysis are harder to obtain most of the time, as they may require sophisticated probability notions.

Algorithms to compute hypervolumes are no different. From the theoretical approach, the best bounds were held by Overmars and Yap [29], for twenty years. The HOY algorithm improved this result, by $\log n$ factor and became the best bound for the last three years. Guerreiro, Fonseca, and Emmerich [19] made the first improvement to HOY, for 4-D. Yildiz and Suri [17] obtained an $O(n^{(d-1)/2} \log n)$ time bound, with the GBox algorithm. This result was improved for $d \geq 7$, by Bringmann, which obtained $O(n^{(d+2)/3})$ running time by reducing Hypervolume to Klee's measure of a union of cubes [39]. We refer to this algorithm as RCube. Bringmann also established, assuming the Exponential Time Hypothesis, that the constant associated with d cannot be lowered indefinitely, i.e., there is no $n^o(d)$ time algorithm for Hypervolume.

Comparing against these results is hard because the worst case of QHV can be very bad. Still we argue that it is

legitimate to compare against the average case of QHV. Comparing the average case of QHV against the average case of GBox or HOY would be better, in fact ideal. Still, as far as we know, no such analysis exists. We also obtained a high probability bound for the average performance of QHV. The bound is the same, for points uniformly distributed on a hypersphere, or hyperplane, $O(dn^{1+\epsilon}\log^{d-2} n)$ time, where $\epsilon < 1$ can be as close to 0 as desired. This is a much stronger argument in favor of this comparison. In fact for all practical purposes, a high probability bound is comparable to a worst case bound.

In theory, compared with GBox, the QHV algorithm is significantly faster. Notice that GBox improved HOY by reducing the power of n from $d/2$ to $(d-1)/2$, i.e., the speed-up factor is $n^{1/2}$. The QHV algorithm always reduces the power to $1+\epsilon$. For the sake of argument, assume it is 1.01, in that case our speed-up factor to HOY is $n^{(d-2.02)/2}/(d\log^{d-3} n)$, to GBox it is $n^{(d-3.02)/2}/(d\log^{d-3} n)$, and to RCube $n^{(d-1.03)/3}/(d\log^{d-2} n)$. Therefore, in theory, QHV is the fastest algorithm for more than three dimension. In 2-D and 3-D, it does not achieve optimal performance, but for any other number of dimensions it obtains the best theoretical bounds, by far.

However, in practice, this result is less significant. We tested our prototype on several public datasets and compared against state-of-the-art software. Our prototype is the fastest existing software for 5-D or more, on most datasets. Still, compared to WFG, it is not that much faster. WFG is also effective for a large number of dimensions and in fact the performance of QHV and WFG becomes similar when the number of dimensions increases. This result shows that, even though there is no average case analysis of WFG, the actual bound is probably very good. Moreover, QHV still has plenty of room for improvement, namely in the degenerated dataset.

The FPRAS of Bringmann and Friedrich is, in several aspects, similar to QHV. Their contribution is again both theoretical and practical. On the one hand, their algorithm is only an approximation, meaning that the results of QHV are more precise, moreover, the FPRAS can miss an $\pm\sqrt{\epsilon}$ interval with 25% probability, but this probability can be further reduced. On the other hand they can tune the precision to obtain a faster algorithm. Including such an option in QHV would also be a significant improvement.

Overall, the QHV algorithm is a significant result. It obtains important theoretical bounds and the resulting prototype is very effective, both in terms of space and time.

B. Conclusion

We expect the QHV algorithm to have a significant impact on the future development of MOEAs, in that it makes comparing more objectives feasible.

QHV is still devoid of extra features. Designing a version of QHV that can compute exclusive hypervolumes is an important unattended task. Other closely related problems may also benefit from the pivot divide and conquer strategy of QHV, namely, computing empirical attainment functions [40].

ACKNOWLEDGMENTS

The authors would like to thank the Walking Fish Group for providing standard test sets of points and answering our questions regarding the WFG prototype, namely, some subtleties on the orientation of the dominance. They are also deeply grateful to A. Guerreiro and C. Fonseca for showing them the hypervolume problem [35] and providing prototypes and positive feedback. They are also grateful to T. Friedrich for his enthusiasm regarding the initial results and for providing the software and bibliography.

APPENDIX A
WORST CASE DETAILS

Geometric series of the worst case

$$T(n) = \sum_{i=0}^{n} d(n-i)2^{di} \tag{7}$$

$$\leq \sum_{i=0}^{n} dn2^{di} \tag{8}$$

$$= dn\sum_{i=0}^{n} 2^{di} \tag{9}$$

$$= dn\Theta(2^{nd}). \tag{10}$$

APPENDIX B
HIGH PROBABILITY PROOF

Proof: Assume $\epsilon, \epsilon' > 0$ are small positive numbers and that $r = o + \epsilon$ and $f = f_o + \epsilon'$.

The proof uses the approach in Probability and Computing [15, exercise 4.20]. Our main novelty is Lemma 3. The proof is the consequence of the following properties.

1) The number of regular nodes in a branch of the recursion tree h is upper bounded by $\log n/(-\log f)$.
2) The probability that a branch of the recursion tree contains more than $a\log n/(-\log f)$ nodes is at most $1/n^{r+1}$, for a certain constant a.
3) The probability that **no** branch of the recursion tree contains more than $a\log n/(-\log f)$ nodes is at least $1-1/n$.
4) Bound $S(n)$ with at least $1-1/n$ probability, by summing the branches, and fixing constants.

To prove point 1, notice that, for a given $n \geq n_0$, a subproblem of a regular node contains fn or less points. Therefore, the number of regular nodes h along a branch must respect the relation $f^h n \leq n_0$. Notice that we can throw away the nodes that contain problems smaller than n_0, by solving them with any algorithm. Therefore, $h \leq \log(n/n_0)/(-\log f)$. Since $n_0 \geq 1$, we have just proven point 1.

To prove point 2, we use Chernoff bounds. We associate with each regular node a geometric random variable Z_i. These variables count the number of nodes between a regular node and its regular ancestor, including the initial regular node and moving up. The leaves are necessarily regular. Note that the sum of these variables yields a random variable $Z = \sum_{i=1}^{h} Z_i$

that bounds the size of a random branch in the tree. We apply a Chernoff bound to Z.

Recall that the moment generator function of a geometric random variable is the following:

$$E[e^{tZ_i}] = \frac{p}{e^{-t} - (1-p)}, \text{ for } 1 - p < e^{-t} < 1.$$

We have $p = 1/2$ and use $x = e^{-t}$. Therefore, the function simplifies to

$$E[x^{-Z_i}] = \frac{1}{2x-1}, \text{ for } 1/2 < x < 1.$$

Since the variables are independent of each other, we can multiply the generating functions to obtain the generating function of Z

$$E[x^{-Z}] = \frac{1}{(2x-1)^h}, \text{ for } 1/2 < x < 1.$$

Hence, the Chernoff bound for the length of a branch in the recursion tree becomes

$$Pr(Z \geq ah) \leq \min_{1/2<x<1} \left(\frac{x^{ah}}{(2x-1)^h} \right).$$

We choose $x = (1 - 1/a)$ for some $a > 2$. With this choice we have that $x^a \leq 1/e$ and obtain that

$$Pr(Z \geq ah) \leq \left(\frac{a}{e(a-2)} \right)^h.$$

To obtain $Pr(Z \geq ah) \leq 1/n^{1+r}$, we require the hypothesis, which states that $ef^{r+1} > 1$. In this case, we choose

$$a \geq \frac{2}{1 - 1/(ef^{1+r})}.$$

This value is constant regarding n. This concluded point 2.

To prove 3, we use a union bound. Recall from Lemma 3 that there are at most n^r leaves in the recursion tree. Therefore, there are at most n^r branches in the tree. We unite, for every branch, the probability that the branch is larger than ah. Let $L(n)$ be a random variable that returns the size of the longest branch in the recursion tree. We obtain that

$$Pr(L(n) \geq ah) \leq n^r/n^{r+1} = 1/n.$$

We can now negate this event and obtain the following:

$$Pr(L(n) < ah) \geq 1 - 1/n/$$

This concludes point 3.

To prove 4, we bound $S(n)$. Notice that if we sum the size of all the branches, with overlap, we obtain a value larger than $S(n)$. Since we have bounded the probability that any branch is larger than ah, we can conclude that, with the same probability, $S(n) \leq ah \cdot n^r$, because there are at most n^r branches. We now use Lemma 1 to bound $T(n)$. Hence

$$Pr\left(T(n) < ahn^r \cdot O(d \log^{d-2} n)\right) \geq 1 - \frac{1}{n}.$$

Recall that there is a $log(n/n_0)$ inside of h, to make this and the a factor disappear assume that we repeat the above proof with $\epsilon/2$, instead of ϵ. We use $\epsilon/2$ everywhere except on the expression that lower bounds a, for which we still use ϵ. This gives us a bound that is smaller than the one in the theorem by $O(n^{\epsilon/2})$. We use this extra slack to bound $a \log(n/n_0) = O(n^{\epsilon/2})$. Therefore, we conclude that

$$Pr\left(T(n) < \frac{n^r}{-\log f} \cdot O(d \log^{d-2} n)\right) \geq 1 - \frac{1}{n}$$

for any n larger than some n_0. ■

REFERENCES

[1] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. Van Briesen, and N. S. Glance, "Cost-effective outbreak detection in networks," in *Proc. KDD*, Aug. 2007, pp. 420–429.

[2] B. Schwartz, *The Paradox of Choice: Why More Is Less* (P. S. Series). New York, NY, USA: Harper Collins, 2005.

[3] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms* (Wiley-Interscience Series in Systems and Optimization). New York, NY, USA: Wiley, 2009.

[4] A. Zhou, B. Qu, H. Li, S. Zhao, P. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state-of-the-art," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 32–49, 2011.

[5] C. Goh, Y. Ong, and K. Tan, *Multi-Objective Memetic Algorithms*, vol. 171. New York, NY, USA: Springer Verlag, 2009.

[6] S. Huband, P. Hingston, L. While, and L. Barone, "An evolution strategy with probabilistic mutation for multi-objective optimization," in *Proc. IEEE Congr. Evol. Comput.*, Dec. 2003, pp. 2284–2291.

[7] J. Knowles, D. Corne, and M. Fleischer, "Bounded archiving using the Lebesgue measure," in *Proc. IEEE Congr. Evol. Comput.*, Dec. 2003, pp. 2490–2497.

[8] N. Beume, M. Emmerich, and B. Noujoks, "An EMO algorithm using the hypervolume measure as selection criterion," in *Proc. Evol. Multi-Objective Optimiz.*, 2005, pp. 62–76.

[9] E. Zitzler and S. Knzli, "Indicator-based selection in multiobjective search," in *Proc. PPSN VIII*, vol. 3242. 2004, pp. 832–842.

[10] C. A. R. Hoare, "Algorithm 64: Quicksort," *Commun. ACM*, vol. 4, no. 7, p. 321, Jul. 1961.

[11] P. Briggs and L. Torczon, "An efficient representation for sparse sets," *ACM Lett. Program. Lang. Syst.*, vol. 2, no. 1–4, pp. 59–69, 1993.

[12] J. Bentley, *Programming Pearls*. Reading, MA, USA: Addison-Wesley Professional, 2000.

[13] A. Aho and J. Hopcroft, *Design and Analysis of Computer Algorithms*. Delhi, India: Pearson Education, 1974.

[14] L. While, P. Hingston, L. Barone, and S. Huband, "A faster algorithm for calculating hypervolume," *IEEE Trans. Evol. Comput.*, vol. 10, no. 1, pp. 29–38, Feb. 2006.

[15] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge, U.K.: Cambridge Univ., 2005.

[16] N. Beume and G. Rudolph, "S-metric calculation by considering dominated hypervolume as Klee's measure problem," *Evol. Comput.*, vol. 17, no. 4, pp. 477–492, 2009.

[17] H. Yildiz and S. Suri, "On Klee's measure problem for grounded boxes," in *Proc. Symp. Comput. Geometry*, 2012, pp. 111–120.

[18] L. Bradstreet, L. While, and L. Barone, "A fast way of calculating exact hypervolumes," *IEEE Trans. Evol. Comput.*, vol. 16, no. 1, pp. 86–95, Feb. 2012.

[19] A. P. Guerreiro, C. M. Fonseca, and M. T. M. Emmerich, "A fast dimension-sweep algorithm for the hypervolume indicator in four dimensions," in *Proc. CCCG*, Aug. 2012, pp. 77–82.

[20] J. Bader and E. Zitzler, "Hype: An algorithm for fast hypervolume-based many-objective optimization," *Evol. Comput.*, vol. 19, no. 1, pp. 45–76, 2011.

[21] J. Wu and S. Azarm, "Metrics for quality assessment of a multiobjective design optimization solution set." *J. Mech. Design*, vol. 123, no. 1, pp. 18–25, 2001.

[22] J. L. Bentley, "Multidimensional divide-and-conquer," *Commun. ACM*, vol. 23, pp. 214–229, Apr. 1980.

[23] M. Fleischer, "The measure of pareto optimal applications to multiobjective metaheuristics," in *Proc. EMO*, 2003, pp. 519–533.

[24] C. Fonseca, L. Paquete, and M. Lopez-Ibanez, "An optimal algorithm for a special case of Klees measure problem in three dimensions," Centre for Intelligent Systems, University of Algarve, Faro, Potugal, Tech. Rep. CSI-RT-I-01/2006, 2006.

[25] C. Fonseca, L. Paquete, and M. Lopez-Ibanez, "An improved dimension-sweep algorithm for the hypervolume indicator," in *Proc. IEEE Congr. Evol. Comput.*, Jul. 2006, pp. 1157–1163.

[26] L. While, L. Bradstreet, L. Barone, and P. Hingston, "Heuristics for optimizing the calculation of hypervolume for multi-objective optimization problems," in *Proc. IEEE Congr. Evol. Comput.*, vol. 3. Sep. 2005, pp. 2225–2232.
[27] V. Klee, "Can the measure of $\bigcup_1^n [a_i, b_i]$ be computed in less than $o(nlogn)$ steps?" *Amer. Math. Monthly*, vol. 84, no. 4, pp. 284–285, Apr. 1977.
[28] J. L. Bentley, "Algorithms for Klee's rectangle problems," Comput. Sci. Dept., Carnegie Mellon Univ., 1977.
[29] M. Overmars and C.-K. Yap, "New upper bounds in Klee's measure problem," in *Proc. 29th Annu. Symp. Found. Comput. Sci.*, Oct. 1988, pp. 550–556.
[30] N. Beume, "Hypervolumen-basierte selektion in einem evolutionären algorithmus zur mehrzieloptimierung," 2006.
[31] L. Bradstreet, L. While, and L. Barone, "A fast many-objective hypervolume algorithm using iterated incremental calculations," in *Proc. IEEE Congr. Evol. Comput.*, Jul. 2010, pp. 1–8.
[32] N. Beume, C. Fonseca, M. López-Ibáñez, L. Paquete, and J. Vahrenhold, "On the complexity of computing the hypervolume indicator," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 1075–1082, Oct. 2009.
[33] L. Bradstreet, L. While, and L. Barone, "A fast incremental hypervolume algorithm," *IEEE Trans. Evol. Comput.*, vol. 12, no. 6, pp. 714–723, Dec. 2008.
[34] Q. Yang and S. Ding, "Novel algorithm to calculate hypervolume indicator of pareto approximation set," in *Proc. Adv. Intell. Comput. Theories Applicat.*, vol. 2. 2007, pp. 235–244.
[35] A. Guerreiro, "Efficient algorithms for the assessment of stochastic multiobjective optimizers," Masters thesis, Dept. Comput. Sci. Eng., IST, Tech. Univ. Lisbon, Lisbon, Portugal, 2011.
[36] K. Bringmann and T. Friedrich, "Approximating the volume of unions and intersections of high-dimensional geometric objects," *Comput. Geometry*, vol. 43, no. 6, pp. 601–610, 2010.
[37] K. Bringmann and T. Friedrich, "An efficient algorithm for computing hypervolume contributions," *Evol. Comput.*, vol. 18, no. 3, pp. 383–402, 2010.
[38] K. Bringmann and T. Friedrich, "Approximating the least hypervolume contributor: NP-hard in general, but fast in practice," in *Proc. EMO*, LNCS 5467. 2009, pp. 6–20.
[39] K. Bringmann, "Bringing order to special cases of Klee's measure problem," *Comput. Res. Repository*, 2013, arXiv:1301.7154
[40] V. Grunert da Fonseca, C. Fonseca, and A. Hall, "Inferential performance assessment of stochastic optimisers and the attainment function," in *Proc. Evol. Multi-Criterion Optimiz.*, 2001, LNCS 1993, pp. 213–225.

Luís M. S. Russo received the Ph.D. degree from Instituto Superior Técnico (IST), Universidade Técnica de Lisboa, Lisbon, Portugal, in 2007.

He is currently an Assistant Professor with IST. His current research interests include algorithms and data structures for string processing and optimization.

Alexandre P. Francisco received the Ph.D. degree in computer science and engineering from Instituto Superior Técnico (IST), Universidade Técnica de Lisboa, Lisbon, Portugal.

He is currently an Assistant Professor with IST. His current research interests include the design and analysis of data structures and algorithms, with applications on network mining and large data processing.