Internal Report 2013–12 August 2013

Universiteit Leiden
Opleiding Informatica

Complexity Reduction and Validation
of Computing the
Expected Hypervolume Improvement

Iris Hupkens

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Expected improvement algorithms are commonly used in global optimization problems where evaluating the objective function is costly. The Expected Hypervolume Improvement (EHVI) is a recent generalization of these algorithms to multiobjective optimization. The computation of the EHVI is based on a multidimensional integration of a piecewise defined nonlinear function. Exact calculation of the EHVI has so far only been possible in 2-D, and even there it is slow. In higher dimensions it can so far only be approximated, for instance by Monte Carlo integration, and no expression/algorithm for direct integration is available.

In this thesis, a new algorithm is devised for the exact calculation of the EHVI in higher dimensions, and its correctness is experimentally verified in three dimensions. Additionally, fast computation schemes are proposed for the exact calculation of the EHVI in two and three dimensions, with time complexity in $O(n^2)$ (previously $O(n^3 \log n)$) and in $O(n^3)$, respectively. Empirical tests show that for Pareto front approximations of modest size (< 100 points) with the new algorithm computation times in the order of a second or less are required to perform exact calculations in three dimensions and imprecise Monte Carlo integration is no longer required. The algorithms have been implemented in C++ and can be readily used in global multiobjective optimization algorithms which use the EHVI.

Contents

Chapter 1

Introduction

In multiobjective optimization, the goal is to find a set of solutions which optimizes multiple objective functions at the same time. Sometimes the function values of solutions can only be determined through costly simulations, so approximation functions are used in their place. This makes it possible to evaluate the function values of the most promising individuals only, instead of wasting time evaluating the function values of individuals that are unlikely to result in an improvement. Given a predictive distribution of the expected function values, the Expected Hypervolume Improvement (or EHVI for short) represents the expected improvement in the hypervolume measure of the solution set [1]. The hypervolume measure itself is a common measure used to determine the quality of a set of solutions to a multiobjective optimization problem [2], which makes the EHVI a natural quality measure to use in multiobjective surrogate-assisted optimization.

The calculation of the EHVI has so far been a problem. Monte Carlo integration can solve the issue of computing the EHVI directly, but to get an accurate approximation out of Monte Carlo integration is slow. An exact calculation approach exists, but it is slow as well, and does not work in dimensions higher than 2. This thesis aims to increase the speed of exact calculation of the EHVI in 2 dimensions, as well as provide a working method of calculating it in higher dimensions. To verify whether it is working, it will be compared to the results from Monte Carlo integration. The empirical performance of directly calculating the EHVI in the three-dimensional case will also be analyzed in order to show the feasibility of using direct calculations in place of Monte Carlo integration.

This thesis is structured as follows:

- Chapter 2 contains a partial summary of technical preliminaries required to understand the thesis.
- Chapter 3 summarizes some related work.
- Chapter 4 contains a proof that the exact calculation of the 2-D EHVI can be done in $O(n^2)$, as well as a proof that the worst-case complexity of calculating the

EHVI in the 2-D case cannot be less than $O(n \log n)$. It also contains the results of an empirical test comparing the performance of the newly-provided algorithm to a naive ($O(n^3 \log n)$) implementation.

- Chapter 5 describes the details of calculating the expected hypervolume improvement in more than 2 dimensions.
- Chapter 6 describes a method for determining the 3-D EHVI with time complexity $O(n^3)$.
- Chapter 7 describes the results of empirical tests of implementations of the algorithms described in Chapter 5, both to verify their correctness and to measure their performance.
- Finally, Chapter 8 contains some concluding remarks and an outline of promising directions for future research.

Chapter 2

Preliminaries

Without loss of generality, we will consider maximization problems in this thesis, meaning that finding the highest possible function values is desired.

A characteristic of multiobjective optimization problems is that there is typically no single best solution. Instead, there is a set of solutions known as the Pareto front, all of which are mutually non-dominated. A solution p *dominates* a solution q if it is strictly better than q: that is, all of p's function values are either greater than or equal to q's corresponding function value, and $p \neq q$. The Pareto front can consist of an infinite number of solutions, so finding a finite Pareto approximation set is typically desired. The hypervolume measure is a quality measure for such Pareto approximation sets, which is defined as the set's dominated hypervolume.

The dominated hypervolume of a set of points P with respect to a reference point r is the hypervolume covered by the boxes that have an element of P as their upper corner and r as their lower corner. The set containing the part of the objective space that is dominated by the points in P will be referred to as $\mathrm{DomSet}(P)$. The hypervolume contribution of a point $p \in P$ is the difference in dominated hypervolume between $P \setminus \{p\}$ and P. The hypervolume contribution of a set of points $S \subseteq P$ is defined analogously, as the difference between $P \setminus S$ and P. The hypervolume improvement of a point $p \notin P$ with respect to P is defined as the hypervolume contribution of p with regards to $P \cup \{p\}$.

The expected hypervolume improvement is the expected hypervolume contribution of a new candidate point, for which the exact function values are not yet known, but which has an associated predictive distribution function (PDF). The formula for the expected hypervolume improvement of a candidate point with respect to a mutually non-dominated set P is:

$$\int_{p \in R} \mathrm{HI}(p, P) \cdot PDF(p) \mathrm{d}p$$

Here R is $\mathbb{R}^m$ for an m-dimensional objective space. The hypervolume improvement function $\mathrm{HI}(p, P)$ will be 0 in the area of the objective space that falls inside the dominated hypervolume of P, as well as in the area that fails to dominate r.

In [3], a formula is derived for exactly calculating this integral. A rectangular expected improvement is calculated, and a correction term representing the dominated hypervolume of P within this rectangular volume is then subtracted. While the formula is correct in the 2-D case, it is incorrect for higher dimensions due to the more complex shape of the dominated hypervolume of P. Due to its complexity, the full formula is omitted from these preliminaries. One part of the formula which will be examined in more detail in Chapter 2.1 is a formula for calculating a partial one-dimensional expected improvement without numerical integration.

For a given mean and standard deviation vector of an independently distributed predictive distribution, the expected hypervolume improvement has desirable monotonicity properties:

1. If two predictive distributions p and q have the same variance, and p's mean vector dominates q's mean vector, then p will have a higher expected hypervolume improvement than q.
2. If two predictive distributions p and q have the same mean value, then the predictive distribution with the highest variance in its components will have a higher expected hypervolume improvement.

The first property was proven in [1]. The second property was analytically proven for the two-dimensional case in [3], but it still needs to be verified whether this property holds in higher dimensions.

2.1 Exact Calculation of Partial One-Dimensional Improvements

In order to calculate the EHVI, we will need to calculate a lot of integrals that have the form of a partial one-dimensional improvement. In [3], a function was derived that could be used for that purpose. We will be using the ψ function as short-hand for the following:

$$\psi(a, b, \mu, \sigma) = \sigma \cdot \phi(\frac{b-\mu}{\sigma}) + (a-\mu)\Phi(\frac{b-\mu}{\sigma})$$

Here, ϕ is the function for the probability density function of the standard normal distribution and Φ is the cumulative probability distribution function of the standard normal distribution. The arguments μ and σ represent the mean value and standard deviation, respectively. Both ϕ and Φ can be evaluated without numerical integration:

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \qquad \Phi(x) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$$

The function ψ can be used to calculate integrals that have the form of a partial expected improvement from b to ∞, as is captured in the following equality:

$$\int\limits_{z=b}^{\infty} (z-a)\frac{1}{\sigma}\phi(\frac{z-\mu}{\sigma}) = \sigma \cdot \phi(\frac{b-\mu}{\sigma}) + (a-\mu)\Phi(\frac{b-\mu}{\sigma})$$

The full 1-D expected improvement can be seen as a special case of this integral, where $a = b$. Integrals whose upper limit is less than ∞ can be written as the sum of two such integrals, allowing partial expected improvements over an interval $[l, u) \subset \mathbb{R}$, $l \geq f'$ to be calculated using ψ:

$$\begin{aligned}
&\int\limits_{z=l}^{u} (z-f')\frac{1}{\sigma}\phi(\frac{z-\mu}{\sigma})\mathrm{d}z \\
&= \int\limits_{z=l}^{\infty} (z-f')\frac{1}{\sigma}\phi(\frac{z-\mu}{\sigma})\mathrm{d}z - \int\limits_{z=u}^{\infty} (z-f')\frac{1}{\sigma}\phi(\frac{z-\mu}{\sigma})\mathrm{d}z \\
&= \psi(f', l, \mu, \sigma) - \psi(f', u, \mu, \sigma)
\end{aligned}$$

The value f' in this case is the current best function value, or *incumbent solution*.

Note that to save on space, in the rest of this thesis we will use ϕ_x and Φ_x to denote the partial probability distribution function and cumulative probability function of the given probability distribution function PDF in the dimension x, so $\phi_x(p_x)$ should be read as $\frac{1}{\sigma}\phi(\frac{p_x-\mu_x}{\sigma_x})$, and $\Phi_x(p_x)$ as $\Phi(\frac{p_x-\mu_x}{\sigma_x})$.

Chapter 3

Related Work

Using the one-dimensional expected improvement to solve engineering problems with expensive-to-evaluate objective functions was initially proposed by Mockus [6] and then later picked up again by Jones et al. in [7] and has been widely used in global optimization with expensive-to-evaluate functions since then. It has been shown to converge under some mild assumptions [8].

The meta-model used for calculating the expected improvement can be generated using Gaussian process regression, or Kriging [16]. It is a technique for estimating the mean function value and the expected error using existing function values, based on the assumption that points will have similar function values to the average of other nearby points.

Multi-objective optimization problems are often solved using evolutionary algorithms, such as NSGA-II [12] or SPEA-2 [13]. These algorithms do not require a-priori knowledge of the relative importance of the different objective functions, and instead aim to provide a varied set of solutions for the human decision maker. The quality of such solution sets is often measured using the hypervolume measure, which was described in the previous chapter. Recently, algorithms have been proposed that internally use the hypervolume measure as a selection criterion [14] [15].

Generalizing the one-dimensional expected improvement to multiobjective optimization problems is still a very new area of research. Besides the EHVI, various other solutions have been proposed.

- Chebyshev scalarization with dynamically-changing weights is used in [9].
- Another proposed solution is to perform scalarization by using the distance from the centroid of the probability distribution to the Pareto approximation set [10].
- In [11], the hypervolume improvement for candidate points is calculated based on the upper confidence bound of the meta-model, and this measure is then used to

choose which point to evaluate next.

The EHVI directly corresponds to a selection criterion that has been effectively used in multiobjective optimization, and the monotonicity properties studied in [1] (which the older solutions for generalizing the expected improvement do not have) are expected to make it a worthwhile contribution to the state-of-the-art once it can be calculated quickly and exactly.

Chapter 4

Calculating the 2-D EHVI

Let P denote a set of n mutually non-dominated points in the two-dimensional plane. Furthermore, let r be a reference point which is dominated by every point in P. The aim is to calculate the expected hypervolume improvement for a point p in the decision space for which we have the mean and standard deviation of a predictive distribution describing the likelihood of results of its exact function value. Typically, such predictive distributions stem from meta-models or low fidelity models of an expensive to compute objective function. For the statistical assumptions of such models see, e.g., [5].

In the two-dimensional case, calculating the EHVI for p exactly can be done by piecewise integration over the half-open rectangular interval boxes formed by the horizontal and vertical lines going through the points in P and through r. The final EHVI is the sum of the contributions calculated for all grid cells. See Figure 4.1 for a visualization of the grid.

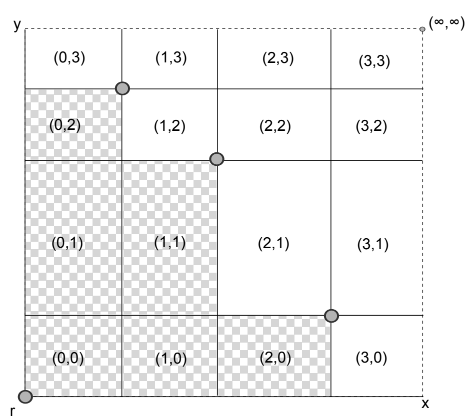

Figure 4.1: An example of the interval boxes for a small population P. Checkered boxes fall in the dominated hypervolume of P. Therefore their contribution to the integral will be 0, and no calculation will be necessary for these boxes.

Individual grid cells will be denoted by $C(a, b)$, where $0 \leq a \leq n$ and $0 \leq b \leq n$. Let

$Q = P \cup (\infty, r_y) \cup (r_x, \infty)$, with Q^x denoting Q sorted in order of ascending x coordinate, and Q^y denoting Q sorted in order of ascending y coordinate. Let C be the set of grid cells representing the interval boxes. The numbers a and b represent positions in the sorting order of Q, starting with 0. Then, a is the position of elements of Q^x and b is the position of elements of Q^y. The lower left corner of a cell will have the coordinates $(Q^x_a.x, Q^y_b.y)$. The upper right corner of the grid cell will have the coordinates $(Q^x_{a+1}.x, Q^y_{b+1}.y)$.

Note, that due to the characteristics of mutually non-dominated points in the two-dimensional plane, it is not necessary to sort Q twice in order to determine Q^x and Q^y. Sorting P in order of ascending x coordinate is equivalent to sorting it in order of descending y coordinate, and the other two points are opposites of each other which are always at the beginning and end of the sorting order. It follows that $Q^x_k = Q^y_{n+1-k}$. Therefore, a single sorting operation with a complexity of $O(n \log n)$ is sufficient for determining the coordinates used for the grid cells.

When dividing the grid in the way described above, $(n + 1)^2$ interval boxes are formed. However, if the upper right corner of an interval box is dominated by or equal to some point in P, its contribution will be zero, and no calculation will need to be done for that interval box. These interval boxes are represented by a grid cell $C(a, b)$ which is within the dominated hypervolume of P. The remaining cells, C_{stairs}, are formed by cells for which this is not the case, meaning that $\forall (C(a, b) \in C_{stairs}, p \in P) : p.x > Q^x_a.x \Rightarrow Q^y_b.y \geq p.y$ and, analogously, $p.y > Q^y_b.y \Rightarrow Q^x_a.x \geq p.x$.

Due to the definition of Q, we know that for $p \in P$ it holds that $p = Q^x_k = Q^y_{n+1-k}$ for some $0 < k \leq n$. If $k > a$ and $n+1-k > b$, p dominates $C(a, b)$. No such point p exists if $b \geq n+1-(a+1)$, so C_{stairs} consists of all cells satisfying $a \geq n-b$. There are $\frac{(n+1)(n+2)}{2}$ of such cells, resulting in a lower bound of $O(n^2)$ on the complexity of any algorithm which iterates over these interval boxes.

If we call the lower corner of the cell l and the upper corner u, the contribution of a grid cell to the integral is defined as follows:

$$\int_{p_y=l_y}^{u_y} \int_{p_x=l_x}^{u_x} \mathrm{HI}(p)\, \phi_x(p_x)\, \phi_y(p_y)\, \mathrm{d}p_x\, \mathrm{d}p_y$$

Dominated cells have a contribution of 0 to the integral, and for cells which are non-dominated, $\mathrm{HI}(p)$ can be calculated as a rectangular volume from which a correction term is subtracted. See Figure 4.2 for a visual representation. The integral for these cells can be calculated as follows, as was described in more detail in [3]:

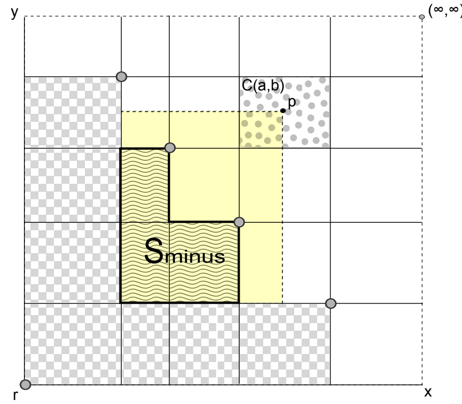

Figure 4.2: Within an integration region $C(a, b)$, the hypervolume improvement of candidate points p is equal to $(p.x - Q^y_{b+1}.x) \cdot (p.y - Q^x_{a+1}.y) - S_{minus}$. In this example, the yellow rectangle represents $(p.x - Q^y_{b+1}.x) \cdot (p.y - Q^x_{a+1}.y)$, and S consists of the two points within the yellow rectangle.

$$
\begin{aligned}
&\int_{p_y=l_y}^{u_y} \int_{p_x=l_x}^{u_x} (p_x - v_x)(p_y - v_y) - S_{minus}\, \phi_x(p_x)\, \phi_y(p_y)\, \mathrm{d}p_x\, \mathrm{d}p_y \\
&= \int_{p_y=l_y}^{u_y} \int_{p_x=l_x}^{u_x} (p_x - v_x)(p_y - v_y)\, \phi_x(p_x)\, \phi_y(p_y)\, \mathrm{d}p_x\, \mathrm{d}p_y \\
&\quad - \int_{p_y=l_y}^{u_y} \int_{p_x=l_x}^{u_x} S_{minus}\, \phi_x(p_x)\, \phi_y(p_y)\, \mathrm{d}p_x\, \mathrm{d}p_y \\
&= (\psi(v_x, l_x, \mu, \sigma) - \psi(v_x, u_x, \mu, \sigma)) \cdot (\psi(v_y, l_y, \mu, \sigma) - \psi(v_y, u_y, \mu, \sigma)) \\
&\quad - S_{minus} \cdot \Phi_x(u_x) - \Phi_x(l_x) \cdot \Phi_y(u_y) - \Phi_y(l_y)
\end{aligned}
$$

The last step is motivated by Section 2.1 and the application of Fubini's Theorem [17]. It can be seen that the formula is of the form $c_1 - S_{minus} \cdot c_2$, where c_1 and c_2 are calculations which are performed in constant time with respect to n for a single cell.

The correction term S_{minus} is equal to the hypervolume contribution of $S \subseteq P$, where S consists of those points dominated by or equal to the lower corner of the cell. Calculating the dominated hypervolume of a set in the two-dimensional plane has a time complexity of $O(n \log n)$. This complexity results from needing to find the neighbors of each point in order to calculate its contribution to the hypervolume. Sorting the set has a time complexity of $O(n \log n)$, after which the dominated hypervolume calculation itself is done in $O(n)$ by iterating over each point and performing an $O(1)$ calculation using the points that come before and after it in the sorting order. When calculating S_{minus}, the points for which the dominated hypervolume is to be calculated come from P, which was already

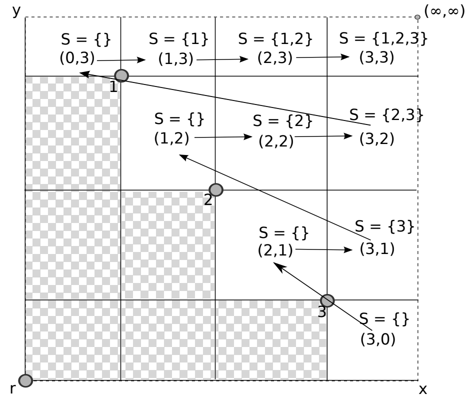

Figure 4.3: An example showing the order of iterations which allows the hypervolume contribution of S to be updated in constant time.

sorted. This brings the complexity of this step down to $O(n)$, but it can be brought down to $O(1)$ when the order of calculations is chosen carefully, giving the algorithm a total complexity of $O(n^2)$.

The points dominated by or equal to the lower corner of $C(a, b)$, which define S, are those points satisfying the following equality:

$$p \in P, Q^x_a.x \geq p.x, Q^y_b.y \geq p.y$$

Because of the sorting order and definition of Q^x and Q^y, this is equivalent to finding the range of points from $Q^x_{(n+1-b)}$ to Q^x_a (limits included). S will be empty if $Q^x_a = Q^x_{(n-b)}$, which is the lowest value of a for which $a \geq n - b$ holds, and otherwise it will form an uninterrupted range with $Q^x_{(n+1-b)}$ as its first element and Q^x_a as its last element.

A row in C_{stairs} is a set of cells $C_{stairs}(a, b)$ where b is the same. In a single row, S will always be either empty or have Q^x_a as its last element. Adding 1 to a adds one point to the range of points in P which falls between Q^x_a and $Q^x_{(n+1-b)}$. This makes it possible to iterate over all cells in C_{stairs} while adding no more than one point to S per iteration. We will do this as follows:

We will start iterating over each row of C_{stairs} at its first cell, where $a = n - b$. In this cell, $S = \emptyset$ and $S_{minus} = 0$. For each iteration within a row after the first one, we add 1 to a and add the point Q^x_a to S. For an example, refer to Figure 4.3 at the end of this section, which shows the order of operations and the contents of S during each step.

Although the above description refers to 'adding points to S', we only need to keep track of the first and last points of S in between algorithm iterations. When a new point is added to S, S_{minus} increases by the area covered by the rectangle from $(Q^x_a.x, Q^x_a.y)$ to the boundary point $(Q^x_{(n-b)}.x, Q^x_{a+1}.y)$. Therefore, to update S_{minus} after the addition of a point to S, only the left neighbor of the first element of S, the last element of S and the right neighbor of the last element of S are needed. Figure 4.4 shows an example of this process. This can be done in constant time in any data structure which allows the neighbors of a point to be looked up in constant time: whenever a is incremented, Q^x_a becomes Q^x_{a+1} and Q^x_{a+1} becomes its right neighbor, Q^x_{a+2}. Whenever b is incremented, the new $Q^x_{(n-b)}$ becomes its left neighbor, $Q^x_{(n-1-b)}$, and as we will then start iterating through values of a at the beginning of the row, Q^x_a becomes the new $Q^x_{(n-b)}$ as well because we have established earlier that $Q^x_a = Q^x_{(n-b)}$ in the first cell in a row of C_{stairs}.

We have shown that the upper bound on the complexity of determining the expected hypervolume improvement is $O(n^2)$. We can also show that the worst-case complexity can be no better than $O(n \log n)$. If the standard deviation of a candidate point's predictive distribution is 0 and the mean value vector is a point which dominates all points in P, then the problem of calculating its EHVI reduces to calculating the hypervolume that will be dominated by $p_{candidate}$ minus the hypervolume dominated by P. If it was possible to solve this calculation with lower complexity than $O(n \log n)$, then it would also be possible to reduce the calculation of P's hypervolume to the problem of calculating the EHVI of a point that dominates P, and it has already been proven in [4] that the complexity of calculating the hypervolume of a set of points in the 2-D plane is in $\Theta(n \log n)$.

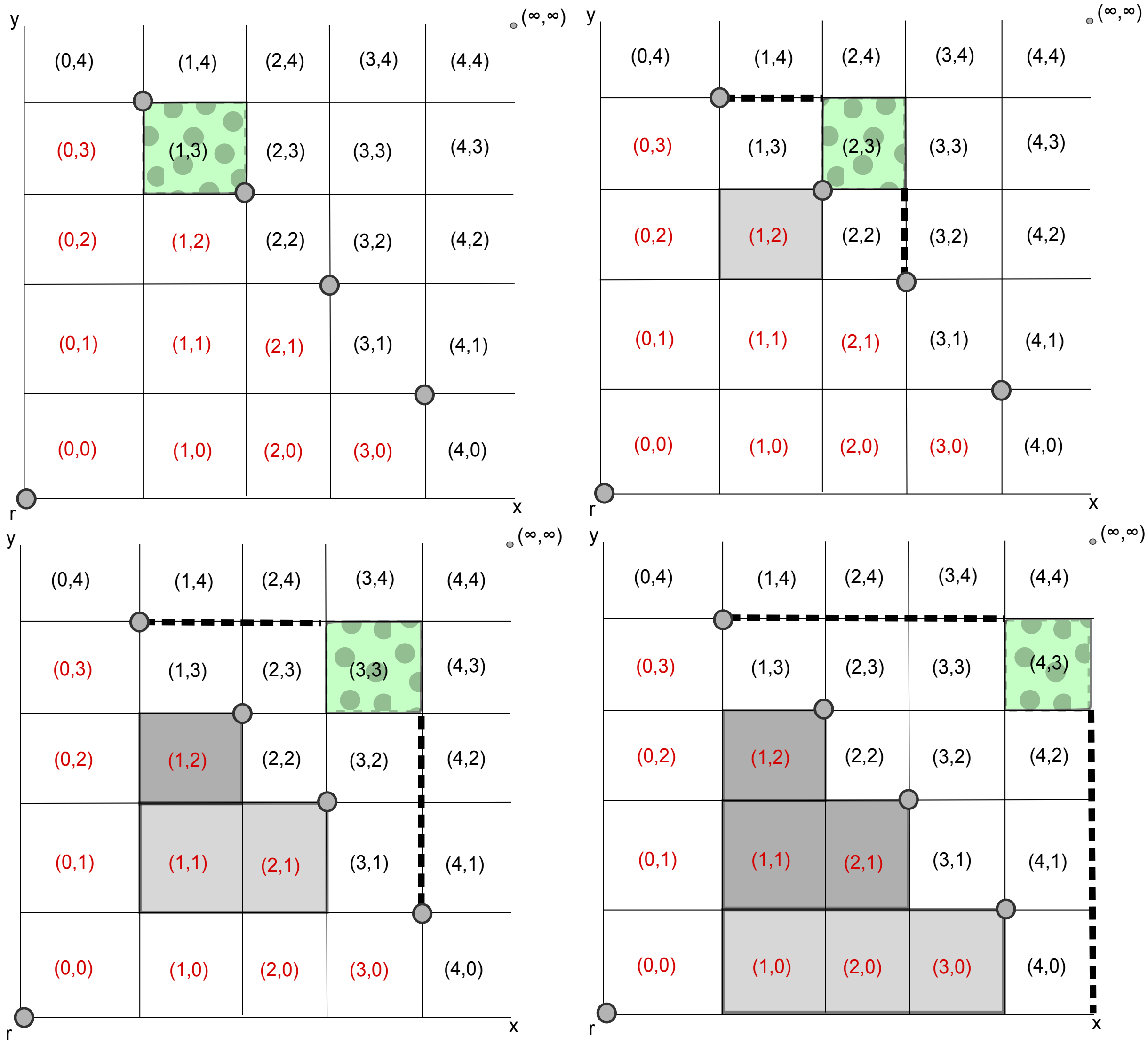

Figure 4.4: An example showing how S_{minus} changes during each iteration within a single row. The rectangular strip which is added after each iteration can be calculated with knowledge of three points: the point Q^x_a is its upper corner, the point Q^x_{a+1} provides the y coordinate of its lower corner, and the point $Q^x_{(n-b)}$ provides the x coordinate of its lower corner. Because $Q^x_{(n-b)}$ does not change, the hypervolume covered by the older points in S stays the same and does not have to be re-calculated.

4.1 Empirical Performance

As an additional verification of the correctness of the algorithm presented above, two implementations were written in C++. The first used the constant-time update scheme, and the second did not: instead of using the constant-time update scheme, S_{minus} was calculated by first finding the set of points S by checking each point in P to see if it was dominated, and then calling a separate function on S to calculate the hypervolume of this set of points.

The expected hypervolume improvement calculated using these implementations was identical for all test problems, but their speed was not. See Figure 4.5 for the empirical performance on a simple test where P consisted of n different points on a diagonal Pareto front. From this, it appears that using the constant-time update scheme becomes worthwhile for $n > 20$, though results might vary slightly depending on implementation and system details.

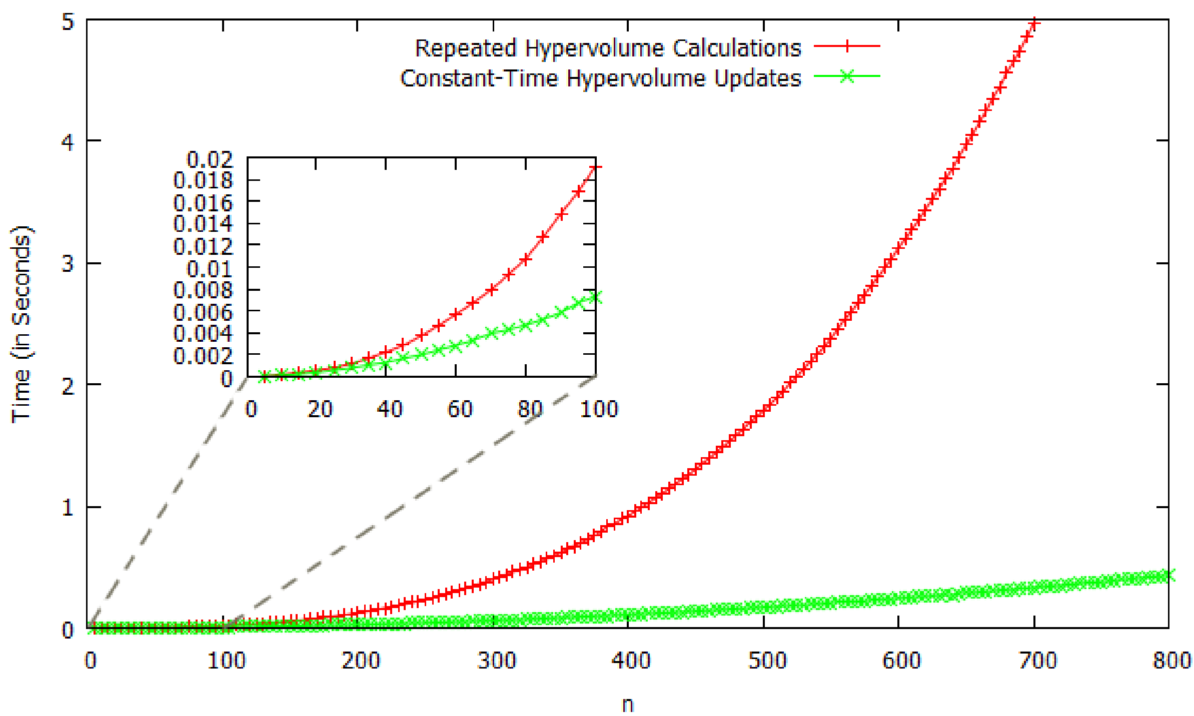

Figure 4.5: Time needed to calculate the expected hypervolume improvement in 2-D, averaged over 10 runs. The times reported were measured on an Intel i7 quadcore CPU with 2.1 GHz clockspeed, and the code was compiled using GNU under Windows with the optimization level set to O3.

Chapter 5

Calculation of the Higher-Dimensional EHVI

The algorithm given in [3] for exactly calculating the expected hypervolume improvement is not correct when the number of dimensions is higher than 2. This is because the shape of the hypervolume improvement becomes more complex when the number of dimensions increases. We will therefore derive a new formula by first decomposing the calculation into parts with less complex shapes, and then simplifying the resulting formula for the sake of more convenient calculation.

5.1 Decomposition Into Parts

In higher dimensions, the search space can be divided into cells the same way it is done in two dimensions, except instead of the boundaries being given by lines going through the points in P and the reference point r, now the cells are separated from each other by $(m-1)$-dimensional hyperplanes (where m is the number of dimensions).

Each cell is denoted by $C(a_1, a_2, \ldots, a_m)$ where a_1 through a_m are non-negative integers representing an associated point in $P \cup \{r\}$. The coordinates of the lower corner and upper corner of the cell can be derived from these integers by sorting P in ascending order of the values of each objective function. We will use the variable l to refer to the lower corner of the cell. If P^d is P sorted in the dth objective function, then the dth function value of l (l_d) is equal to the d-th function value of the a_d-th element of P^d, unless a_d is 0. In that case, $l_d = r_d$. The upper corner u is given by finding the next point in each sorting order and using the dth function value from that, or ∞ if a_d was equal to n and no next point thus exists.

The hypervolume improvement of a point p with respect to P is given by the function $\mathrm{HyperVolume}\,(A \setminus \mathrm{DomSet}(P))$, where A is the dominated hypervolume covered by p.

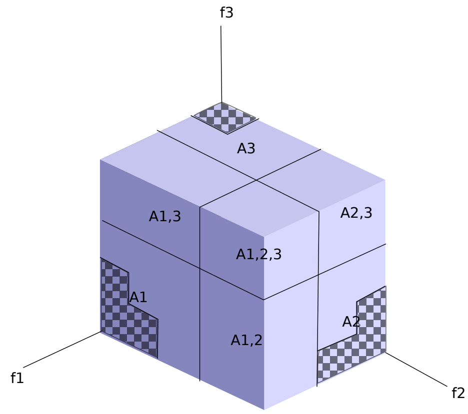

Figure 5.1: An example showing how the quantities A_C are defined in a three-dimensional objective space. $A_\emptyset$ is within the rectangular volume. The checkered areas represent the dominated hypervolume of $P \cap A$.

This is the same as calculating $\text{HyperVolume}(A) - \text{HyperVolume}\,(\text{DomSet}(P) \cap A)$. We will denote the set of dimensions by $D = \{1, 2, \dots, m\}$. We can decompose the calculation of the hypervolume improvement of a point $p \in C(a_1, a_2, \dots, a_m)$ as follows:

$$\text{HI}(p) = \sum_{C \subseteq D} I_C$$
$$I_C = \text{HyperVolume}\,(A_C) - \text{HyperVolume}\,(\text{DomSet}(P) \cap A_C)$$

A_C is given by:

$$A_C = \left[\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_k \end{pmatrix}, \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{pmatrix} \right]$$
$$v_d = \begin{cases} l_d & d \in C \\ r_d & d \notin C \end{cases}$$
$$w_d = \begin{cases} p_d & d \in C \\ l_d & d \notin C \end{cases}$$

See Figure 5.1 for an example in 3 dimensions.

The values of r_d and l_d are constant for all points that fall within a given interval box: r is the reference point and is, of course, always constant, while l represents the position of the lower corner of the cell. From this, it follows that I_C represents the portion of the hypervolume improvement which is constant with regards to the values of $p_d, d \notin C$, and which is variable with regards to the values of $p_d, d \in C$. In fact, it is *linearly related* to these values. This is a direct consequence of the way the cell boundaries are defined:

Let S_C be a cross-section of $\mathrm{DomSet}(P) \cap A_C$ which goes through p. This cross-section is defined by a projection to the dimensions not in C (if C consists of k dimensions, the slice will be $(m-k)$-dimensional as a result). The projection of $\mathrm{DomSet}(P)$ uses only those points in P for which the function values in the dimensions given by C are larger than the corresponding function values of p. We shall call this selection P'. No points in P can fall between cell boundaries in any dimension, so the composition of P' must be the same for all points within a cell. The projection of A_C to the dimensions not in C is constant for all points within a cell as well, because the coordinates defining A_C are independent of p in all dimensions not in C. $\mathrm{HyperVolume}(S_C)$ is constant as a result. Because A_C does not span across cell boundaries in the dimensions in C, $\mathrm{HyperVolume}(\mathrm{DomSet}(P) \cap A_C)$ is equal to the hypervolume of S_C multiplied by the length of A_C in all dimensions in C, and those lengths are given by $(p_d - l_d)$ with $d \in C$.

There is one quantity I_C for which $C = D$. This quantity I_D is special because it is linearly related to all values of p. I_D falls entirely within the cell, and as such, instead of projecting P onto a zero-dimensional space, it can simply be said that $\mathrm{HyperVolume}(\mathrm{DomSet}(P) \cap A_D) = \mathrm{HyperVolume}(A_D)$ if the cell is dominated, and $\mathrm{HyperVolume}(A_D \cap \mathrm{DomSet}(P)) = 0$ if it is not. Therefore, $I_D = \mathrm{HyperVolume}(A_D)$ for non-dominated cells.

The previously-discussed 2-D calculation scheme can be seen as a special case of this general decomposition, with two differences. The first is that the different quantities making up A are not calculated separately, and the second difference is that instead of using the reference point r to delimit A, a point is chosen which causes $\mathrm{HyperVolume}(\mathrm{DomSet}(P) \cap A_{\{1\}})$ and $\mathrm{HyperVolume}(\mathrm{DomSet}(P) \cap A_{\{2\}})$ to be 0. Replacing r by a point which causes the one-dimensional dominated hypervolumes to be 0 is always possible, but in the 2-D case this causes the constant $I_\emptyset$ to be the only quantity requiring the calculation of a hypervolume improvement, which is especially convenient. Chapter 5.3 will extend this technique to the 3-dimensional case, but first an example of the decomposition in the 3-dimensional case will be given in the section below, along with a demonstration of how this decomposition can be translated to the calculation of the partial expected hypervolume improvement for each cell.

5.2 Decomposition of the EHVI in the 3-D Case

The expected improvement of a point p with associated probability distribution function PDF in the 3-D case is defined as follows:

$$\text{EHVI}(\vec{p}) = \iiint\limits_{\mathbf{R}^3} HI(\vec{p}, P) \cdot PDF(\vec{p}) \mathrm{d}p_x \, \mathrm{d}p_y \, \mathrm{d}p_z$$

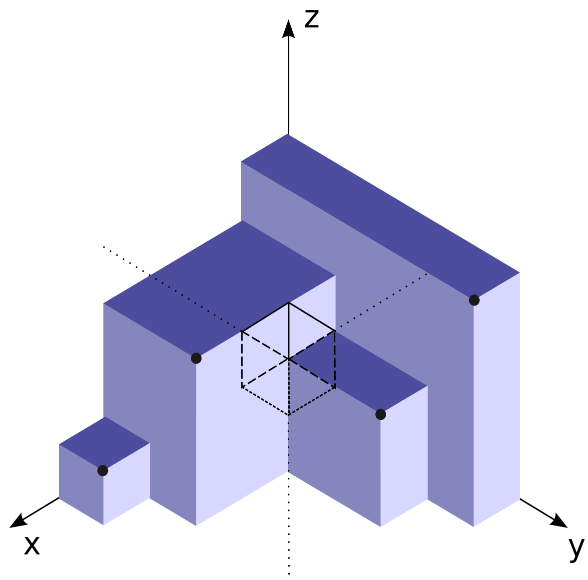

Figure 5.2: A visual representation of the integration region. The black dots are points in the Pareto approximation set P. The opaque volume underneath the points is the shape of the dominated hypervolume of P. The dashed box in the center is an example interval box. The dashed lines attached to this interval box represent the region within which the dominated hypervolume of P is relevant to the calculation of the partial expected hypervolume improvement for this interval box.

Here, PDF is a joint probability distribution function of three independent Gaussian probability distributions, one for each dimension. As in the 2-D case, we can calculate this integral by performing integration over a set of interval boxes covering the integration domain and summing up the results. The interval boxes will be delimited by the 2-D planes going through all coordinates of the points in P. See Figure 5.2 for a visualization. Each interval box will have its lower corner denoted by some vector $l \in \mathbf{R}^3$ and its upper corner denoted by some vector $u \in \mathbf{R}^3$. We get:

$$\text{EHVI}(\vec{p}) = \sum \left(\int\limits_{p_z = l_z}^{u_z} \int\limits_{p_y = l_y}^{u_y} \int\limits_{p_x = l_x}^{u_x} HI(p) \, \phi_x(p_x) \, \phi_y(p_y) \, \phi_z(p_z) \, \mathrm{d}p_x \, \mathrm{d}p_y \, \mathrm{d}p_z \right)$$

The hypervolume improvement, HI(p), is 0 when p is dominated by any point in P or when p does not dominate the reference point, so we need to do no further calculations for cells which are within the dominated hypervolume of P or whose lower corner stretches out to $-\infty$ in any dimension. For all cells where this is not the case, the hypervolume improvement of points within that cell can be decomposed as described in Section 5.1. We will denote these quantities by I_C where C is a subset of $\{x, y, z\}$ denoting the dimensions in which the quantity is variable.

$$\mathrm{HI}(p) = I_\emptyset + I_x + I_y + I_z + I_{xy} + I_{xz} + I_{yz} + I_{xyz}$$

$I_\emptyset$ is equal to the hypervolume improvement of l. Every point that dominates or is equal to l will also dominate this hypervolume, and therefore $\mathrm{HI}(l)$ is part of $\mathrm{HI}(p)$ for all p in the cell with the lower corner l. It can be calculated as the hypervolume dominated by l and not by any point in P:

$$I_\emptyset = \mathrm{Vol}\left(\left[\begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix}, \begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix}\right] \setminus \mathrm{DomSet}(P)\right)$$

I_x is the part of $\mathrm{HI}(p)$ which is variable in the x dimension and constant in the y and z dimensions. It is the hypervolume dominated by the point $\{p_x, l_y, l_z\}$ and not by l or any point in P. The hypervolume dominated by $\{p_x, l_y, l_z\}$ and not by l is formed by the box with upper corner $\{p_x, l_y, l_z\}$ and lower corner $\{l_x, r_y, r_z\}$. Because this volume spans no more than the length of the cell in the x dimension and points which determine the shape of the dominated hypervolume of P only lie on the 2-dimensional planes which delimit interval boxes, the size of the 3-D dominated volume of P increases linearly with x. This allows I_x to be defined as a constant 2-D hypervolume improvement in the y, z plane which is multiplied by a variable difference in the x plane, given by $p_x - l_x$, to compute the 3-D hypervolume improvement. The 2-D polyhedron with which to calculate this 2-D hypervolume improvement is a slice of the 3-D dominated hypervolume where x falls within the cell, and can be determined by taking the y, z coordinates of all points in P with $x > l_x$, as follows:

$$I_x = (p_x - l_x) \cdot \mathrm{Area}\left(\left[\begin{pmatrix} r_y \\ r_z \end{pmatrix}, \begin{pmatrix} l_y \\ l_z \end{pmatrix}\right] \setminus \mathrm{DomSet}\left(\pi_{yz}\left(\sigma_{x>l_x}(P)\right)\right)\right)$$

Here, π and σ are respectively the projection and selection operator from relational algebra, where P is interpreted as a ternary relation.

I_y and I_z are defined analogous to I_x.

I_{xy} is the part of $\mathrm{HI}(p)$ which is variable in the x and y dimensions but constant in the z dimension. It is the hypervolume dominated by $\{p_x, p_y, l_z\}$ and not by any point in P, $\{p_x, l_y, l_z\}$, $\{l_x, p_y, l_z\}$ or l. It is formed by the 3-D rectangle with upper corner $\{p_x, p_y, l_z\}$ and lower corner $\{l_x, l_y, r_z\}$, from which the volume dominated by P is then subtracted. Again, because this volume spans only the length of the cell in the x and y dimensions, it increases linearly in those dimensions and can be seen as a 2-D area defined by the distance between p_x and l_x and p_y and l_y multiplied by a one-dimensional hypervolume improvement in the z dimension:

$$I_{xy} = (p_x - l_x)(p_y - l_y) \cdot \text{Length}\left([r_z, l_z] \setminus \text{DomSet}\left(\pi_z\left(\sigma_{x>l_x, y>l_y}(P)\right)\right)\right)$$

A one-dimensional hypervolume improvement is simply a regular improvement, which is defined as the new value minus the highest existing value, or zero if this difference is smaller than zero. This allows the formula I_{xy} to be written more simply as:

$$I_{xy} = (p_x - l_x)(p_y - l_y)(l_z - \text{Max}\left(\{r_z\} \cup \pi_z\left(\sigma_{x>l_x, y>l_y}(P)\right)\right))$$

I_{xz} and I_{yz} are defined analogous to I_{xy}.

I_{xyz} is the part of $\text{HI}(p)$ which is variable in all three coordinates. Intuitively, this describes the part of $\text{HI}(p)$ which falls fully within the interval box. No part of the cell is dominated by a point in P, and therefore this partial hypervolume improvement can be calculated as the volume of the box which has l as its lower corner and p as its upper corner:

$$I_{xyz} = (p_x - l_x)(p_y - l_y)(p_z - l_z)$$

Now that we have decomposed the region that determines the hypervolume improvement, we can decompose the calculation of the partial integrals as well. If we refer to the constant part of each of the quantities described above as I_C^{const}, we have to compute the following for each interval box with a non-zero contribution:

$$\begin{aligned}\int_{p_z=l_z}^{u_z} \int_{p_y=l_y}^{u_y} \int_{p_x=l_x}^{u_x} & (I_\emptyset^{const} + I_x^{const}(p_x - l_x) + \dots \\ & + I_{xy}^{const}(p_x - l_x)(p_y - l_y) + \dots \\ & + (p_x - l_x)(p_y - l_y)(p_z - l_z)) \\ & \cdot \phi_x(p_x)\, \phi_y(p_y)\, \phi_y(p_y)\, \mathrm{d}p_x\, \mathrm{d}p_y\, \mathrm{d}p_z\end{aligned}$$

The constant quantities are as follows. $I_\emptyset$ is completely constant, therefore:

$$I_\emptyset^{const} = \text{Vol}\left(\left[\begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix}, \begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix}\right] \setminus \text{DomSet}(P)\right)$$

For I_x, I_y and I_z, the constant part is a 2-dimensional hypervolume improvement. I_x^{const} is defined as follows, with the other two constants defined analogously:

$$I_x^{const} = \text{Area}\left(\left[\begin{pmatrix} r_y \\ r_z \end{pmatrix}, \begin{pmatrix} l_y \\ l_z \end{pmatrix}\right] \setminus \text{DomSet}\left(\pi_{yz}\left(\sigma_{x>l_x}(P)\right)\right)\right)$$

The constant part of I_{xy}, I_{xz} and I_{yz} is a one-dimensional improvement:

$$I_{xy}^{const} = l_z - \text{Max}\left(\{r_z\} \cup \pi_z\left(\sigma_{x>l_x, y>l_y}(P)\right)\right)$$

I_{xyz} has no constant part, which is why I_{xyz}^{const} is omitted from the formula.

The sum rule allows us to decompose our integral into a sum of smaller integrals, each of which represents the expected improvement associated with a single quantity from the decomposition of HI(p). We will refer to these partial integrals as EI_C where $C \subseteq \{x, y, z\}$.

$$\sum_{C \subseteq \{x,y,z\}} EI = EI_\emptyset + EI_x + \ldots + EI_{xy} + \ldots + EI_{xyz}$$

$$EI_\emptyset = \int_{p_z=l_z}^{u_z} \int_{p_y=l_y}^{u_y} \int_{p_x=l_x}^{u_x} I_\emptyset^{const} \cdot \phi_x(p_x)\, \phi_y(p_y)\, \phi_z(p_z)\, \mathrm{d}p_x\, \mathrm{d}p_y\, \mathrm{d}p_z$$

$$EI_x = \int_{p_z=l_z}^{u_z} \int_{p_y=l_y}^{u_y} \int_{p_x=l_x}^{u_x} I_x^{const}(p_x - l_x) \cdot \phi_x(p_x)\, \phi_y(p_y)\, \phi_z(p_z)\, \mathrm{d}p_x\, \mathrm{d}p_y\, \mathrm{d}p_z$$

$$\ldots$$

$$EI_{xy} = \int_{p_z=l_z}^{u_z} \int_{p_y=l_y}^{u_y} \int_{p_x=l_x}^{u_x} I_{xy}^{const}(p_x - l_x)(p_y - l_y) \cdot \phi_x(p_x)\, \phi_y(p_y)\, \phi_z(p_z)\, \mathrm{d}p_x\, \mathrm{d}p_y\, \mathrm{d}p_z$$

$$\ldots$$

$$EI_{xyz} = \int_{p_z=l_z}^{u_z} \int_{p_y=l_y}^{u_y} \int_{p_x=l_x}^{u_x} (p_x - l_x)(p_y - l_y)(p_z - l_z) \cdot \phi_x(p_x)\, \phi_y(p_y)\, \phi_z(p_z)\, \mathrm{d}p_x\, \mathrm{d}p_y\, \mathrm{d}p_z$$

Fubini's theorem [17] states that iterated integration, performed in any order, can be used to calculate a multiple integral under the condition that the multiple integral is absolutely convergent. The integrals we are considering here all converge to finite numbers, so we can safely use iterated integration.

In $EI_\emptyset$, factoring out the contribution of $I_\emptyset^{const}$ results in an integral which consists solely of a Gaussian PDF over a 3-D box. We can calculate this integral by using the formula for the Gaussian cumulative probability distribution (which requires only the Gaussian error function, *erf*) and subtracting the cumulative probability at the lower corner of the cell from the cumulative probability at the upper corner of the cell. This allows $EI_\emptyset$ to be calculated without numerical integration if we are willing to accept *erf* as a closed-form expression:

$$EI_\emptyset = I_\emptyset^{const} \cdot \prod_{c\in\{x,y,z\}} (\Phi_c(u_c) - \Phi_c(l_c))$$

For EI_x, we can factor ϕ_y and ϕ_z out of the integral and simply multiply everything with the cumulative distribution, because everything apart from the Gaussian PDF is constant in those integration variables. This leaves a one-dimensional integral containing the integration variable ϕ_x:

$$EI_x = \left(\int_{p_x=l_x}^{u_x} (p_x - l_x)\phi_x(p_x)\,\mathrm{d}p_x \right) \cdot \prod_{c\in\{y,z\}} (\Phi_c(u_c) - \Phi_c(l_c)) \cdot I_x^{const}$$

The integral between the large parentheses has the proper form to be calculated using the ψ function, in the following way:

$$\int_{p_x=l_x}^{u_x} (p_x - l_x)\phi_x(p_x)\,\mathrm{d}p_x = \psi(l_x, l_x, \mu_x, \sigma_x) - \psi(l_x, u_x, \mu_x, \sigma_x)$$

This gives the following formula:

$$EI_x = I_x^{const} \cdot \prod_{c\in\{y,z\}} (\Phi_c(u_c) - \Phi_c(l_c)) \cdot (\psi(l_x, l_x, \mu_x, \sigma_x) - \psi(l_x, u_x, \mu_x, \sigma_x))$$

EI_y and EI_z are defined analogously.

To calculate EI_{xy}, we can place the constant factor I_{xy}^{const} as well as the probability distribution for z outside of the integral right away. In effect, we are integrating in the z dimension, and then placing the result outside of the integral because it is constant in both of the other dimensions.

$$EI_{xy} = I_{xy}^{const}(\Phi_z(u_z) - \Phi_z(l_z)) \cdot \int_{p_y=l_y}^{u_y} \int_{p_x=l_x}^{u_x} (p_x - l_x)(p_y - l_y) \cdot \phi_x(p_x)\,\phi_y(p_y)\,\mathrm{d}p_x\,\mathrm{d}p_y$$

At first this formula appears complex, but it becomes easier to calculate because the parts are not dependent on each other. We can see $(p_x - l_x)$ as a function which is constant in y and variable in x, and $(p_y - l_y)$ as a function which is constant in x and variable in y. This results in a product of two integrals which can be calculated separately:

$$\begin{aligned}
&\int_{p_y=l_y}^{u_y} \left(\int_{p_x=l_x}^{u_x} (p_x - l_x)(p_y - l_y) \cdot \phi_x(p_x)\, \phi_y(p_y)\, \mathrm{d}p_x \right) \mathrm{d}p_y \\
&= \int_{p_y=l_y}^{u_y} (p_y - l_y) \left(\int_{p_x=l_x}^{u_x} (p_x - l_x) \cdot \phi_x(p_x) \mathrm{d}p_x \right) \phi_y(p_y)\, \mathrm{d}p_y \\
&= \int_{p_y=l_y}^{u_y} (p_y - l_y)\phi_y(p_y)\, \mathrm{d}p_y \cdot \int_{p_x=l_x}^{u_x} (p_x - l_x) \cdot \phi_x(p_x) \mathrm{d}p_x
\end{aligned}$$

Both of the integrals have the form of a calculation of the partial one-dimensional expected improvement. We can therefore use ψ to calculate them. We get the following formula:

$$(\psi(l_x, l_x, \mu_x, \sigma_x) - \psi(l_x, u_x, \mu_x, \sigma_x)) \cdot (\psi(l_y, l_y, \mu_y, \sigma_y) - \psi(l_y, u_y, \mu_y, \sigma_y))$$

Which means that our complete formula for EI_{xy} will look like this:

$$\begin{aligned}
EI_{xy} = &I_{xy}^{const} \\
&\cdot (\Phi_z(u_z) - \Phi_z(l_z)) \\
&\cdot (\psi(l_x, l_x, \mu_x, \sigma_x) - \psi(l_x, u_x, \mu_x, \sigma_x)) \\
&\cdot (\psi(l_y, l_y, \mu_y, \sigma_y) - \psi(l_y, u_y, \mu_y, \sigma_y))
\end{aligned}$$

EI_{xz} and EI_{yz} are defined analogously.

The formula for EI_{xyz} is easily derived using the same method that was used for deriving the formula for EI_{xy} and is even simpler. Instead of a product of two expected improvements, a constant and a Gaussian cumulative probability distribution, it is a product of three expected improvements:

$$\prod_{c \in \{x,y,z\}} (\psi(l_c, l_c, \mu_c, \sigma_c) - \psi(l_c, u_c, \mu_c, \sigma_c))$$

The amount of calculations required for the decomposition described above is quite large. One thing that might be noticed when looking at the different formulas, is that partial calculations can be reused. In particular, only six calls to the ψ function need to be made. If efficient calculation is a goal, however, then larger improvements can be reached. This will be explored in the next section.

5.3 Calculation of the 3-D EHVI

Consider that, in the 2-D case, we are able to calculate the hypervolume by integrating over a box bounded by the dominated hypervolume and subtracting a correction term S_{minus}. We can do something similar in the 3-D case. Recall that we decomposed $\mathrm{HI}(p)$ as follows:

$$\mathrm{HI}(p) = I_\emptyset + I_x + I_y + I_z + I_{xy} + I_{xz} + I_{yz} + I_{xyz}$$

Together, they form the volume of $\left(\left[\begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix}, \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}\right] \setminus \mathrm{DomSet}(P)\right)$. Instead of writing $HI(p)$ as a sum of hypervolume improvements, we can also write it as a single rectangular volume from which a dominated hypervolume is subtracted:

$$\mathrm{HI}(p) = \mathrm{Vol}\left(\left[\begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix}, \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}\right]\right) - \mathrm{Vol}\left(\mathrm{DomSet}(P) \cap \left[\begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix}, \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}\right]\right)$$

We can then decompose the calculation of the dominated hypervolume instead of the calculation of the hypervolume improvement. In the following decomposition of the total subtracted dominated hypervolume S^-, each part S^-_C is equal to the subtracted dominated hypervolume of I_C. When p is within the integration cell bounded from below by l, we get the following:

$$\begin{aligned} S^- &= S^-_\emptyset + S^-_x + S^-_y + S^-_z + S^-_{xy} + S^-_{xz} + S^-_{yz} \\ &= \mathrm{Vol}\left(\mathrm{DomSet}(P) \cap \left[\begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix}, \begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix}\right]\right) \\ &+ (p_x - l_x) \cdot \mathrm{Area}\left(\mathrm{DomSet}\left(\pi_{yz}\left(\sigma_{x>l_x}(P)\right)\right) \cap \left[\begin{pmatrix} r_y \\ r_z \end{pmatrix}, \begin{pmatrix} p_y \\ p_z \end{pmatrix}\right]\right) \\ &+ \ldots \\ &+ (p_x - l_x) \cdot (p_y - l_y) \cdot \big(\mathrm{Max}(r_z, \pi_z(\sigma_{x>l_x, y>l_y}(P))) - r_z\big) \\ &+ \ldots \end{aligned}$$

The first thing to note is that if $r_z \geq \mathrm{Max}(r_z, \pi_z(\sigma_{x>l_x,y>l_y}(P)))$, $S^{-}_{xy} = 0$. The analogous cases are true for S^{-}_{xz} and S^{-}_{yz}, allowing us to define a point v for which, if $r = v$, all three quantities are 0:

$$v = \left[\begin{pmatrix} \mathrm{Max}(r_x, \pi_x(\sigma_{y>l_y,z>l_z}(P))) \\ \mathrm{Max}(r_y, \pi_y(\sigma_{x>l_x,z>l_z}(P))) \\ \mathrm{Max}(r_z, \pi_z(\sigma_{x>l_x,y>l_y}(P))) \end{pmatrix}\right]$$

The bounding box bounded by v from below and p from above contains the entire volume of $HI(p)$. This allows us to use v in place of r and rewrite our initial equation in a way that reduces the number of components from 8 to 5:

$$\begin{aligned}
\mathrm{HI}(p) = \mathrm{Vol}&\left(\left[\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}, \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}\right]\right) \\
&- \mathrm{Vol}\left(\mathrm{DomSet}(P) \cap \left[\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}, \begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix}\right]\right) \\
&- (p_x - l_x) \cdot \mathrm{Area}\left(DomSet\left(\pi_{yz}\left(\sigma_{x>l_x}(P)\right)\right) \cap \left[\begin{pmatrix} v_y \\ v_z \end{pmatrix}, \begin{pmatrix} l_y \\ l_z \end{pmatrix}\right]\right) \\
&- (p_y - l_y) \cdot \mathrm{Area}\left(DomSet\left(\pi_{xz}\left(\sigma_{y>l_y}(P)\right)\right) \cap \left[\begin{pmatrix} v_x \\ v_z \end{pmatrix}, \begin{pmatrix} l_x \\ l_z \end{pmatrix}\right]\right) \\
&- (p_z - l_z) \cdot \mathrm{Area}\left(DomSet\left(\pi_{xy}\left(\sigma_{z>l_z}(P)\right)\right) \cap \left[\begin{pmatrix} v_x \\ v_y \end{pmatrix}, \begin{pmatrix} l_x \\ l_y \end{pmatrix}\right]\right)
\end{aligned}$$

The integral corresponding to $\mathrm{Vol}\left(\left[\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}, \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}\right]\right)$ is the only component in this equation which is variable in more than one dimension, but since it is a rectangular volume, it is simply a product of one-dimensional improvements:

$$\prod_{c \in \{x,y,z\}} \left(\psi(v_c, l_c, \mu_c, \sigma_c) - \psi(v_c, u_c, \mu_c, \sigma_c)\right)$$

$S^{-}_{\emptyset}$ is a constant. Even without examining the corresponding integral it is clear that it only needs to be multiplied with the probability that a given point is within the cell. The formula for calculating this correction term is:

$$S^{-}_{\emptyset} \cdot \prod_{c \in \{x,y,z\}} \left(\Phi_c(u_c) - \Phi_c(l_c)\right)$$

S_x^-, S_y^-, and S_z^- are not constants, but they are each linearly related to only one coordinate of p. We will look at S_x^- as an example:

The constant part of S_x^- is $\text{Area}\left(\text{DomSet}\left(\pi_{yz}\left(\sigma_{x>l_x}(P)\right)\right) \cap \left[\begin{pmatrix} v_y \\ v_z \end{pmatrix}, \begin{pmatrix} l_y \\ l_z \end{pmatrix}\right]\right)$. This has to be multiplied by $(p_x - l_x)$. The expected value of S_x^- is therefore equal to a constant multiplied by the partial expected improvement of p_x over the interval $[l_x, u_x)$. This is given by:

$$\int\limits_{p_x=l_x}^{u_x} (p_x - l_x)\phi_x(p_x)\,\mathrm{d}p_x = \psi(l_x, l_x, \mu_x, \sigma_x) - \psi(l_x, u_x, \mu_x, \sigma_x)$$

Using a new call to ψ to calculate this term is not necessary. We can use the fact that ψ represents the function of a one-dimensional expected improvement over a certain range bounded from below. The partial expected improvement for the region below the lower cell bound l is a constant term multiplied by the chance of being within the cell's range, which is captured in the equation below:

$$\psi(v_c, l_c, \mu_c, \sigma_c) - \psi(v_c, u_c, \mu_c, \sigma_c) = \psi(l_c, l_c, \mu_c, \sigma_c) - \psi(l_c, u_c, \mu_c, \sigma_c) + (\Phi_c(u_c) - \Phi_c(l_c)) \cdot (l_c - v_c)$$

Both $(\Phi_c(u_c) - \Phi_c(l_c)) \cdot (l_c - v_c)$ and $\psi(v_c, l_c, \mu_c, \sigma_c) - \psi(v_c, u_c, \mu_c, \sigma_c)$ were calculated earlier, so we can reuse them to easily find $\psi(l_c, l_c, \mu_c, \sigma_c) - \psi(l_c, u_c, \mu_c, \sigma_c)$.

This means that the formula for calculating the partial expected hypervolume improvement of a cell will look like this if the cell is not dominated:

$$\begin{aligned} &\text{Let} \quad \Delta\psi_c = \psi(v_c, l_c, \mu_c, \sigma_c) - \psi(v_c, u_c, \mu_c, \sigma_c),\ c \in \{x, y, z\} \\ &\text{and} \quad \Delta\phi_c = \Phi_c(u_c) - \Phi_c(l_c),\ c \in \{x, y, z\} \end{aligned}$$

$$
\begin{aligned}
EI = &\prod_{c\in\{x,y,z\}} \Delta\psi_c \\
&- Vol\left(DomSet(P) \cap \left[\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}, \begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix}\right]\right) \cdot \prod_{c\in\{x,y,z\}} \Delta\phi_c \\
&- (\Delta\psi_z - \Delta\phi_z \cdot (p_z - v_z)) \cdot Area\left(DomSet\left(\pi_{yz}\left(\sigma_{x>l_x}(P)\right)\right) \cap \left[\begin{pmatrix} v_y \\ v_z \end{pmatrix}, \begin{pmatrix} l_y \\ l_z \end{pmatrix}\right]\right) \cdot \prod_{c\in\{x,y\}} \Delta\phi_c \\
&- (\Delta\psi_y - \Delta\phi_y \cdot (p_y - v_y)) \cdot Area\left(DomSet\left(\pi_{xz}\left(\sigma_{y>l_y}(P)\right)\right) \cap \left[\begin{pmatrix} v_x \\ v_z \end{pmatrix}, \begin{pmatrix} l_x \\ l_z \end{pmatrix}\right]\right) \cdot \prod_{c\in\{x,z\}} \Delta\phi_c \\
&- (\Delta\psi_x - \Delta\phi_x \cdot (p_x - v_x)) \cdot Area\left(DomSet\left(\pi_{xy}\left(\sigma_{z>l_z}(P)\right)\right) \cap \left[\begin{pmatrix} v_x \\ v_y \end{pmatrix}, \begin{pmatrix} l_x \\ l_y \end{pmatrix}\right]\right) \cdot \prod_{c\in\{y,z\}} \Delta\phi_c
\end{aligned}
$$

And it will be 0 otherwise.

5.4 Simple Higher-Dimensional EHVI Calculations

Although we are currently decomposing our integral into different quantities in order to calculate it, we can also calculate the sum of these quantities using a single dominated hypervolume calculation. This section will give the general formula for doing so. Recall how we decomposed the calculation of the hypervolume improvement in Chapter 5.1:

$$
\begin{aligned}
&\mathrm{HI}(p) = \sum_{C\subseteq D} I_C \\
&I_C = \mathrm{HyperVolume}(A_C) - \mathrm{HyperVolume}(\mathrm{DomSet}(P) \cap A_C)
\end{aligned}
$$

This sum can be rearranged to the following:

$$
\sum_{C\subseteq D} \mathrm{HyperVolume}(A_C) - \sum_{C\subseteq D} \mathrm{HyperVolume}(\mathrm{DomSet}(P) \cap A_C)
$$

Since the quantities A_C sum to a generalized rectangular volume, we could just as readily calculate the total volume of A directly. This is what we did in Chapter 5.3, where the correction terms $\mathrm{HyperVolume}(\mathrm{DomSet}(P) \cap A_C)$ were still calculated separately. The equality $\sum_{C\subseteq D} \mathrm{HyperVolume}(\mathrm{DomSet}(P) \cap A_C) = \mathrm{HyperVolume}(\mathrm{DomSet}(P) \cap A)$ follows directly from the definition of S_C, but we initially decomposed this calculation to solve the problem of calculating the corresponding integral.

We have determined that each partial quantity HyperVolume(DomSet(P) $\cap A_C$) depends linearly on the dimensions which its corresponding volume A_C depends on, and is constant in the same dimensions in which A_C is constant. This is true as well when HyperVolume(DomSet(P) $\cap A$) is first calculated, and then split into the various volumes representing DomSet(P) $\cap A_C$. Because of this, we can calculate an m-dimensional EHVI using only a single m-dimensional hypervolume calculation per cell. We need to calculate the hypervolume improvement of each cell's *center of mass*, $\bar{p}$.

$$\bar{p}_d = \frac{\int_{p_d=l_d}^{u_d} p_d \cdot \phi_d(p_d)\mathrm{d}p}{\Phi_d(u_d) - \Phi_d(l_d)}$$

The integral can be calculated as if it is an expected improvement where the incumbent solution is 0. However, we already need to calculate $\psi(r_d, l_d, \mu_d, \sigma_d) - \psi(l_d, u_d, \mu_d, \sigma_d)$ to determine the volume of A, and the following equation holds:

$$\frac{\psi(0, l_d, \mu_d, \sigma_d) - \psi(0, u_d, \mu_d, \sigma_d)}{\Phi_d(u_d) - \Phi_d(l_d)} = \frac{\psi(r_d, l_d, \mu_d, \sigma_d) - \psi(r_d, u_d, \mu_d, \sigma_d)}{\Phi_d(u_d) - \Phi_d(l_d)} + r_d$$

Dividing a partial expected improvement over a range $[l_d, u_d)$ by the chance of being in that range (given by $\Phi_d(u_d) - \Phi_d(l_d)$) gives the expected improvement of points which are known to lie within that range. Adding the value of r_d gives the expected dth coordinate of a point in the objective space bounded from below by r.

This means that the general formula for calculating the partial expected improvement in a cell is the following if the cell is not dominated:

$$EI = \prod_{d \in D} \left(\psi(r_d, l_d, \mu_d, \sigma_d) - \psi(r_d, u_d, \mu_d, \sigma_d)\right) - S^- \cdot \prod_{d \in D} \left(\Phi_d(u_d) - \Phi_d(l_d)\right)$$

$$S^- = \mathrm{HyperVolume}\left(\mathrm{DomSet}(P) \cap \left[\begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{pmatrix}, \begin{pmatrix} \bar{p}_1 \\ \bar{p}_2 \\ \vdots \\ \bar{p}_m \end{pmatrix}\right]\right)$$

$$\bar{p}_d = r_d + \frac{\psi(r_d, l_d, \mu_d, \sigma_d) - \psi(r_d, u_d, \mu_d, \sigma_d)}{\Phi_d(u_d) - \Phi_d(l_d)}$$

And 0 otherwise.

5.5 Complexity

Any algorithm which iterates over all grid cells described in Chapter 5 will have a time complexity of $\Omega(n^m)$. This is further increased by the complexity of the calculations within each grid cell. The algorithm of Chapter 5.4 requires an m-dimensional hypervolume to be calculated for each cell that is not dominated. Calculating a 3-dimensional hypervolume can be done in $O(n \log n)$, which results in a time complexity of $O(n^4 \log n)$. However, as will be shown in Chapter 6, constant-time calculations within each grid cell are possible with $O(n^3)$ total preparation time, resulting in an algorithm of complexity $O(n^3)$. Similar $O(n^m)$ algorithms are conjectured to exist for $m > 3$.

One important thing to note, is that the expected hypervolume improvements for multiple individuals can be calculated at the same time without having to perform the hypervolume calculations more than once when using the decomposition described in Chapter 5.2 or 5.3, because the hypervolume calculations are not dependent on the mean and standard deviation of the probability distribution. The algorithm described in Chapter 5.4 does not have this advantage.

Chapter 6

$O(n^3)$-time 3-D EHVI Calculations

In Chapter 4 we showed that calculating the 2-D expected hypervolume improvement is possible with time complexity $O(n^2)$. Although the algorithm described in that section made use of characteristics of a 2-D Pareto approximation set which are not present in higher dimensions, this section will show that there is also a way to calculate the 3-D EHVI with time complexity $O(n^3)$. In other words: the calculations necessary for computing the partial expected hypervolume improvement of each grid cell will be performed in constant time. The trade-off is that we will need $O(n^2)$ extra memory.

The only calculations which have a complexity higher than constant time are the dominated hypervolume calculations. If we use the simple algorithm described in Chapter 5.4, we only need to perform a single 3-dimensional hypervolume calculation to find the correction term that we need. However, we will start out with the algorithm described in Chapter 5.3 (*without* replacing r by v), because it lends itself better to the re-use of old hypervolume calculations. Three sets of correction terms are needed to calculate the partial expected hypervolume improvement of a cell:

- $S^-_\emptyset$, a constant correction term which requires a three-dimensional hypervolume calculation.
- S^-_x, S^-_y and S^-_z, which each require a two-dimensional hypervolume calculation. We will call the 2-D areas used in the calculation of these correction terms *xslice*, *yslice* and *zslice*, respectively.
- S^-_{xy}, S^-_{xz} and S^-_{yz}, which requires a 'one-dimensional' hypervolume calculation.

Instead of calculating these correction terms afresh for each cell, it is possible to perform all necessary hypervolume calculations in only $O(n^3)$ time total. The first step is to create a data structure which allows us to see if a cell is dominated in $O(1)$ time. This can simply be a two-dimensional array holding the highest value of z for which the cell is dominated, which we shall call H_z. A simple way to fill this array is to iterate over all points $q \in P$ in order of ascending z value, setting the array value $H_z(a_1, a_2)$ to z if q dominates the lower corner of $C(a_1, a_2, 0)$. This only needs to be done once, so the $O(n^3)$ time complexity is no problem. Figure 6.1 shows an example.

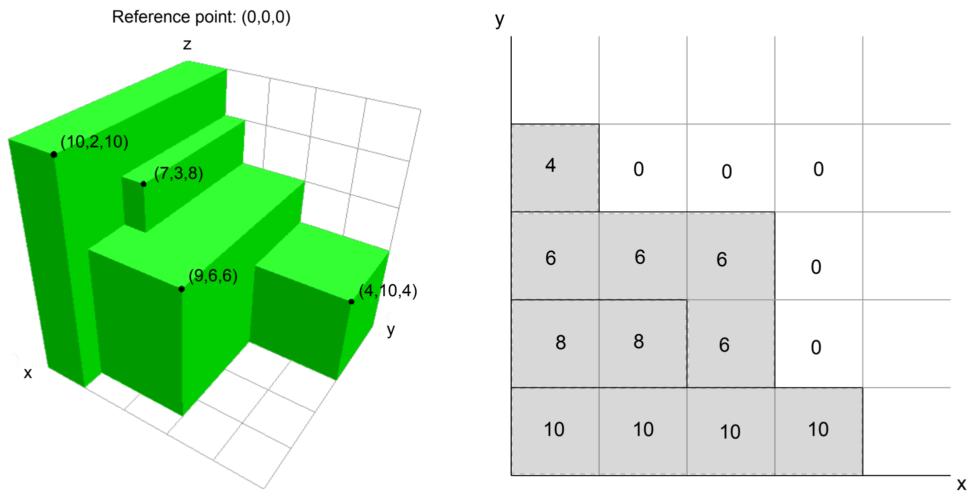

Figure 6.1: Example height array H_z for a population consisting of 4 points, which is visualized on the left. Cells on the outermost edge of the integration area (which stretch out to ∞ in some dimension) are always non-dominated.

Besides containing information that allows constant-time evaluation of whether a cell is dominated, the value of S^-_{xy} for a cell $C(a_1, a_2, a_3)$ that is not dominated is also given by $H_z(a_1, a_2)$. If we build two more height arrays H_x and H_y where we use the highest value of x and y instead of z, we can determine the results of all three of the one-dimensional hypervolume calculations in constant time during cell calculations.

Now, only the two-dimensional hypervolume calculations represented by $xslice$, $yslice$ and $zslice$, and the three-dimensional hypervolume calculation represented by $S^-_\emptyset$, still have a complexity greater than constant time. For notational simplicity, we have omitted their dependence on a particular cell from the notation until now, but in order to show the relations between correction terms of different cells, we will write '$S^-_\emptyset$ belonging to $C(a_1, a_2, a_3)$' as $C(a_1, a_2, a_3).S^-_\emptyset$, and likewise for the two-dimensional hypervolumes.

The value of $S^-_\emptyset$ is related to the values of $xslice$, $yslice$ and $zslice$ in the following way:

- $C(a_1, a_2, a_3).xslice = \frac{C(a_1+1, a_2, a_3).S^-_\emptyset - C(a_1, a_2, a_3).S^-_\emptyset}{u.x - l.x}$
- $C(a_1, a_2, a_3).yslice = \frac{C(a_1, a_2+1, a_3).S^-_\emptyset - C(a_1, a_2, a_3).S^-_\emptyset}{u.y - l.y}$
- $C(a_1, a_2, a_3).zslice = \frac{C(a_1, a_2, a_3+1).S^-_\emptyset - C(a_1, a_2, a_3).S^-_\emptyset}{u.z - l.z}$

With our height array H_z, we can calculate all values of $zslice$ for a given value of a_3 in $O(n^2)$ time. We can also calculate all values of $S^-_\emptyset$ for a given value of a_3 in $O(n^2)$ time, provided $a_3 = 0$ or we have both $S_\emptyset^-$ and *zslice* for the cells where a_3 is one lower. The details of these calculations will be given below. If we go through our cells in the right order (with a_3 starting at 0, incrementing it only after we have performed the calculations for all cells with a given value of a_3), we only need to update the values of *zslice* and $S_\emptyset^-$ n times, resulting in an algorithm for the full computation with complexity in $O(n^3)$. If we know the value of $S_\emptyset^-$ for all cells with a given value of a_3, we can use the formulas given above to calculate *xslice* and *yslice* in constant time whenever we need them, so we do not need to calculate these constants in advance.

The details of calculating *zslice* using the height array are as follows. We will iterate through the possible values of a_1 and a_2 in ascending order. We know that $zslice(a_1, a_2)$ is 0 if $a_1 = 0$ or $a_2 = 0$. If our height array shows that $C(a_1 - 1, a_2 - 1, a_3)$ is dominated, $zslice(a_1, a_2)$ is set equal to the area of the 2-D rectangle from its lower corner to $\{r_x, r_y\}$. Else, if that cell is not dominated, $zslice(a_1, a_2)$ is set equal to $zslice(a_1 - 1, a_2) + zslice(a_1, a_2 - 1) - zslice(a_1 - 1, a_2 - 1)$. The value of $zslice(a_1 - 1, a_2 - 1)$ is removed as this is the area which is overlapping, causing it to be added twice otherwise.

For an example, refer to Figure 6.2.

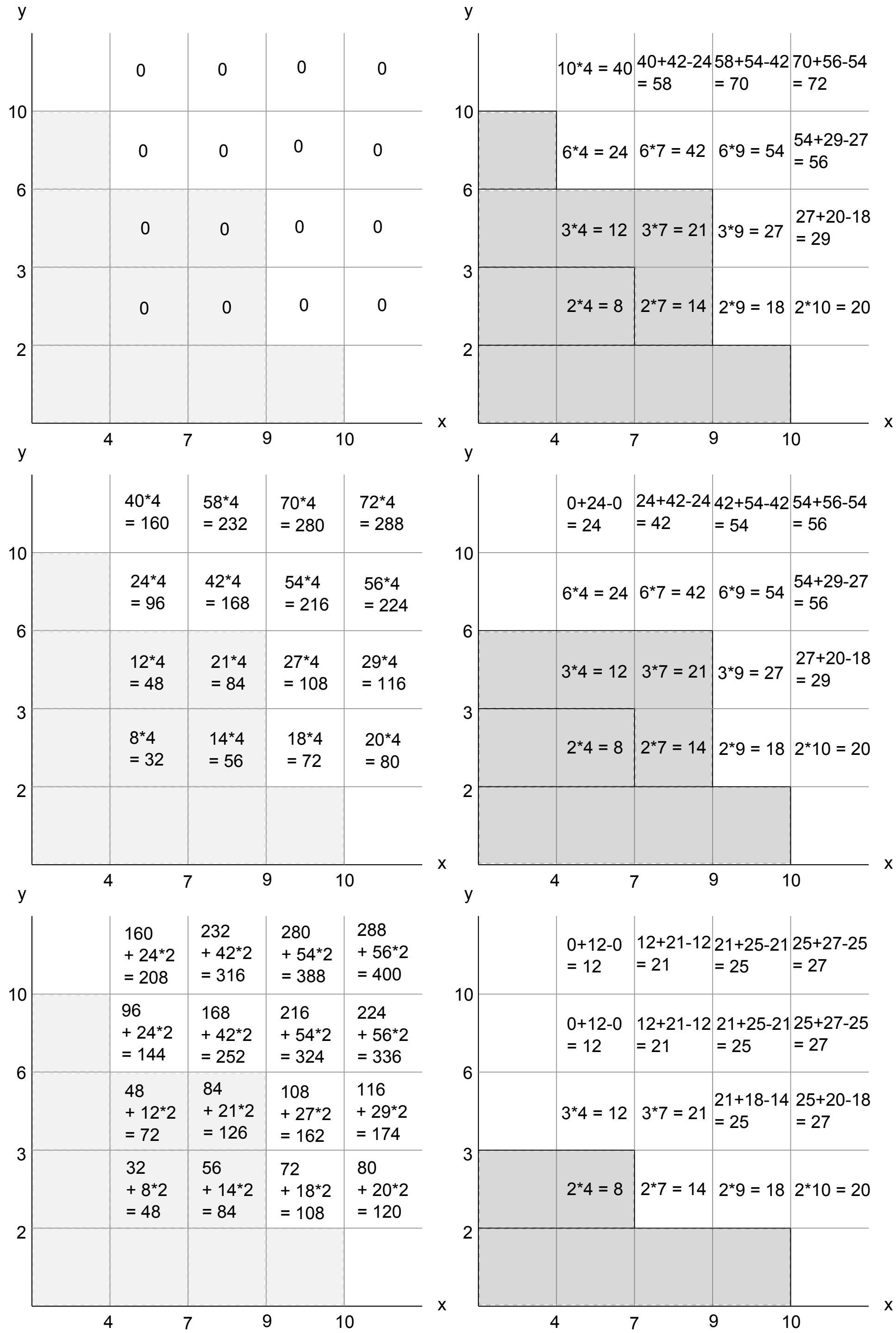

Figure 6.2: Some values of $zslice$ and S^{-} for the example shown in Figure 6.1, with $a_3 = 0$, 1 and 2, respectively. The x and y values of each cell's lower corner are shown on the axes. The grids with the values of S^{-} are on the left and the grids with the values of $zslice$ are on the right.

Chapter 7

Empirical Tests and Results

Five different implementations of a 3-D expected hypervolume improvement calculation algorithm were used throughout the following tests, referred to as the 8-term, 5-term, 2-term, slice-update and Monte Carlo schemes. The goal of comparing the exact calculation algorithms to a Monte Carlo scheme is twofold. First, by computing the Expected Hypervolume Improvement in different ways, the algorithms and their implementations will be thoroughly validated. Second, the time consumption of the algorithms will be compared. This is of particular interest because Monte Carlo schemes are often used as fast approximations to exact computations.

- The 8-term scheme is a direct implementation of the calculations described in Chapter 5.2.
- The 5-term scheme implements the slightly simplified calculations described in Chapter 5.3.
- The 2-term scheme implements the calculations described in Chapter 5.4.
- The slice-update scheme implements the algorithm described in Chapter 6.
- The Monte Carlo scheme uses Monte Carlo integration to give an approximation of the expected hypervolume improvement. Its random number generator uses the Box-Muller transform [19] in combination with the Mersenne Twister algorithm [18] (specifically, the 32-bit MT19937 variant from the C++ standard library, implemented in GCC) to generate normally distributed pseudo-random numbers. Due to the nature of Monte Carlo algorithms, it is impossible to get an exact answer out of this scheme. The expected error of Monte Carlo integration is related to the number of trials m by $\frac{1}{\sqrt{m}}$, which means that to make the estimate ten times more accurate, a hundred times more trials are required.

The implementations of ψ and the Gaussian cumulative distribution function were identical for all schemes, except for the Monte Carlo scheme where they were not used. The

2-D and 3-D hypervolume calculation functions were also identical between those schemes which used them. Standard C++ library functions were used for sorting and for the implementation of the Gaussian error function *erf*.

7.1 Monte Carlo Verification

As a verification of the correctness of the algorithms, the expected hypervolume improvements calculated by all schemes on several test problems were compared to each other and to the value which the Monte Carlo scheme converged towards.

The graph in Figure 7.1 shows the results of running the algorithms on a simple test problem. The population consisted of three points: $(1, 2, 3)$, $(2, 3, 1)$ and $(3, 1, 2)$. The reference point was set to $(0, 0, 0)$. The median vector for the Gaussian distribution was set to $(3, 3, 3)$, placing it right between cell borders, and the standard deviation was set to $(2, 2, 2)$. All non-Monte Carlo schemes gave exactly identical answers, which was likely due to the simplicity of the test case, because rounding errors in the floating-point calculations would have resulted in small differences otherwise. The Monte Carlo scheme was allowed to run for 100.000.000 iterations.

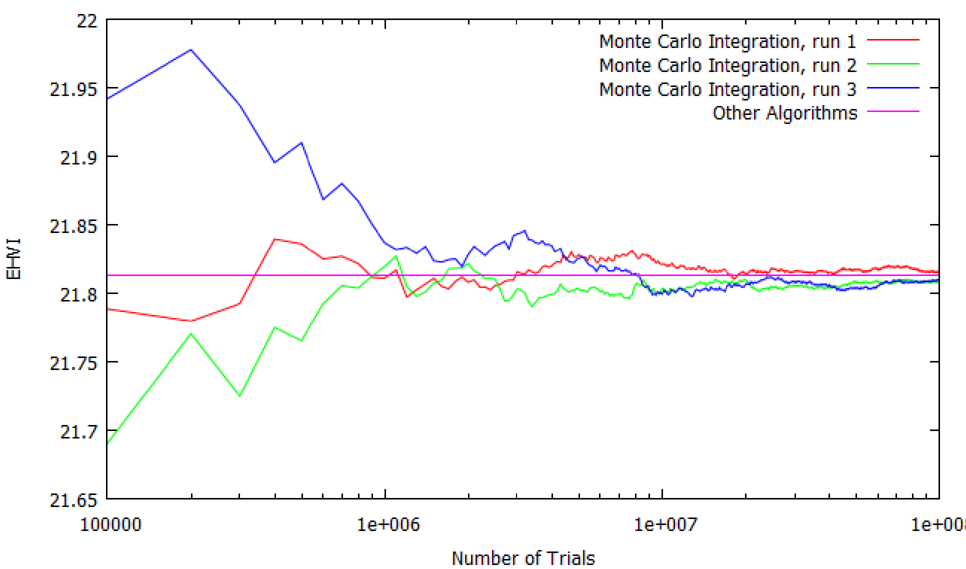

Figure 7.1: Logarithmic-scale graph of the convergence of Monte Carlo integration. The answer was measured every 100.000 iterations.

Figure 7.2 shows the results of running the algorithm on a few more complex populations. The first consists of 30 points, some of which had identical values to another point in the population in one of their dimensions (creating cells of size 0). The second consists of 100 points with a bias towards one area of the search space. The results of all non-Monte Carlo schemes on these two test problems were identical to 15 and 14 digits, respectively. The double-precision floating numbers which were used in the implementations are accurate to approximately the 15th decimal, so the answers can safely be considered identical.

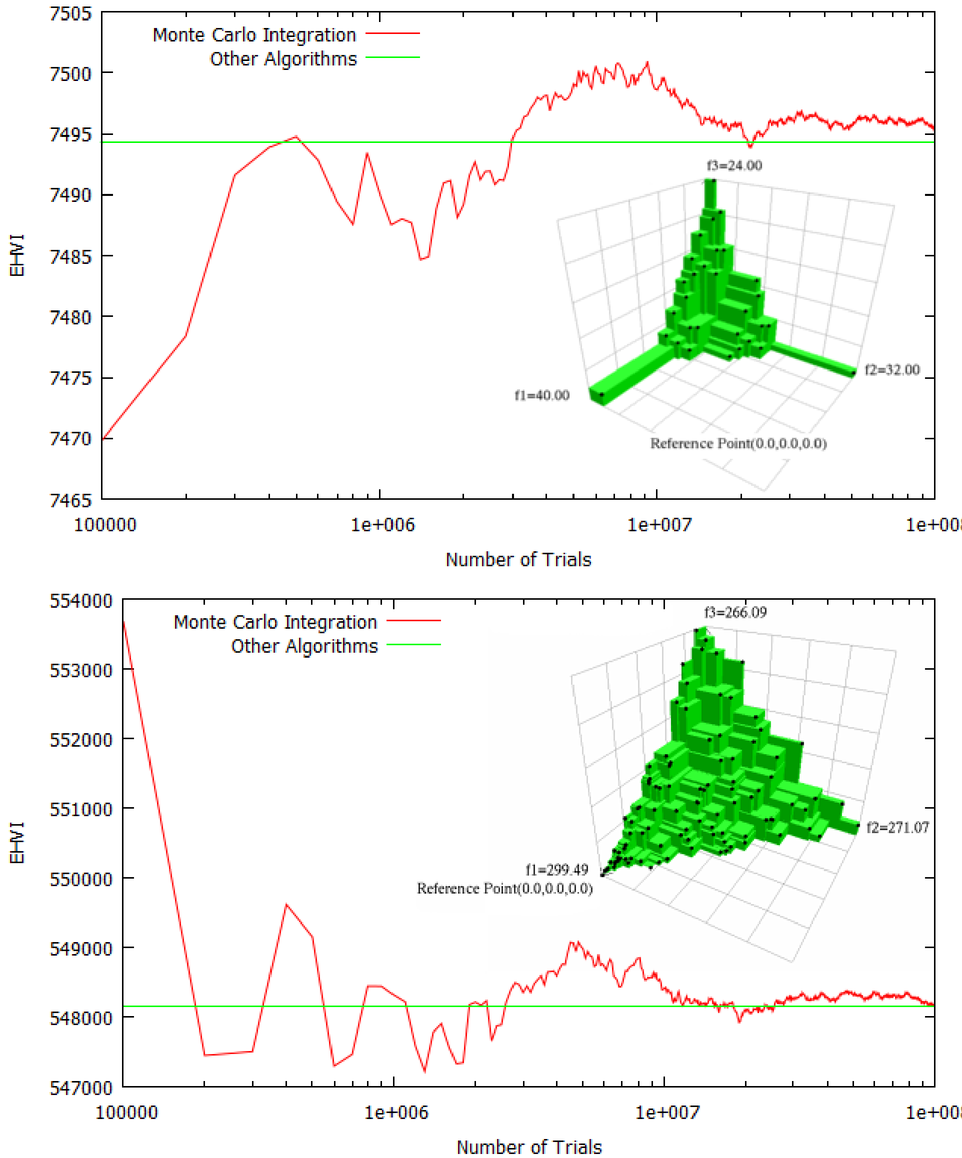

Figure 7.2: Two logarithmic-scale graphs showing the convergence of Monte Carlo integration, along with visualizations of the Pareto approximation sets.

The convergence of the Monte Carlo integration, as well as the near-identical answers generated by the different approaches towards calculating the expected hypervolume improvement, both support the validity of the calculations described in this thesis.

7.2 Empirical Performance

To test the empirical performance of the exact calculation schemes, they were tested on mutually non-dominated populations of varying sizes that were generated by selecting n pseudo-random points which were uniformly distributed on a spherical surface. The time needed for calculating the expected hypervolume improvement was measured (along with all operations required to do so, such as sorting the populations, but not including the time needed to generate the populations). The seed of the pseudorandom generator was the same for each calculation scheme that was tested. Figure 7.3 shows the results. There is a noticeable difference in speed between the 8-term, 5-term and 2-term scheme, but they are in the same complexity class and for any given n, their performance relative to each other is roughly the same. The slice-update scheme, by contrast, performs better relative to the other schemes when n increases, as would be expected due to its lower complexity. Even for small n it outperforms the other algorithms.

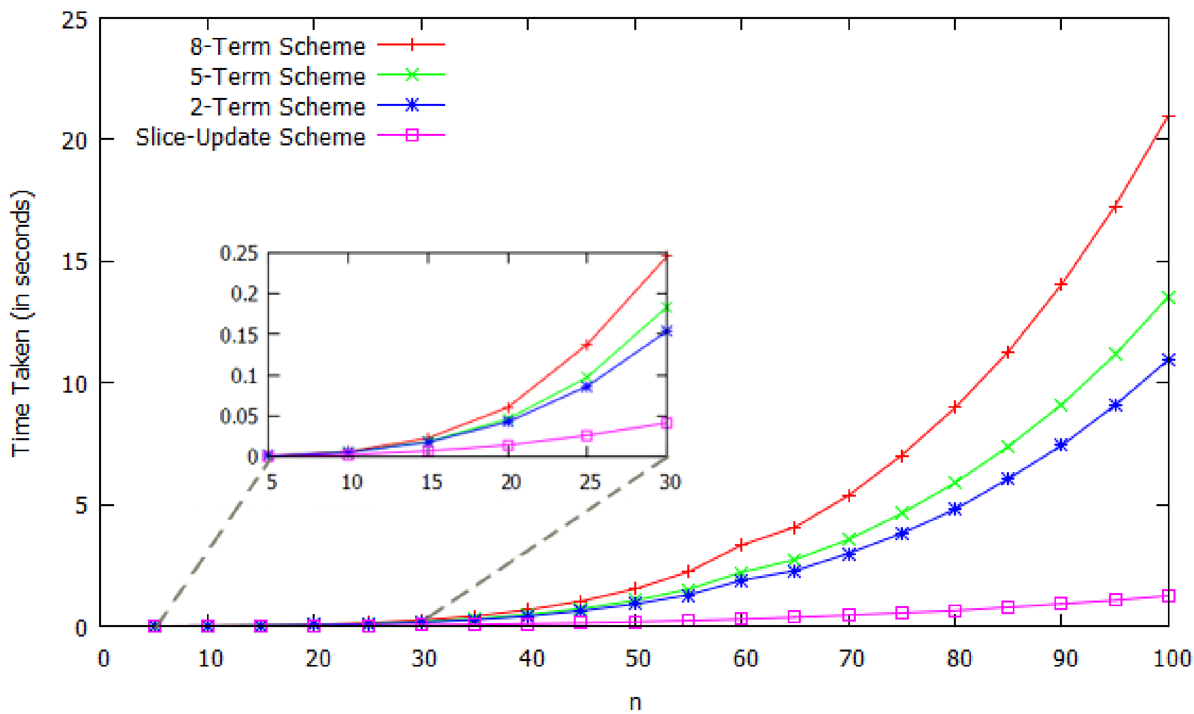

Figure 7.3: Time needed to calculate the expected hypervolume improvement for a Pareto approximation set consisting of n points randomly selected on the surface of a sphere, averaged over 10 runs.

What is interesting is that going from 8 terms to 5 terms causes a greater improvement than going from 5 terms to 2 terms, even though 2-dimensional hypervolume calculations are completely removed from the equation when going to 2 terms. No solid conclusions can be drawn from the magnitude of the differences, as they might depend on the implementation details of the code and the compiler optimizations. However, this does show that simplifying calculations can make a big difference for the speed of an algorithm. A benefit of the 2-term scheme which is not captured in the graph, is that it is the simplest scheme in terms of the number of operations that must be implemented, so the time needed to implement it will be shorter.

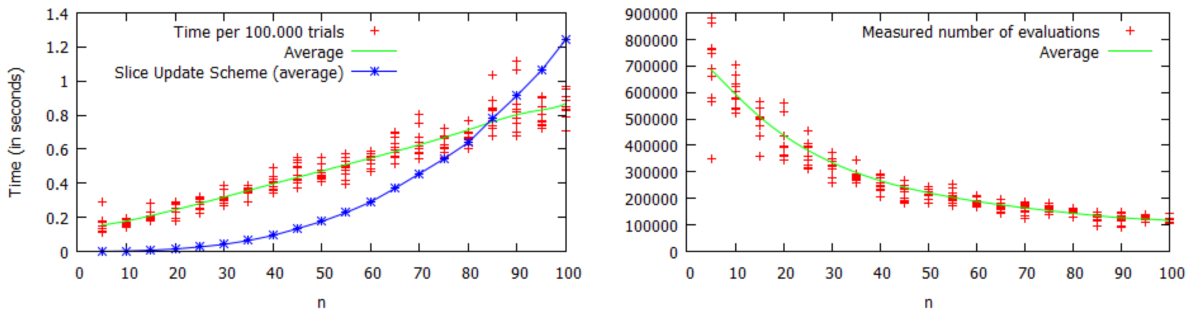

Figure 7.4: The figure on the left shows the number of Monte Carlo trials that can be performed in a second given a spherical Pareto approximation set consisting of n points. The figure on the right plots the same data as a graph of the time required for 100.000 Monte Carlo iterations, compared to the time needed for the fastest non-Monte Carlo scheme.

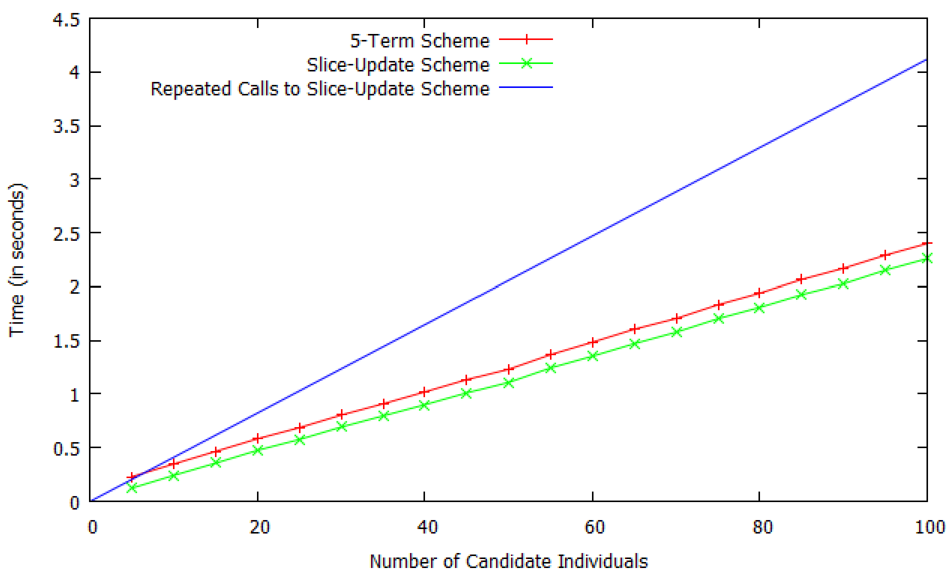

Figure 7.5: Graph showing the time needed to simultaneously calculate the EHVI on a number of candidate points using either the 5-term or slice-update schemes, for a population size of 30. The expected time taken when simply calling the slice-update scheme on each candidate individual separately is also plotted in this graph for purposes of comparison.

The Monte Carlo scheme is a special case, in that the time it takes to run depends on the desired accuracy, and this accuracy in turn also depends on the variance of the predictive distribution. When this variance is higher, the accuracy will be lower. For a rough idea of its performance relative to the exact calculation schemes, see Figure 7.4, which shows the number of Monte Carlo trials which can be performed if the algorithm is allowed to run for a second. Because of the $O(n \log n)$ time complexity of each individual trial, it is less affected by n than any of the exact calculation schemes. If n is large enough and the desired accuracy is low enough, it might be the faster option. However, when n is reasonably small, there is no advantage to using it.

The complexity of calculating the expected hypervolume improvement of multiple points by repeatedly using one of the described algorithms is of course linear in the number of candidate individuals. Here, the 8-term, 5-term and slice-update schemes have an advantage not shared by the Monte Carlo and 2-term schemes, in that their hypervolume calculations are independent of the probability distribution for which the EHVI is being calculated. This makes it possible to calculate several expected hypervolume improvements on the same population with a relatively small corresponding increase in calculation time, because the additional calculations have complexity $O(n^3)$. It is expected to be less impressive for the slice-update scheme, as this already has a time complexity of $O(n^3)$, but the amount of overhead that is avoided might still be noticeable. To determine the impact of this advantage on the relative performance of the schemes, Figure 7.5 shows the results of using the schemes to calculate the EHVI for a vector of probability distributions instead of just one.

As can be seen, the time taken increases linearly in the number of individuals evaluated at the same time, but the constant added on top of that is larger for the 5-term scheme than for the slice-update scheme. When n is 30 it is only a difference equivalent to evaluating a few more candidate individuals, however. Because the 5-term scheme is somewhat easier to implement, it might be preferable to use it if the number of candidate individuals is expected to be high in comparison to n.

Chapter 8

Conclusion and Future Work

The main results realized in this thesis are as follows:

- We have designed and implemented a faster algorithm for calculating the EHVI in two dimensions. The original algorithm had a time complexity in $O(n^3 \log n)$, whereas the new algorithm has a time complexity in $O(n^2)$. An empirical test shows improved speed even for relatively small n (> 20).
- We have provided a correct exact calculation algorithm for calculating the EHVI in more than two dimensions, written out in detail and validated in three dimensions through comparison with Monte Carlo integration.
- For the 3-dimensional case, we have additionally designed and implemented fast algorithms. Algorithms in two different complexity classes were implemented:

  With space complexity in $O(n^2)$ and time complexity in $O(n^3)$.

  With space complexity in $O(n)$ and time complexity in $O(n^4 \log n)$.
- An empirical study shows that in practice, calculating the EHVI in three dimensions can be done in a second or less when n, the size of the Pareto approximation set, is no larger than about 100. This makes it viable to use the exact calculation scheme in place of Monte Carlo integration.

Besides these results, we have also established a relationship between the expected improvement and the center of mass, which might be important for future theoretical results.

Now that an algorithm exists for exactly calculating the expected hypervolume improvement in higher dimensions, it should be verified whether the monotonicity properties of the EHVI hold in higher dimensions. Another interesting area of future work would be to investigate the performance of calculating the EHVI for dimensions greater than 3. In particular, investigating whether the $O(n^3)$ algorithm for calculating the 3-dimensional expected hypervolume indicator could be extended to higher dimensions might be worthwhile.

Appendix A

Sourcecode

The full implementations of the algorithms described in this paper are provided on the website of the LIACS natural computing group, at `natcomp.liacs.nl`. The functions implementing the 2-D hypervolume calculations, the 2-term scheme, and the slice-update scheme, are additionally provided here for convenient reference.

A.1 2-D EHVI calculation function

```
#include "helper.h"
#include "ehvi_hvol.h"
#include <deque>
#include <algorithm> //sorting
#include <math.h> //INFINITY macro
using namespace std;

//When doing 2d hypervolume calculations, uncomment the following line
//to NOT use O(1) S-minus updates (O(n log n) instead):
//#define NAIVE_DOMPOINTS

//Returns the expected 2d hypervolume improvement of population P with reference
//point r, mean vector mu and standard deviation vector s.
double ehvi2d(deque<individual*> P, double r[], double mu[], double s[]){
  sort(P.begin(), P.end(), xcomparator);
  double answer = 0; //The eventual answer
  int k = P.size(); //Holds amount of points.
  #ifdef NAIVE_DOMPOINTS
  deque<individual*> dompoints; //For the old-fashioned way.
  #endif
  double Sminus; //Correction term for the integral.
  int Sstart = k-1, Shorizontal = 0;
  //See thesis for explanation of how the O(1) iteration complexity
  //is reached. NOTE: i = y = f[1], j = x = f[0]
  for (int i=0;i<=k;i++){
    Sminus = 0;
```

```
    Shorizontal = Sstart;
    for (int j=k-i;j<=k;j++){
      double fmax[2]; //staircase width and height
      double cl1, cl2, cu1, cu2; //Boundaries of grid cells
      if (j == k){
        fmax[1] = r[1];
        cu1 = INFINITY;
      }
      else {
        fmax[1] = P[j]->f[1];
        cu1 = P[j]->f[0];
      }
      if (i == k){
        fmax[0] = r[0];
        cu2 = INFINITY;
      }
      else {
        fmax[0] = P[k-i-1]->f[0];
        cu2 = P[k-i-1]->f[1];
      }
      cl1 = (j == 0 ? r[0] : P[j-1]->f[0]);
      cl2 = (i == 0 ? r[1] : P[k-i]->f[1]);
      //Cell boundaries have been decided. Determine Sminus.
      #ifdef NAIVE_DOMPOINTS
      dompoints.clear();
      for (int m = 0; m < k; m++){
        if (cl1 >= P[m]->f[0] && cl2 >= P[m]->f[1]){
          dompoints.push_back(P[m]);
        }
      }
      Sminus = calculateS(dompoints, fmax);
      #else
      if (Shorizontal > Sstart){
        Sminus += (P[Shorizontal]->f[0] - fmax[0]) * (P[Shorizontal]->f[1] - fmax[1]);
      }
      Shorizontal++;
      #endif
      //And then we integrate.
      double psi1 = exipsi(fmax[0],cl1,mu[0],s[0]) - exipsi(fmax[0],cu1,mu[0],s[0]);
      double psi2 = exipsi(fmax[1],cl2,mu[1],s[1]) - exipsi(fmax[1],cu2,mu[1],s[1]);
      double gausscdf1 = gausscdf((cu1-mu[0])/s[0]) - gausscdf((cl1-mu[0])/s[0]);
      double gausscdf2 = gausscdf((cu2-mu[1])/s[1]) - gausscdf((cl2-mu[1])/s[1]);
      double sum = (psi1*psi2) - (Sminus*gausscdf1*gausscdf2);
      if (sum > 0)
        answer += sum;
    }
    Sstart--;
  }
  return answer;
}
```

A.2 2-term scheme for 3-D EHVI calculation

```
#include "helper.h"
```

```
#include "ehvi_hvol.h"
#include <deque>
#include <algorithm> //sorting
#include <math.h> //INFINITY macro
using namespace std;

//2-term 3-D EHVI calculation scheme.
double ehvi3d_2term(deque<individual*> P, double r[], double mu[], double s[]){
  double answer = 0; //The eventual answer.
  int n = P.size(); //Holds amount of points.
  double Sminus; //Correction term for the integral.
  deque<individual*> Py, Pz; //P sorted by y/z coordinate
  sort(P.begin(), P.end(), ycomparator);
  for (int i=0;i<P.size();i++){
    Py.push_back(P[i]);
  }
  sort(P.begin(), P.end(), zcomparator);
  for (unsigned int i=0;i<P.size();i++){
    Pz.push_back(P[i]);
  }
  sort(P.begin(), P.end(), xcomparator);
  for (int z=0;z<=n;z++){
    for (int y=0;y<=n;y++){
      for (int x=0;x<=n;x++){
        double fmax[3]; //upper corner of hypervolume improvement box
        double cl[3], cu[3]; //Boundaries of grid cells
        cl[0] = (x == 0 ? r[0] : P[x-1]->f[0]);
        cl[1] = (y == 0 ? r[1] : Py[y-1]->f[1]);
        cl[2] = (z == 0 ? r[2] : Pz[z-1]->f[2]);
        cu[0] = (x == n ? INFINITY : P[x]->f[0]);
        cu[1] = (y == n ? INFINITY : Py[y]->f[1]);
        cu[2] = (z == n ? INFINITY : Pz[z]->f[2]);
        //Calculate expected one-dimensional improvements w.r.t. r
        double psi1 = exipsi(r[0],cl[0],mu[0],s[0]) - exipsi(r[0],cu[0],mu[0],s[0]);
        double psi2 = exipsi(r[1],cl[1],mu[1],s[1]) - exipsi(r[1],cu[1],mu[1],s[1]);
        double psi3 = exipsi(r[2],cl[2],mu[2],s[2]) - exipsi(r[2],cu[2],mu[2],s[2]);
        //Calculate the probability of being within the cell.
        double gausscdf1 = gausscdf((cu[0]-mu[0])/s[0]) - gausscdf((cl[0]-mu[0])/s[0]);
        double gausscdf2 = gausscdf((cu[1]-mu[1])/s[1]) - gausscdf((cl[1]-mu[1])/s[1]);
        double gausscdf3 = gausscdf((cu[2]-mu[2])/s[2]) - gausscdf((cl[2]-mu[2])/s[2]);
        //Calculate the 'expected position of p' and the correction term Sminus
        if (gausscdf1 == 0 || gausscdf2 == 0 || gausscdf3 == 0)
          continue; //avoid division by 0, cell contribution is 0 in these cases anyway.
        fmax[0] = (psi1/gausscdf1) + r[0];
        fmax[1] = (psi2/gausscdf2) + r[1];
        fmax[2] = (psi3/gausscdf3) + r[2];
        Sminus = hvol3d(Pz, r, fmax);
        //the expected hypervolume improvement is the expected rectangular volume
        //w.r.t. r minus the correction term Sminus
        double sum = (psi1*psi2*psi3) - (Sminus*gausscdf1*gausscdf2*gausscdf3);
        if (sum > 0) //Safety check; "Not-A-Number > 0" returns false
          answer += sum;
      }
    }
  }
  return answer;
```

```
}
```

A.3 Slice-update scheme for 3-D EHVI calculation

```
#include "ehvi_consts.h"
#include <deque>
#include <algorithm>
#include <iostream> //For error on exception only.
#include <cmath> //INFINITY macro
#include "ehvi_hvol.h"

using namespace std;

#ifndef EHVI_SLICEUPDATE
#define EHVI_SLICEUPDATE

double ehvi3d_sliceupdate(deque<individual*> P, double r[], double mu[], double s[]){
//EHVI calculation algorithm with time complexity O(n^3).
  double answer = 0; //The eventual answer.
  specialind *newind;
  int n = P.size(); //Holds amount of points.
  thingy *Pstruct; //2D array with info about the shape of the dominated hypervolume
  deque<specialind*> Px, Py, Pz; //P sorted by x/y/z coordinate with extra info
  double cellength[3] = {0};
  try{
    //Create sorted arrays which contain extra information allowing the location in
    //the other sorting orders to be ascertained in O(1).
    sort(P.begin(), P.end(), xcomparator);
    for (unsigned int i=0;i<n;i++){
      newind = new specialind;
      newind->point = P[i];
      newind->xorder = i;
      Px.push_back(newind);
      Py.push_back(newind);
      Pz.push_back(newind);
    }
    sort(Py.begin(), Py.end(), specialycomparator);
    for (unsigned int i=0;i<n;i++){
      Py[i]->yorder = i;
    }
    sort(Pz.begin(), Pz.end(), specialzcomparator);
    for (unsigned int i=0;i<n;i++){
      Pz[i]->zorder = i;
    }
    //Then also reserve memory for the structure array.
    Pstruct = new thingy[n*n];
    for (int k=0;k<n*n;k++){
      Pstruct[k].slice = 0;
      Pstruct[k].chunk = 0;
      Pstruct[k].highestdominator = -1;
      Pstruct[k].xlim = 0;
      Pstruct[k].ylim = 0;
    }
  }
```

```
  catch (...) {
    cout << "An exception was thrown. There probably isn't enough memory available."
         << endl;
    cout << "-1 will be returned." << endl;
    return -1;
  }
  //Now we establish dominance in the 2-dimensional slices. Note: it is assumed that
  //P is mutually nondominated. This implementation of that step is O(n^3).
  for (int i=0;i<n;i++){
    for (int j=Pz[i]->yorder;j>=0;j--)
      for (int k=Pz[i]->xorder;k>=0;k--){
        Pstruct[k+j*n].highestdominator = i;
      }
    for (int j=Px[i]->zorder;j>=0;j--)
      for (int k=Px[i]->yorder;k>=0;k--){
        Pstruct[k+j*n].xlim = Px[i]->point->f[0] - r[0];
      }
    for (int j=Py[i]->zorder;j>=0;j--)
      for (int k=Py[i]->xorder;k>=0;k--){
        Pstruct[k+j*n].ylim = Py[i]->point->f[1] - r[1];
      }
  }
  //And now for the actual EHVI calculations.
  for (int z=0;z<=n;z++){
    //Recalculate Pstruct for the next 2D slice.
    if (z>0)
      for (int i=0;i<n*n;i++){
        Pstruct[i].chunk += Pstruct[i].slice * cellength[2];
      }
    //This step is O(n^2).
    for (int y=0;y<n;y++){
      for (int x=0;x<n;x++){
        if (Pstruct[x+y*n].highestdominator < z){ //cell is not dominated

          if (x > 0 && y > 0){
            Pstruct[x+y*n].slice = (Pstruct[x+(y-1)*n].slice
                                   - Pstruct[(x-1)+(y-1)*n].slice)
                                   + Pstruct[(x-1)+y*n].slice;
          }
          else if (y > 0){
            Pstruct[x+y*n].slice = Pstruct[x+(y-1)*n].slice;
          }
          else if (x > 0){
            Pstruct[x+y*n].slice = Pstruct[(x-1)+y*n].slice;
          }
          else
            Pstruct[x+y*n].slice = 0;
        }
        else {
          Pstruct[x+y*n].slice = (Px[x]->point->f[0] - r[0])
                               * (Py[y]->point->f[1] - r[1]);
        }
      }
    }
    //Okay, now we are going to calculate the EHVI, for real.
    for (int y=0;y<=n;y++){
```

```
    for (int x=0;x<=n;x++){
      double cl[3], cu[3]; //Boundaries of grid cells
      cl[0] = (x == 0 ? r[0] : Px[x-1]->point->f[0]);
      cl[1] = (y == 0 ? r[1] : Py[y-1]->point->f[1]);
      cl[2] = (z == 0 ? r[2] : Pz[z-1]->point->f[2]);
      cu[0] = (x == n ? INFINITY : Px[x]->point->f[0]);
      cu[1] = (y == n ? INFINITY : Py[y]->point->f[1]);
      cu[2] = (z == n ? INFINITY : Pz[z]->point->f[2]);
      cellength[0] = cu[0] - cl[0];
      cellength[1] = cu[1] - cl[1];
      cellength[2] = cu[2] - cl[2];
      if (cellength[0] == 0 || cellength[1] == 0 || cellength[2] == 0
          || (x < n && y < n && Pstruct[x+y*n].highestdominator >= z))
        continue; //Cell is dominated or of size 0.
      //We have easy access to Sminus and zslice because they are part of Pstruct.
      //xslice and yslice can be calculated from Pstruct->chunk.
      double slice[3], Sminus, v[3];
      if (x > 0 && y > 0){
        Sminus = Pstruct[(x-1)+(y-1)*n].chunk;
        slice[0] = (x == n ? 0 : (Pstruct[x+(y-1)*n].chunk - Sminus) / cellength[0]);
        slice[1] = (y == n ? 0 : (Pstruct[(x-1)+y*n].chunk - Sminus) / cellength[1]);
        slice[2] = Pstruct[(x-1)+(y-1)*n].slice;
      }
      else {
        Sminus = 0;
        slice[0] = ((y == 0 || x == n) ? 0 :
                      (Pstruct[x+(y-1)*n].chunk - Sminus) / cellength[0]);
        slice[1] = ((x == 0 || y == n) ? 0 :
                      (Pstruct[(x-1)+y*n].chunk - Sminus) / cellength[1]);
        slice[2] = 0;
      }
      if (y == n || z == n)
        v[0] = 0;
      else
        v[0] = Pstruct[y+z*n].xlim;
      if (x == n || z == n)
        v[1] = 0;
      else
        v[1] = Pstruct[x+z*n].ylim;
      if (x == n || y == n)
        v[2] = 0;
      else
        v[2] = (Pstruct[x+y*n].highestdominator == -1 ? 0 :
                  (Pz[Pstruct[x+y*n].highestdominator]->point->f[2] - r[2]));
      //All correction terms have been established.
      //Calculate the cell's contribution to the integral.
      double psi1 = exipsi(r[0],cl[0],mu[0],s[0]) - exipsi(r[0],cu[0],mu[0],s[0]);
      double psi2 = exipsi(r[1],cl[1],mu[1],s[1]) - exipsi(r[1],cu[1],mu[1],s[1]);
      double psi3 = exipsi(r[2],cl[2],mu[2],s[2]) - exipsi(r[2],cu[2],mu[2],s[2]);

      double gausscdf1 = gausscdf((cu[0]-mu[0])/s[0]) - gausscdf((cl[0]-mu[0])/s[0]);
      double gausscdf2 = gausscdf((cu[1]-mu[1])/s[1]) - gausscdf((cl[1]-mu[1])/s[1]);
      double gausscdf3 = gausscdf((cu[2]-mu[2])/s[2]) - gausscdf((cl[2]-mu[2])/s[2]);

      double ex1 = psi1 - (gausscdf1 * (cl[0]-r[0]));
      double ex2 = psi2 - (gausscdf2 * (cl[1]-r[1]));
```

```
            double ex3 = psi3 - (gausscdf3 * (cl[2]-r[2]));

            double sum = (psi1*psi2*psi3) - (Sminus*gausscdf1*gausscdf2*gausscdf3);
            //Subtract correction terms:
            sum -= (slice[0] * gausscdf2 * gausscdf3 * ex1);
            sum -= (slice[1] * gausscdf1 * gausscdf3 * ex2);
            sum -= (slice[2] * gausscdf1 * gausscdf2 * ex3);
            sum -= v[0] * ex2 * ex3 * gausscdf1;
            sum -= v[1] * ex1 * ex3 * gausscdf2;
            sum -= v[2] * ex1 * ex2 * gausscdf3;
            if (sum > 0)
              answer += sum;
          }
        }
      }
    return answer;
  }
#endif
```

Appendix B

How to Use the Software

The code is organized as follows: *ehvi_sliceupdate.cc* contains the implementation of the slice-update scheme, *ehvi_montecarlo.cc* contains the implementation of the Monte Carlo integration scheme, *ehvi_calculations.cc* contains the implementations of the other schemes (2-term, 5-term and 8-term) as well as the implementation of the 2-dimensional calculation scheme. *ehvi_multi.cc* contains special implementations of the 5-term and slice-update schemes which calculate the EHVI for multiple individuals at the same time. The files *helper.cc* and *ehvi_hvol.cc* contain functions which are used by multiple update schemes. Hypervolume calculations are implemented in *ehvi_hvol.cc* and the rest (such as the psi function) is in *helper.cc*. There is also a file *ehvi_consts.h* which allows the seed of the random number generator and the number of Monte Carlo iterations to be changed.

The functions in these files can be used directly in C++ code, but *main.cc* contains facilities to use it as a stand-alone command-line application. To use it in this way, the software can be compiled with gcc by unzipping the code into a directory and using the command:

```
g++ -O3 -o EHVI -std=c++0x *.cc
```

If compiling it in a directory which also contains other .cc files is desired, the following can be used instead:

```
g++ -O3 -o EHVI -std=c++0x main.cc helper.cc ehvi_sliceupdate.cc \
ehvi_multi.cc ehvi_montecarlo.cc ehvi_hvol.cc ehvi_calculations.cc
```

Running the software without command line arguments will allow a test case to be entered into the terminal (with an arbitrary population size and 1 candidate individual), which is then run through the available calculation schemes. Providing the name of a file as an argument will load that file and perform EHVI calculations on it. The file should consist of the following:

- A single integer representing n
- n*3 floating point numbers representing the coordinates of P.
- 3 floating point numbers representing r
- An arbitrary number of individuals, represented by first 3 coordinates of their mean value, and then the 3 values of their standard deviation in each dimension

No other text should be present in the file. All numbers should be divided by whitespace, and additional whitespace is ignored.

The default scheme used will be the slice-update scheme, but other schemes can be specified as arguments after the filename. The list of possible schemes to request is:

```
2term
5term
8term
sliceupdate
montecarlo
```

However, only *5term* and *sliceupdate* can be used if more than one candidate individual is provided.

```
4
8 8 2
11 6 7
9 5 8
14 3 9

0 0 0
6 6 6 3 3 3
5 2 4 1 3 6
1 7 2 3 5 3
2 3 5 2 8 3
```

Then the following command will calculate the EHVI for the four candidate individuals using the 5-term scheme:

```
./EHVI multitest.txt 5term
```

And the output of the program will be:

```
Loading testcase from file...
Calculating with 5-term scheme (multi-version)...
47.24623199
11.21775781
8.935099634
19.88518203
```

Only the actual answers are output to stdout while the rest of the output is written to stderr, making it possible to write the answers to a file using simple shell commands.

Bibliography

[1] Wagner, T.; Emmerich, M.; Deutz, A. and Ponweiser, W. (2010) “On expected-improvement criteria for model-based multi-objective optimization”, in ‘Proc. of PPSN XI Vol. 1, Springer-Verlag, Berlin, Heidelberg, pp. 718-727.

[2] Fleischer, M. (2003) “The Measure of Pareto Optima Applications to Multi-objective Metaheuristics”. Evolutionary Multi-Criterion Optimization. Second International Conference, EMO 2003, pg. 519-533.

[3] Emmerich, M. T M; Deutz, A.H.; Klinkenberg, J.W. (2011) “Hypervolume-based expected improvement: Monotonicity properties and exact computation,” 2011 IEEE Congress on Evolutionary Computation (CEC), pp.2147-2154

[4] Nicola Beume, Carlos M. Fonseca, Manuel Lopez-Ibanez, Luis Paquete, and Jan Vahrenhold. (2009) “On the complexity of computing the hypervolume indicator.” IEEE Trans. Evolutionary Computation, 13(5) pp. 1075-1082.

[5] Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P. (1989) “Design and analysis of computer experiments”. Statistical science, 4(4), 409-423.

[6] Mockus, J., Tiesis, V., Zilinskas, A. (1978) “The application of Bayesian methods for seeking the extremum”. In: Dixon, L., Szego, G. (Eds.), Towards Global Optimization, vol. 2. North Holland, New York, pp. 117129.

[7] Donald R. Jones, Matthias Schonlau, and William J. Welch. (1998) “Efficient Global Optimization of Expensive Black-Box Functions”. J. of Global Optimization 13, 4 (December 1998), 455-492.

[8] Emmanuel Vazquez and Julien Bect. (2010) “Convergence properties of the expected improvement algorithm with xed mean and covariance functions”. Journal of Statistical Planning and Inference 140, pp. 3088-3095

[9] Knowles, J. (2006) “ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems”. IEEE Transactions on Evolutionary Computation. 10 (1): 50-66.

[10] Keane, A.J. (2006) “Statistical improvement criteria for use in multiobjective design optimisation”. AIAA Journal, 44, (4), 879-891.

[11] Wolfgang Ponweiser, Tobias Wagner, Dirk Biermann, and Markus Vincze. (2008) “Multiobjective Optimization on a Limited Budget of Evaluations Using Model-Assisted $\mathcal{S}$-Metric Selection”. In Proceedings of the 10th international conference on Parallel Problem Solving from Nature: PPSN X. Springer-Verlag, Berlin, Heidelberg, 784-794.

[12] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. (2000) “A fast and elitist multi-objective genetic algorithm: NSGA-II”.

[13] E. Zitzler, M. Laumanns and L. Thiele. (2001) “SPEA2: Improving the Strength Pareto Evolutionary Algorithm”.

[14] Michael Emmerich, Nicola Beume, and Boris Naujoks. (2005) “An EMO Algorithm Using the Hypervolume Measure as Selection Criterion”. In 2005 Intl Conference, March 2005, pages 62-76.

[15] Christian Igel, Nikolaus Hansen, and Stefan Roth. (2007) “Covariance Matrix Adaptation for Multi-objective Optimization”. Evol. Comput. 15, 1 (March 2007), 1-28.

[16] Williams, Christopher K.I. (1998) “Prediction with Gaussian processes: From linear regression to linear prediction and beyond”. In M. I. Jordan. Learning in graphical models. MIT Press. pp. 599612.

[17] Fubini, G. ”Sugli integrali multipli.” (1958) Opere scelte, Vol. 2. Cremonese, pp. 243-249.

[18] Matsumoto, M.; Nishimura, T. (1998) “Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator”. ACM Transactions on Modeling and Computer Simulation 8 (1): 330

[19] G. E. P. Box, Mervin E. Muller. (1958) “A Note on the Generation of Random Normal Deviates”. The Annals of Mathematical Statistics, Vol. 29, No. 2. pp. 610-611