Incremental hypervolume calculation in d dimensions

Kyrylo Yefimenko
kyrylo.yefimenko@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

December 2019

Abstract

In this paper the problem of the calculation of hypervolume for a set of points for any dimension d and its incremental variant are described. The IQHVII algorithm is presented for the incremental case and is based on the QHV-II algorithm developed in [1] for non incremental case. This new algorithm uses a datastructure of a form of a tree called quadtree to store the problem's nondominated points. This tree allows the calculation of the hypervolume to, possibly, be faster as there are no dominated points. Additionally, the IQHVII algorithm is analysed theoretically and some experiments are performed and presented.

Keywords: Hypervolume, Incremental Hypervolume Calculation, QHV, QHV-II, Quadtree, IQHVII

1. Introduction

Multi-objective optimization problems are of high importance in a vast range of fields including engineering, economics and management, and sciences such as chemistry, biology and medicine. They consist of optimizing several objectives at the same time which are usually conflicting. More precisely, the problem consists of finding the whole set of Pareto optimal solutions in a feasible region in the decision space D mapped to a feasible region in the objective space $O \subset \mathbb{R}^d$ by d objective functions which can be represented as $f : D \longrightarrow O$ with $f(x) = (f_1(x), \ldots, f_d(x))$.

For a nontrivial multi-objective optimization problem it is not an easy task to find the set of Pareto optimal solutions, and so, a considerable amount of time is used computing it. Therefore, it is important to choose an efficient algorithm that can compute the hypervolumes generated by different sets of points and compare them quickly, thus deciding which set is a better solution to the problem. The hypervolumes can be calculated in different ways and, in this paper, we will use the hypervolume indicator method for those computations.

The hypervolume of a set of solutions measures the size of the portion of objective space that is dominated by those solutions collectively. Hypervolume encapsulates in a single unary value a measure of the spread of the solutions along the Pareto front, as well as the closeness of the solutions to the Pareto-optimal front, making it an adequate metric for the problem. It also incorporates many mathematical properties favourable for use in multi-objective optimization problems. The exact calculation of the hypervolume is shown to be #P-hard[1] and its approximation NP-hard in [2].

There are several non trivial algorithms for exact and approximated hypervolume calculations and the main focus of this paper is to study the QHV-II algorithm's incremental variant with the main idea of calculating the final hypervolume without recalculating, fully or partially, the hypervolume of the initial state.

2. Background

To tackle the problem of calculating the incremental hypervolume correctly and rigorously, it is necessary to have some fundamental definitions present.

2.1. Notation

In this paper, points are represented by lower case letters and a d-dimensional point x is represented as $x = (x_1, \ldots, x_d)$. A general point shall be represented by p. Sets are represented by capital letters and a general set will be represented by S. We shall also assume that n is the number of points and d is the number of objectives. The hypervolume of a box A is expressed as $H(A)$.

2.2. Definitions

Multi-objective optimization problems are solved by determining all efficient solutions in the decision space and the corresponding nondominated points in the objective space such that none of the objectives can be improved in value without degrading some of the other objective values. The concepts necessary to understand the problem are defined

[1]#P is the analog of NP for counting problems.

below.

Definition 1 (Decision Space). *The decision space D of a multi-objective optimization problem is the space containing the solutions to the problem.*

Definition 2 (Objective Space). *The objective space O of a multi-objective optimization problem is the space containing the evaluation of the solutions to the problem.*

Definition 3 (Feasible Region). *The feasible region of a decision space D is the set of all possible points of a multi-objective optimization problem that satisfy the problem's constraints, potentially including inequalities, equalities, and integer constraints. The feasible region of the objective space O is the image of the feasible region of a decision space given by some function $f : D \longrightarrow O$.*

To find the solutions to the multi-objective optimization problem the concepts of dominance and strong dominance are used. These definitions are also essential in the hypervolume calculation problem.

Definition 4 (Dominance [3]). *Let $x, y \in \mathbb{R}^d$ denote two points. Then, x dominates y if and only if $x \geq y$ and $x \neq y$ (i.e., $x_j \geqslant y_j$ for all $j = 1, \ldots, d$ and $x_j > y_j$ for at least one j). This type of dominance is also called weak dominance.*

Definition 5 (Strong Dominance [3]). *Let $x, y \in \mathbb{R}^d$ denote two points. Then, x strongly dominates y if and only if $x > y$ (i.e., $x_j > y_j$ for all $j = 1, \ldots, d$).*

To find the best solution to a multi-objective problem we must compare several possible solutions known as Pareto fronts and find the Pareto optimal front. The Pareto optimal front of an optimization problem is a set of points such that there is no other possible set that would improve the solution. More precisely, for the hypervolume calculation problem we have that the Pareto optimal front is a set of points of the objective space O which are not dominated by any other point. For example, if we consider the point p in Figure 1 we notice that the points b and c are dominated by p because $b_x \leqslant p_x \wedge b_y \leqslant p_y$ and $c_x \leqslant p_x \wedge c_y \leqslant p_y$, respectively. On the other hand, the points a, d and e are not dominated by p because $a_y > p_y$, $d_x > p_x$ and $e_x > p_x$, respectively. It is worth noting that as $a_y > p_y$ and $a_x < p_x$, no point dominates other point and so they are said to be incomparable. Thus, the Pareto dominance relation does not establish a total order. In this case, the Pareto optimal front is the set $\{a, p, d, e\}$, as no other combination of points will have a larger hypervolume and therefore improve the solution.

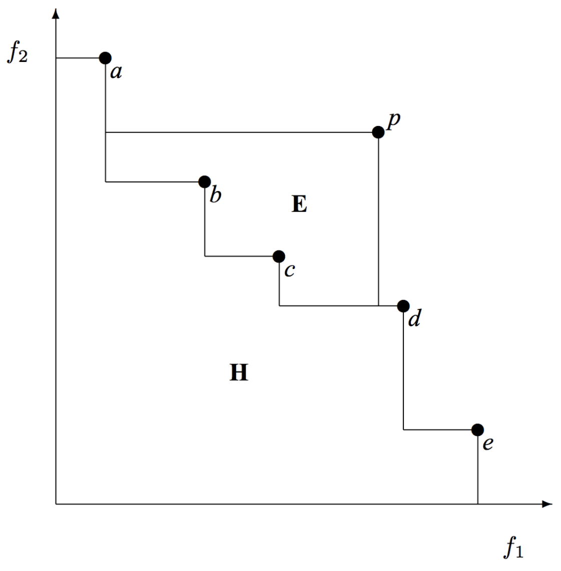

Figure 1: Example of a possible setup in 2 dimensions.

For a nontrivial multi-objective optimization problem no single solution exists that simultaneously optimizes each objective. In that case, the objective functions are said to be conflicting, and there exists a (possibly infinite) number of Pareto optimal solutions.

Definition 6 (Efficient Point [3]). *Consider a decision space D, an objective space O and a function $f : D \longrightarrow O$. Then a point $p \in D$ is said to be efficient if there is no $q \in D$ such that $f(q)$ dominates $f(p)$.*

It is worth noting that a nondominated point p in the objective space is always an image of an efficient solution to the problem in the decision space, thus is part of the Pareto optimal solution.

To compare several possible solutions of a multi-objective optimization problem the hypervolume indicator can be used. It is a set measure that computes the hypervolume of the dominated portion of the objective space bounded by a reference point. The calculated volume later helps in comparing Pareto fronts to decide which one is an optimal solution to the problem. Based on all the definitions above we can now formally define this metric [4]:

Definition 7 (Hypervolume Indicator). *The hypervolume indicator $H(S)$ of a set $S \subset \mathbb{R}^d$ can be defined as the hypervolume of the space that is dominated by the set S and is bounded by the reference point $r \in \mathbb{R}^d$:*

$$H(S) = \lambda\left(\bigcup_{x \in S} [r, x]\right), \tag{1}$$

where $\lambda(S)$ is the Lebesgue measure of a set S and $[r,x]$ is the d-dimensional hyperoctant consisting of all points that are weakly dominated by the point x but not weakly dominated by the reference point and can be represented as $[r,x] = \{y \in \mathbb{R}^d : r \leq y \wedge y \leq x\}$.

For example, if we consider the Figure 1 with $S = \{a,b,c,d,e\} \subset \mathbb{R}^2$ and $r = (0,0)$, then the area represented by H is the hypervolume of S, that is $H(S)$. For sets, we say that the set A is a better solution than the set B if $H(A) > H(B)$.

The hypervolume contribution of a point is based on hypervolume indicator and is defined as follows:

Definition 8 (Hypervolume Contribution)**.** *The hypervolume contribution, also know as exclusive hypervolume contribution, of a point $p \in \mathbb{R}^d$ to a set $S \subset \mathbb{R}^d$ is:*

$$H(p,S) = H(S \cup \{p\}) - H(S \backslash \{p\}). \qquad (2)$$

In Figure 1, the area denoted by E is the hypervolume contribution of p to the set $S = \{a,b,c,d,e\}$.

Definition 9 (Inclusive Hypervolume)**.** *The inclusive hypervolume of a point $p \in \mathbb{R}^d$ is the hypervolume defined by p and the reference point, ignoring all other points.*

2.3. Relevant Hypervolume Problems

There is a quite rich list of hypervolume related problems beyond the classic hypervolume calculation and incremental case problems. Several of them [4] are referenced in this section and the goal will be to study the UpdateHypervolume problem which is the incremental case. For all the problems below, S represents a set of points which are usually nondominated points, although this characteristic is not mandatory. If $q \in S$ is dominated, it has a null contribution to S, but if it is dominated by a single point $p \in S$, then the contribution of p to S will be lower than if $q \notin S$. Thus, this property is relevant for some problems.

Problem 1 (Hypervolume)**.** *Given a set of points $S \subset \mathbb{R}^d$ and a reference point $r \in \mathbb{R}^d$, compute the hypervolume indicator of S, that is, $H(S)$.*

Problem 2 (OneContribution)**.** *Given a set of points $S \subset \mathbb{R}^d$, a reference point $r \in \mathbb{R}^d$ and a point $p \in \mathbb{R}^d$, compute the hypervolume contribution of p to S, that is, $H(p,S)$.*

Problem 3 (AllContributions)**.** *Given a set of points $S \subset \mathbb{R}^d$ and a reference point $r \in \mathbb{R}^d$, compute the hypervolume contributions $H(p,S)$ for all points $p \in S$ to S.*

Problem 4 (LeastContributor)**.** *Given a set of points $S \subset \mathbb{R}^d$ and a reference point $r \in \mathbb{R}^d$, find a point $p \in S$ with minimal hypervolume contribution to S.*

Problem 5 (UpdateHypervolume)**.** *Given a set of points $S \subset \mathbb{R}^d$, a reference point $r \in \mathbb{R}^d$, the value of $H(S)$ and a point $p \in \mathbb{R}^d$, compute:*

- ***Incremental:*** *$H(S \cup \{p\}) = H(S) + H(p,S)$, if $p \notin S$.*
- ***Decremental:*** *$H(S \setminus \{p\}) = H(S) - H(p,S)$, if $p \in S$.*

Problem 6 (UpdateAllContributions)**.** *Given a set of points $S \subset \mathbb{R}^d$, a reference point $r \in \mathbb{R}^d$, the value of $H(q,S)$ $\forall q \in S$ and a point $p \in \mathbb{R}^d$, compute:*

- ***Incremental:*** *$H(q, S \cup \{p\}) = H(q,S) - H(p,q,S)$ for all $q \in S$ and also $H(p,S)$, where $p \notin S$.*
- ***Decremental:*** *$H(q, S \setminus \{p\}) = H(q,S) + H(p,q,S)$, where $p \in S$.*

All the problems are related in some way and most of them are solvable by solving one or more of the other problems. Thus, the state-of-the-art algorithms frequently exploit these relations. For instance, the UpdateHypervolume problem can be solved by computing either Hypervolume for the set $S \cup \{p\}$ or OneContribution of p to S. On the other hand, the Hypervolume problem can be solved by computing a sequence of UpdateHypervolume, recreating the set S by adding its points one by one.

3. QHV

Quick Hypervolume (QHV) developed in [5] is an exact hypervolume calculation algorithm for a d-dimensional space and is a major contribution to the state-of-the-art. It is a divide and conquer algorithm, based on QuickSort, over a set of points S and can be defined in 3 main steps:

1. Select a pivot point. The point is processed and excluded from the following recursive calls.
2. Divide the space using axis parallel hyperplanes according to the pivot and classify points and their projections into the new space regions. This step generates 2^d subproblems.
3. Recursively solve each of the subproblems and add up the hypervolumes.

More precisely, the idea of QHV is to, first, select a pivot p such that p is not dominated by any other point. This could be done by comparing all points which would yield $\mathcal{O}(n^2)$-time. A better way is to compute the inclusive hypervolume for each point

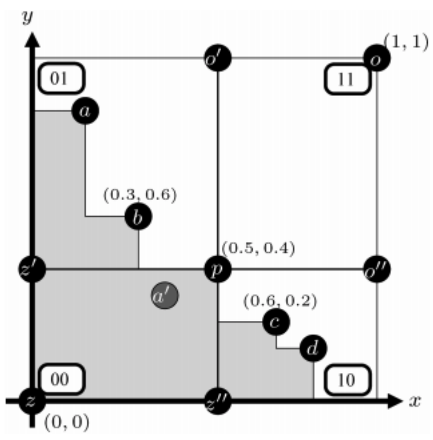

Figure 2: Dividing the 2-dimensional region at pivot p. The new regions are labeled by binary numbers. The point z is the reference point and the point o is there just for the visualization and is not part of the set of points the hypervolume is calculated for.

and pick the point with largest hypervolume as a pivot in $\mathcal{O}(n)$-time. The next step is to divide the space at pivot p. The hyperoctants generated by this step are labeled by binary strings of length d. If a hyperoctant is labeled by a string s then if $s_i = 0$, it contains all points $q \in \mathbb{R}^d$ such that $q_i < p_i$ and if $s_i = 1$ then it contains all points $q \in \mathbb{R}^d$ such that $q_i > p_i$. Considering the Figure 2, when dividing the space at p, the 11 quadrant is always empty and can be ignored in recursive calls. Also, the 00 quadrant is completely full and equals to the inclusive hypervolume of p leaving us with 01 and 10 quadrants for the conquer procedure. These facts can be generalized for a d-dimensional space, that is, the hyperoctant 1^d is always empty and 0^d is always equal to the inclusive hypervolume of $p \in \mathbb{R}^d$ and all other hyperoctants are considered for the recursive calls. Finally, the algorithm uses the conquer technique and, recursively, applies the steps to all hyperoctants but 0^d and 1^d. QHV also includes a subroutine to discard dominated points from each generated hyperoctant which makes the recursive computations faster.

4. QHV-II

QHV has been improved in [1] by changing the way the space is divided into subproblems which resulted in the QHV-II algorithm. Instead of splitting the region at the point p in 2^d hyperoctants, QHV-II splits it in d hyperoctants, thus generating less subproblems, and is done the following way:

- H_1 is defined by points s such that $s_1 \geqslant p_1$.
- ...
- H_j is defined by points s such that $s_l \leqslant p_l$, $\forall l = 1, \ldots, j-1 \wedge s_j \geqslant p_j$.

The QHV-II algorithm has been implemented in Kotlin which runs on Java Virtual Machine and the implementation details and complexity analysis follow. First, the implemented algorithm does not remove the dominated points unless they are dominated by the pivot in which case they are automatically removed in $\mathcal{O}(d)$-time when the set of points is iterated over while creating subproblems' hyperoctants. Second, the implementation does not use any other strategy to compute the hypervolume when some base case of number of points in a problem is reached, i.e., QHV-II is applied at every stage of the hypervolume computation, be that for a problem of 2 points or one of $n > 2$ points. The pseudocode for the algorithm is presented as Algorithm 1.

Algorithm 1 QHV-II

Input: Initial set of points S and the reference point r which is **0** by default.

Output: Exact hypervolume of set S.

```
1:  if $S$ is empty then
2:      return 0
3:  end if
4:  if $S.size$ is 1 then
5:      $p \leftarrow$ The only point of S.
6:      return $H(\{p\})$
7:  end if
8:  $p \leftarrow S$'s pivot
9:  $hypervolume \leftarrow H(\{p\})$
10: Split space in $d$ hyperoctants at point $p$ generating the set $\{H_1, \ldots, H_d\}$ with the respective reference points $\{r_1, \ldots, r_d\}$
11: for all $H_l \in \{H_1, \ldots, H_d\}$ do
12:     Construct set $S_l$ containing the points dominating $r^l$ and if necessary projected onto $H_l$
13:     $hypervolume \leftarrow hypervolume +$ QHV-II$(S_l, r_l)$
14: end for
15: return $hypervolume$
```

4.1. Theoretical Best Case Analysis

For the best case, it is necessary that the points are uniformly distributed by the subproblems, i.e., each subproblem is generated with n/d points, where n is the number of points in the current problem. The algorithm behaves as follows. It receives a hyperoctant, calculates the hypervolume generated by the pivot in $\mathcal{O}(nd)$-time and breaks the problem in d subproblems with n/d points each which is also done in $\mathcal{O}(nd)$-time. Thus, the recurrence is as follows:

$$T(n) = O(nd) + O(nd) + dT(n/d) \tag{3}$$

where $T(n)$ is the time that the algorithm takes to process n points. Solving the recurrence we get

$$T(n) = O(nd \log_d n) \tag{4}$$

which is the time complexity of the QHV-II algorithm for the best case.

In this scenario, no points are projected to the subsequent subproblems and only the initial n points are used during the algorithm's execution. Therefore, the total space necessary for this scenario is $\mathcal{O}(n)$.

4.2. Theoretical Worst Case Analysis

The worst case occurs when each subproblem has $n-1$ points associated to it. This happens when all the points except the pivot are projected to all the subproblems. The algorithm behaves as in the best case, but this time each subproblem is generated with $n-1$ points. Hence, the recurrence is

$$T(n) = \mathcal{O}(nd) + \mathcal{O}(nd) + dT(n-1) \tag{5}$$

which, when solved, results in

$$T(n) = O(nd \times d^{n-1}) = O(nd^n) \tag{6}$$

which is the worst case time complexity for the QHV-II algorithm.

In this case, at each iteration i, d subproblems with $n-1$ points are created. Hence, the space necessary to store all the points throughout the whole procedure is composed by the space necessary to persist the initial set of n points and the subsequent $d(n-i)$ points at each iteration. The whole space complexity can be described by

$$\begin{aligned}\mathcal{O}\left(n + \sum_{i=1}^{n}(n-i)d\right) &= \mathcal{O}(n) + \mathcal{O}\left(d\sum_{i=1}^{n}(n-i)\right)\\ &= \mathcal{O}(n) + \mathcal{O}\left(d\frac{n(n-1)}{2}\right)\\ &= \mathcal{O}(n) + \mathcal{O}(dn^2)\\ &= \mathcal{O}(dn^2+n).\end{aligned} \tag{7}$$

5. Quadtree

A quadtree is an hierarchical data structure with a form of a tree and was first introduced by R. A. Finkel and J. L. Bentley in [6] where it was used to efficiently persist and retrieve two dimensional points. It was later applied to the $d-$dimensional optimization problems by W. Habenicht in [7] with the main idea of using a quadtree to efficiently identify, persist and retrieve nondominated points.

Given the nature of the QHV-II algorithm and, in particular, the way it partitions the space in subproblems guides us to believe that a quadtree would seem an appropriate choice to store the problem's nondominated points. There are three core ideas behind using a quadtree for the incremental hypervolume calculation problem: 1) the concept of successorship; 2) the point insertion; and 3) the discovery of dominated points. The first two ideas are core for a normal quadtree which serves to store any kind of points in some dimension d, whereas the third idea shall be discussed in a context of a domination free quadtree which is defined further.

Definition 10 (ϕ-successor)**.** *Let $p, q \in \mathbb{R}^d$ and $\phi \in \{0,1\}^d$ such that*

$$\phi_i = \begin{cases} 1 & \text{if } q_i \geqslant p_i \\ 0 & \text{if } q_i < p_i \end{cases},$$

then q is $\phi = \phi_1 \ldots \phi_d$-successor of p.

The above definition is enough to construct a quadtree from a set of points. The set is iterated over and the points are inserted in the tree based on their successorship relative to the points already in the tree. This means that the order of the inserted points matters when constructing a quadtree, i.e., two different insertion sequences might result in two different quadtrees.

Given that there could be points which are $0\ldots0$- and $1\ldots1$-successors of some other point, the idea of having a quadtree consisting only of mutually nondominated points arises inevitably. This quadtree would likely ease the hypervolume calculation process as we would no longer need to worry about the dominated points which could appear in large amounts.

Definition 11 (Domination Free Quadtree)**.** *A quadtree is said to be domination free if and only if it only contains mutually nondominated points if and only if there are no $0\ldots0$- or $1\ldots1$-successors in the quadtree.*

6. IQHVII

The IQHVII algorithm has also been implemented in Kotlin and uses a domination free quadtree to store the nondominated points.

It is interesting to observe that to insert p into a domination free quadtree with r as its root and p being ϕ-successor of r we do not need to check every single tree branch. The only places in the quadtree where there might be points *dominated* by p are the subtrees whose roots are r's ψ-successors where ψ has 0 in at least same positions as ϕ. Furthermore, the only places where there might be points that *dominate* p are the subtrees whose roots are r's ψ-successors where each ψ has 1 in at least same positions as ϕ.

The insertion algorithm presented as Algorithm 2 takes the ideas above into account when p is inserted. It first verifies if the quadtree is empty, and if so, sets p as its root. Next, it searches through the branches that might have points that dominate p. This procedure is described in Algorithm 3 and makes use of an auxiliary procedure $\Phi(a, b)$ which computes ϕ-successorship of b relative to a. If there is some point that dominates p, p is immediately discarded. If the search is unsuccessful, i.e., there are no such points, then the algorithm advances to the next phase. It then scans the branches that might have points dominated by p and, if there is any, adds the tree whose root is that point to a simple list, *treesToReinsert*, so the point's subtrees can be reinserted, and removes it from the original quadtree. The dominated point itself is added to the *dominatedPoints* set which contains all the points dominated by p. The procedure of finding dominated points is presented as Algorithm 4.

Next, the *treesToReinsert* list is iterated over and the points are reinserted into the quadtree. Their removal and further reinsertion is necessary because a point's position in a tree is decided based on the points already in that tree. Thus, by removing a dominated point, its children will be reinserted relatively to a different set of points, potentially altering the tree structure. Note that when reinserting those points, it is only necessary to check if they are dominated by the newly inserted point p to possibly discard them as p is the only point in the structure that wasn't yet compared to those points.

Algorithm 2 Insert into a domination free quadtree (IDFQT)

Input: A quadtree T and a point $p \in \mathbb{R}^d$.
Output: A boolean whether p was successfully inserted or not.

```
 1: if $T$ is empty then
 2:     $root \leftarrow p$
 3:     return true
 4: end if
 5: if pointIsDominated$(T, p)$ then
 6:     return false
 7: end if
 8: if treeShouldBeReinserted$(T, p)$ then
 9:     $treesToReinsert.append(T)$
10:     $root$'s children $\leftarrow \{\}$
11:     $root \leftarrow p$
12: end if
13: insertPoint$(T, p)$
14: return true
```

Next, the point p is inserted based on its ϕ-successorship throughout the quadtree which is represented in Algorithm 5. After p's insertion, the trees in the *treesToReinsert* list are reinserted using the *reinsertTree* procedure which recursively inserts the points of a tree into another tree.

During the whole insertion process several extra dominated points are generated by mixing p with each of the points it was compared to. To do so, the MIX procedure is used. It takes two points a and b and creates a new point which has the minimum of each coordinate between the two points. The idea behind the extra dominated points is that if a point q is in the tree branch scanned by p and is not dominated by p and p is q's $\ldots 0 \ldots 1 \ldots$-successor then q has a value of at least one coordinate greater than p's meaning that the hyperoctant generated by p will surely intersect the one generated by q, creating an extra dominated point. These extra dominated points enable the ability of always performing hypervolume calculations relatively to the original reference point.

Finally the hypervolume of the points in *dominatedPoints* list is calculated and subtracted from the inclusive hypervolume of p. The resulting hypervolume is then added to the total.

The whole described procedure of inserting a point and calculating the new total hypervolume is the IQHVII algorithm and is presented as Algorithm 6.

Algorithm 3 Check if a point is dominated by some other point in a tree (pointIsDominated)

Input: A quadtree T and a point $p \in \mathbb{R}^d$.
Output: A boolean whether p is dominated by some point in T or not.

```
 1: if $T$ is empty then
 2:     return false
 3: end if
 4: if $root$ dominates $p$ then
 5:     return true
 6: end if
 7: $\phi \leftarrow \Phi(root, p)$
 8: for all $child \in root$'s children do
 9:     $child_\phi \leftarrow \Phi(root, child.root)$
10:     if $child_\phi$ has ones in the same positions as $\phi$ then
11:         if pointIsDominated$(child, p)$ then
12:             return false
13:         end if
14:     end if
15: end for
16: $dominatedPoints.append($MIX$(root, p))$
17: return false
```

Given the triviality and lack of excitement the scenario with no points in the domination free quadtree brings to the table, the algorithm's complexity analysis was performed assuming that there are n points already inserted in the quadtree with the point r as its root.

Algorithm 4 Check if a tree should be reinserted (treeShouldBeReinserted)

Input: A quadtree T and a point $p \in \mathbb{R}^d$.

Output: A boolean whether T should be reinserted or not.

```
1: if $T$ is empty then
2:     return false
3: end if
4: if $p$ dominates $root$ then
5:     return true
6: end if
7: $dominatedPoints.append(MIX(root, p))$
8: for all $child \in root$'s children do
9:     if treeShouldBeReinserted($child, p$) then
10:        $treesToReinsert.append(child)$
11:        delete $child$ from $T$
12:    end if
13: end for
14: return false
```

Algorithm 5 Insert a point p into tree T (insertPoint)

Input: A quadtree T and a point $p \in \mathbb{R}^d$.

```
1: if $T$ is empty then
2:     $root \leftarrow p$
3:     return
4: end if
5: $\phi \leftarrow \Phi(root, p)$
6: $child \leftarrow root$'s child with $\Phi(root, child) = \phi$
7: if $child$ does not exist then
8:     $child \leftarrow$ quadtree with $p$ as its root
9: else
10:    insertPoint($child, p$)
11: end if
12: return
```

Algorithm 6 IQHVII

Input: A domination free quadtree T, current hypervolume H and a new point $p \in \mathbb{R}^d$.

Output: The new total hypervolume.

```
1: $inserted \leftarrow$ IDFQT($T, p$)
2: if $inserted$ then
3:     for all $tree \in treesToReinsert$ do
4:         reinsertTree($T, tree$)
5:     end for
6:     return $H$ − QHV-II($dominatedPoints$) + QHV-II($\{p\}$)
7: end if
8: return $H$
```

6.1. Theoretical Best Case Analysis

The best case scenario occurs when there are no branches in the quadtree that should be scanned for points that might dominate the new point p and the ones that might have points dominated by p, so no points need to be reinserted. Hence, we can arrange the n points in such manner that none of the points belongs to any of those branches.

For instance, imagine that p is $10\ldots0$-successor for r and all the r's children are ψ_i-successors where $0 < i \leqslant numberOfChildren$ and $numberOfChildren$ is given by

$$\begin{aligned}(d-1)\binom{d-2}{2} &= (d-1)\frac{(d-2)!}{2(d-4)!} \\ &= \frac{(d-1)!}{2(d-4)!} \\ &= \frac{(d-1)(d-2)(d-3)(d-4)!}{2(d-4)!} \\ &= \frac{(d-1)(d-2)(d-3)}{2}.\end{aligned} \tag{8}$$

Additionally, none of ψ_i has one in the first position and each of the ψ_i has one in at least some other position. This setup allows p being compared only to r and its children and inserted immediately into the r's $10\ldots0$ branch. Hence, the time p takes to compare itself to the root r and its children is

$$\mathcal{O}(d) + \mathcal{O}\left(d\frac{(d-1)(d-2)(d-3)}{2}\right) = \mathcal{O}(d + d^4). \tag{9}$$

The insertion procedure also generates one dominated point which is the result of mixing r with p. The hypervolume of this point is calculated using QHV-II is $\mathcal{O}(d)$-time and the hypervolume of p is calculated identically. The arithmetic operations done on these hypervolumes are performed in $\mathcal{O}(1)$-time. Therefore, the final time complexity is

$$\begin{aligned}\mathcal{O}(d + d^4) + \mathcal{O}(d) + \mathcal{O}(d) &= \mathcal{O}(d + d^4 + 2d) \\ &= \mathcal{O}(3d + d^4).\end{aligned} \tag{10}$$

The space complexity for the best case is $\mathcal{O}(dn)$-space to persist the quadtree and $\mathcal{O}(d)$-space to persist the generated dominated point, totalling to $\mathcal{O}(d(n+1))$.

6.2. Theoretical Worst Case Analysis

The worst case scenario occurs when the new point p dominates the quadtree's root r and all the points should be reinserted. As the IQHVII first checks if p is dominated by some point in the quadtree, we can arrange all the n points in such manner that p must first be compared to the n points and only then reinsert the dominated tree. This whole procedure would result in the following. It would take $\mathcal{O}(dn)$-time for comparisons done when scanning the branches for points which might dominate p and $\mathcal{O}(d(n-1))$-time to mix each point with p to generate $n-1$ extra dominated points which we will consider to be mutually nondominated for the worst case scenario. When the $n-1$ points would be reinserted, each of them would be compared to p to verify if it is dominated by p, taking another $\mathcal{O}(d(n-1))$-time. The reinsertion of the points is straight forward and it is not necessary to check for dominated points or points that dominate the reinserting point as the only new point in the domination free quadtree is p. Hence, in the worst case scenario, the reinsertion of $n-1$ points would take $\mathcal{O}(dn)$-time. To compute the hypervolume of the n dominated points QHV-II would take $\mathcal{O}(nd^n)$-time in the worst case. The inclusive hypervolume of p can be computed in $\mathcal{O}(d)$-time and the arithmetic operations between the hypervolume values can be executed in $\mathcal{O}(1)$-time. This whole procedure has the total time complexity of

$$\begin{aligned}\mathcal{O}(2dn) &+ \mathcal{O}(2d(n-1)) + \mathcal{O}(nd^n) + \mathcal{O}(d)\\ &= \mathcal{O}(d(4n-1)) + \mathcal{O}(nd^n) \qquad (11)\\ &= \mathcal{O}(n(4d+d^n)).\end{aligned}$$

In this scenario $\mathcal{O}(dn)$-space is necessary to store the initial points in the domination free quadtree, $\mathcal{O}(dn)$-space for the generated dominated points and $\mathcal{O}(dn)$-space for the points to be reinserted, totalling to $\mathcal{O}(3dn)$-space complexity.

7. Results

Both of the implemented algorithms were experimentally tested on a machine with 2.3GHz Intel Core i5 processor and 8GB of RAM memory. The algorithms were ran against the same input described in [8] and, also, compared to the execution times of the WFG algorithm described in [9].

The IQHVII tests were executed in the following manner. Given a dataset of n points, the last point is removed and QHV-II is run computing the initial hypervolume of $n-1$ points. During this procedure, the $n-1$ points are inserted into a domination free quadtree. Next, the IQHVII algorithm is ready to be tested, and the n-th point is inserted and the hypervolume is updated. The time it takes to insert the n-th point and to update the hypervolume is the IQHVII's total execution time.

It can be seen that, in all of the scenarios that have been tested, the WFG algorithm behaves much better than the implemented QHV-II. From the theoretical analysis this outcome was expected as QHV-II has worse time complexity than WFG. Nonetheless, there are several technical issues that could have impacted the performance of QHV-II. First, it was implemented in Kotlin using object oriented programming paradigm instead of using a language like C for a more optimized performance and memory managing. And second, the implemented algorithm was not optimized to run some other algorithm for subproblems, which were small enough, in order to achieve faster execution times in those cases.

For the IQHVII's case, it is clear that IQHVII is much faster than QHV-II, so we can freely conclude that it would be better to add a new point, calculate its exclusive hypervolume and add it to the total hypervolume using IQHVII rather than to recalculate completely the hypervolume using QHV-II. Nevertheless, IQHVII ran a little slower than WFG.

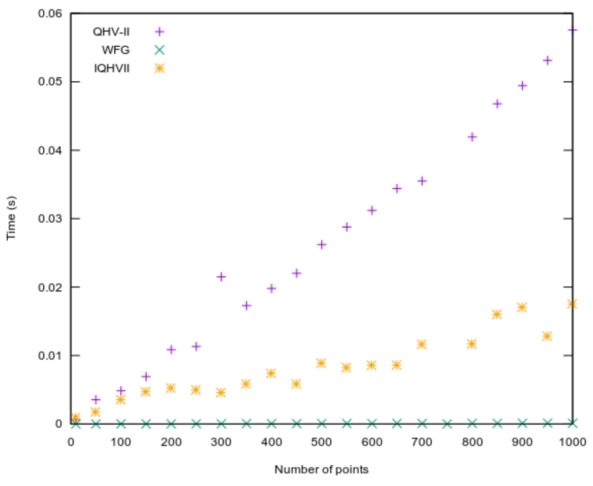

Figure 3: IQHVII vs. QHV-II vs. WFG for a degenerate dataset in 5 dimensions.

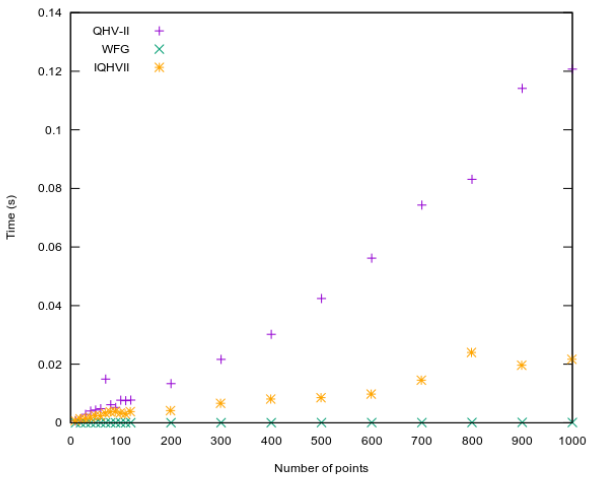

Figure 4: IQHVII vs. QHV-II vs. WFG for a degenerate dataset in 6 dimensions.

8. Conclusions

The results of the theoretical analysis of the incremental implementation of the QHV-II algorithm

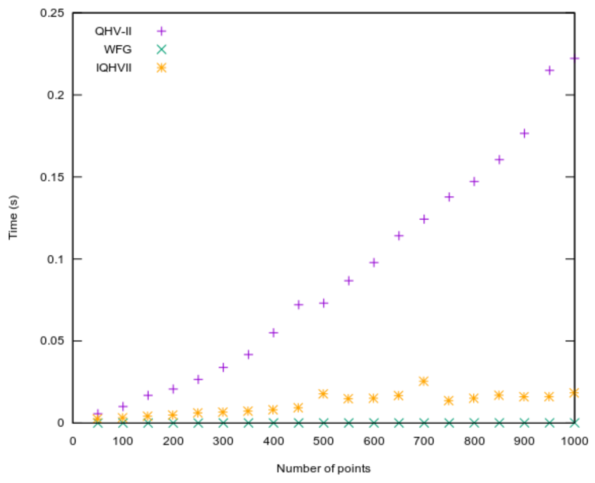

Figure 5: IQHVII vs. QHV-II vs. WFG for a degenerate dataset in 7 dimensions.

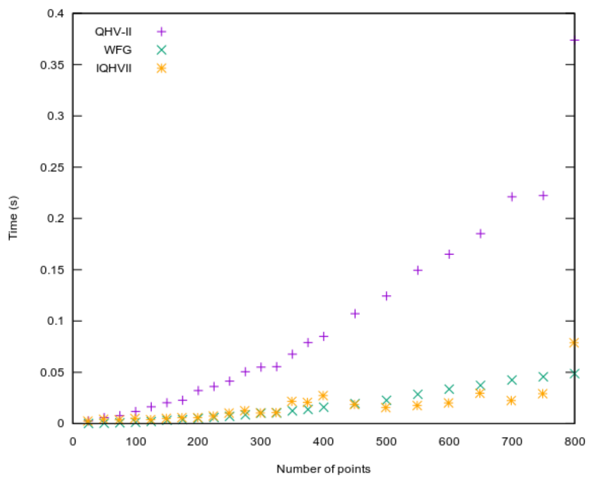

Figure 8: IQHVII vs. QHV-II vs. WFG for a random dataset in 6 dimensions.

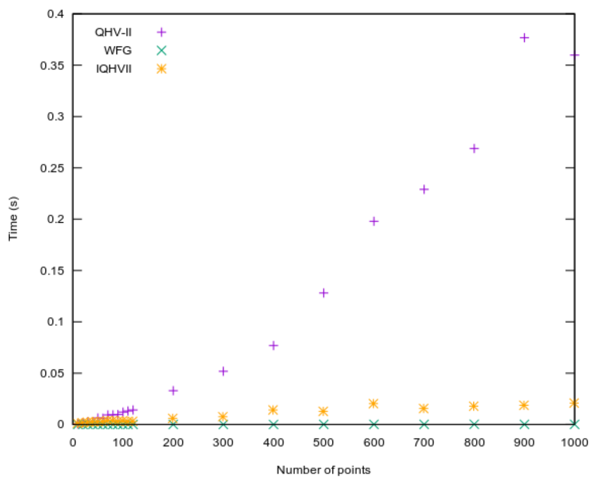

Figure 6: IQHVII vs. QHV-II vs. WFG for a degenerate dataset in 8 dimensions.

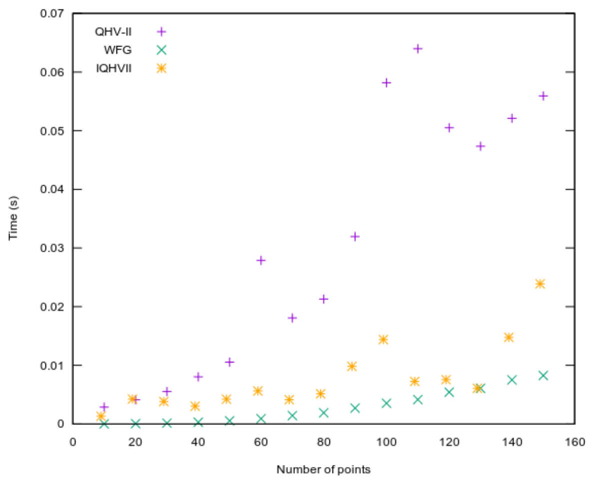

Figure 9: IQHVII vs. QHV-II vs. WFG for a random dataset in 7 dimensions.

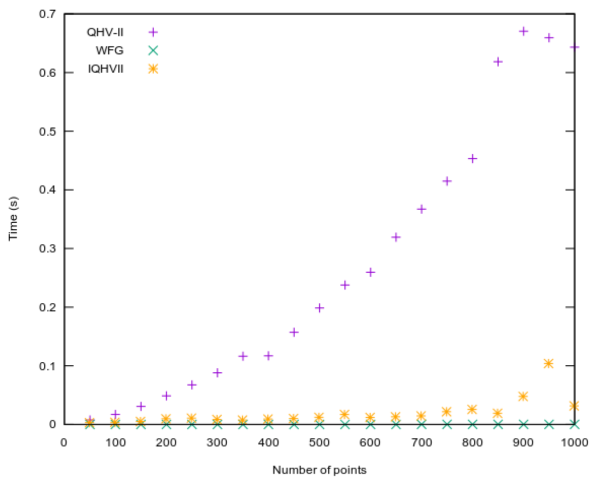

Figure 7: IQHVII vs. QHV-II vs. WFG for a degenerate dataset in 9 dimensions.

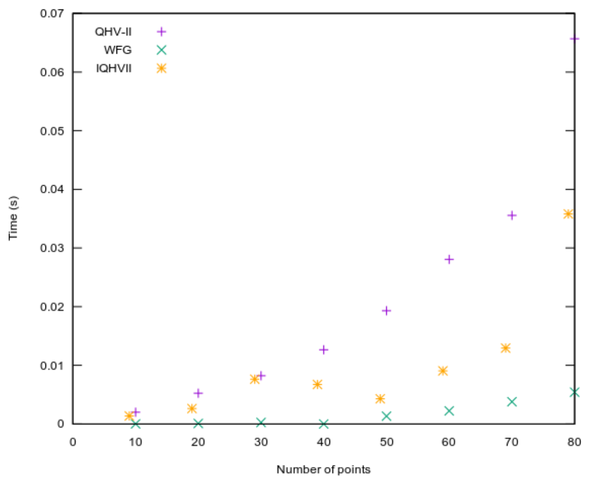

Figure 10: IQHVII vs. QHV-II vs. WFG for a random dataset in 8 dimensions.

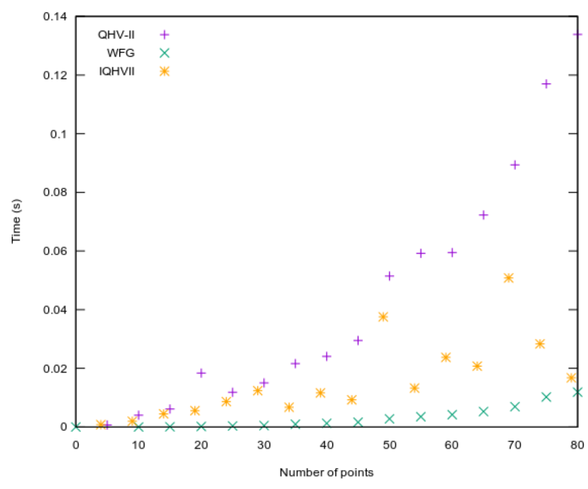

Figure 11: IQHVII vs. QHV-II vs. WFG for a random dataset in 9 dimensions.

show that it is faster to compute the hypervolume of a set of points to which a new point p has been added by computing p's exclusive hypervolume and adding it to the total hypervolume instead of recomputing the whole hypervolume using the QHV-II algorithm. The experimental results confirmed the theoretical analysis and showed that the IQHVII algorithm outperforms QHV-II. Although it mostly ran slower than WFG, the difference was slight and the reasoning could be the difference between the programming languages both algorithms were implemented in and the optimization levels each of them suffered.

8.1. Future Work

There are quite a few ideas for the future work on the incremental implementation of the QHV-II algorithm. It would be interesting to see if the IQHVII algorithm can efficiently generate mutually nondominated points dominated by the new point p and study whether calculating the hypervolumes relative to a point different from the original reference point is possible. Both of these ideas have the potential to reduce the size of the problem of calculating the hypervolume of points dominated by p. Also, it would be very interesting to compare an implementation which would allow adding n new points at a time with the case of updating the hypervolume n times using IQHVII. Finally, it would be great to see the IQHVII algorithm implemented in a language like C, optimized for some base cases and ran using parallel threads.

References

[1] Andrzej Jaszkiewicz. Improved quick hypervolume algorithm. *CoRR*, abs/1612.03402, 2016.

[2] Karl Bringmann and Tobias Friedrich. Approximating the least hypervolume contributor: Np-hard in general, but fast in practice. *Theoretical Computer Science*, 425:104 – 116, 2012. Theoretical Foundations of Evolutionary Computation.

[3] José Rui Figueira. *Slides das aulas de Complementos de Investigação Computacional.* 2018. MEGI, IST, Lisboa.

[4] Andreia P. Guerreiro and Carlos M. Fonseca. Computing and updating hypervolume contributions in up to four dimensions. *IEEE Trans. Evolutionary Computation*, 22(3):449–463, 2018.

[5] L. M. S. Russo and A. P. Francisco. Quick hypervolume. *IEEE Transactions on Evolutionary Computation*, 18(4):481–502, Aug 2014.

[6] R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, Mar 1974.

[7] W. Habenicht. Quad trees, a datastructure for discrete vector optimization problems. In Pierre Hansen, editor, *Essays and Surveys on Multiple Criteria Decision Making*, pages 136–145, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg.

[8] Data sets used for computational experiments. *www.wfg.csse.uwa.edu.au/hypervolume.*

[9] L. While, L. Bradstreet, and L. Barone. A fast way of calculating exact hypervolumes. *IEEE Transactions on Evolutionary Computation*, 16(1):86–95, Feb 2012.