Towards Multi-objective Bayesian Global Optimization for Space Missions

THESIS

submitted in partial fulfillment of the
requirements for the degree of

BACHELOR OF SCIENCE
in
COMPUTER SCIENCE

Author :	Kean Tettelaar
Student ID :	1500325
Supervisor :	Michael Emmerich, LIACS
2^{nd} corrector :	Yali Wang, LIACS

Leiden, The Netherlands, August 5, 2020

Towards Multi-objective Bayesian Global Optimization for Space Missions

Kean Tettelaar

LIACS, Leiden University

August 5, 2020

Abstract

This thesis will develop the initial steps for multi-objective Bayesian Global Optimization (BGO) for space mission design using ESA/ESTEC's scientific optimization library PyGMO which they use for designing space missions. The goal of this thesis is to perform the first steps for a professional integration of the EHVI optimization technique in the PyGMO library developed at ESA/ESTEC (Noordwijk, The Netherlands). I will discuss the expected hypervolume improvement (EHVI) technique, which is a generalized BGO technique to multi-objective optimization problems. This thesis will develop and discuss an implementation of this technique in Python, and the PyGMO library. Then by benchmarking the EHVI algorithm, on the ZDT optimization test problems, against similar algorithms already available in PyGMO and looking at various performance and convergence metrics I conclude that the EHVI technique is a powerful technique for problems with a limited budget of black-box function evaluations, for a professional integration in PyGMO there is still more work to be done.

Contents

Version of August 5, 2020– Created August 5, 2020 - 10:57

Chapter 1

Introduction

Mathematical optimization is finding the optimum of a mathematical programming problem or objective function. An optimization problem consists of finding a minimum or maximum of a real function by choosing some input values within the bounds of the function and algorithmically selecting better values until the optimum is reached. Some algorithms utilize techniques that compute many function values, and each iteration or generation modify the solutions found so far to create a ”better” set than the last iteration. Others try to minimize the number of function values that are computed because this function might be computationally expensive to evaluate.

The ideas of Bayesian Global Optimization (BGO) proposed in the 1970s. These ideas had a significant, lasting effect on the development of global optimization techniques. BGO is a type of technique that comparatively does not need many function evaluations; as such, it performs faster than other optimization techniques when the evaluation of this function is computationally costly.

The Expected Improvement algorithm is a BGO technique in which the goal is to find the minimum (or maximum) of a function. This function is called the objective function in the optimization terms. BGO assumes this objective function can be approximated by a Gaussian process or Gaussian random field. A Gaussian process is a data-driven model and can infer a distribution over functions directly. With some known objective function values a GP can be converted into a posterior distribution over the objective function, and can then be used to make predictions about unknown points. Where the Expected Improvement technique only has a

single objective, there are many examples of problems or functions with two or multiple objectives which all need to be minimized or maximized at the same time.

When dealing with these so-called multi-objective optimization problems the outcome of optimization techniques or algorithms is no longer a single "best" value but a collection of points in the objective-space which has a dimensionality equal to the number of objectives, this collection is referred to as the *Pareto front*. Vectors in the Pareto front *dominate* all other solution vectors in one or more objectives. A more detailed explanation of Bayesian Global Optimization and the Pareto front is given in Chapter 2 *Bayesian Global Optimization*.

A generalized Expected Improvement technique for multiple objectives called Expected Hypervolume Improvement (EHVI) developed by Emmerich, Yang, Deutz,Wang and Fonseca [1] that brings multi-objective optimization capability to Bayesian Global Optimization. This technique is the main focus of this thesis.

Concerning space missions, ESA/ESTEC located in Noordwijk has designed a scientific software library, consisting of single-objective and multi-objective optimization problems and algorithms that are used to optimize these problems, called PyGMO. The space mission problems are more often than not of the multi-objective kind. One can imagine that the fuel cost and duration of a mission are both objectives that are to be minimized for an efficient and cost-effective mission. ESTEC has expressed interest in BGO optimization algorithms for their library since they do not have any yet. Especially the realm of multi-objective BGO techniques is the topic of much research, and it seems to have computational advantages in some situations compared to other techniques. However, they are not yet developed or integrated into their PyGMO library.

This thesis is organized as follows: Discussion and presentation of the underlying techniques and elements of Multi-objective Bayesian Global Optimization in Chapter 2. Chapter 3 covers the EHVI algorithm and the implementation in Python of this algorithm. Chapter 4 introduces the scientific optimization library PyGMO developed by ESA/ESTEC and the integration of the EHVI algorithm in their API. In Chapter 5, presents methods for benchmarking the EHVI algorithm against similar multi-objective optimization algorithms already available in PyGMO. The results of these benchmarks are presented in Chapter 6. Finally, Chapter 7 will cover the

discussion of the results, the limitations, and problems of the EHVI algorithm that presented themselves during this process. This final chapter will also cover further prospects of the EHVI algorithm's development and possible integration in the PyGMO library.

Chapter 2

Bayesian Global Optimization

An optimization problem with some objective function f to be optimized is called a global optimization problem if there is no analytical expression or information about the nature of the objective function. In particular, if it is unknown if f has one or more minima (or maxima). The evaluation of f is restricted to sampling the function at a point x.
Optimization problems with black-box objective functions are generally classified as global optimization problems because their nature is unknown. The optimization problems discussed in this thesis are all of the black-box kind. Furthermore, the optimization problems discussed are continuous, well defined, and unconstrained.
From now on, we will assume and consider minimization when discussing optimization without loss of generality.

When performing global optimization, one thing to keep in mind is that there can be multiple local minima in which an algorithm can get stuck. So techniques for escaping these local minima, like global search or the ability to accept worse solutions, are necessary.
Generally, finding the exact global minimum for multi-objective optimization problems is considered an impossible task, so the problem is specified as finding approximations.

If the objective function f is cheap to evaluate, you could sample many points using grid search or a similar technique to get an estimation of the objective function. If this is not the case, meaning an objective function evaluation is comparatively expensive, it is crucial to minimize the number of evaluations performed. This domain of global optimization is where Bayesian methods are most useful. The BGO techniques attempt to find

the global minima in a minimum number of objective function evaluations.
In BGO there exists a prior belief or distribution of the objective function f which is updated with each function evaluation to get a posterior distribution of f, which then better approximates the objective function f.
This posterior distribution or surrogate function of f is relatively cheap to evaluate. This comparatively inexpensive evaluation is where the strength of BGO comes into play since this function will be probed instead of f.
The most commonly used surrogate model for BGO are Gaussian processes; these are covered thoroughly in Section 2.1.

BGO also makes use of acquisition functions which probes the surrogate function for areas where there exists an improvement over the current best solution. The objective function is evaluated in these areas with the most significant improvement. These acquisition functions make the trade-off between exploration and exploitation of the search space, to escape local minima. The acquisition function we will discuss in this thesis is the *expected improvement* model, and it will be discussed more in Section 2.2.

2.1 Gaussian Processes

A Gaussian process is by definition a collection of random variables, the joint distribution of any finite number of these variables is also a Gaussian distribution. We will refer to a Gaussian process as a GP going forward.

A GP is essentially a multivariate Gaussian distribution of infinite dimension. Such a distribution is defined by the mean μ and a covariance matrix Σ.
The mean is the expected value of the gaussian distribution. The covariance matrix tells us the variance for each random variable (dimension) and how the different random variables are correlated; this matrix is always positive definite.

In a GP any point $x \in R^d$ within the bounds of an objective function is assigned a random variable $F(x)$, and the joint distribution of any finite number of these variables $P(f(x_1), ..., f(x_n))$ is itself a Gaussian distribution.
The correlation $P(F_u, F_v)$ between random variables F_u and F_v for every pair of indices $u \in R^d$ and $v \in R^d$ in a GP is defined by a correlation

function. The correlation is always positive definite and depends on the relation between u and v for instance, the *Euclidean* distance between the two points.

Gaussian distributions are closed under *marginalization* and *conditioning*; this means that we can extract information about the marginalized probability distributions of subsets of the multivariate distribution. Specifically, these marginalized distributions only depend on their mean μ and entry in the covariance matrix Σ:

Let X and Y be subsets of a Gaussian distribution G, $X \subset G$ and $Y \subset G$. These subsets have μ_X and μ_Y as their mean values, and covariance matrix entries Σ_{XX} and Σ_{YY}; the marginalized probability distributions of X and Y are only dependent on those variables:

$$P(X) \sim N(\mu_X, \Sigma_{XX}) \text{ and } P(Y) \sim N(\mu_Y, \Sigma_{YY}),$$

for some function N.

Conditioning allows us to predict the probability of one variable depending on another. This prediction means we can calculate the marginal distributions of each random variable in a GP which allows us to infer the conditional distribution between X and Y.

When adding data to the GP prior distribution, we need to compute μ and Σ. GP's often assume $\mu = 0$, which simplifies the conditioning.
The covariance matrix Σ determines the shape of our distribution and the characteristics of the objective function that we are modelling.

A kernel function computes a covariance matrix. This kernel evaluates pairwise combinations of the prior- or training- data to calculate the similarity or correlation between two points.
There exist many different kernels used to compute covariance matrices for Gaussian processes. In this thesis, we will be using a stationary kernel in which the correlation of two points is dependent on their relative *Euclidean* distance to one another.
This kernel is known as the *Radial Basis Function* or RBF kernel. This kernel computes the correlation between two points x and x' separated by their *Euclidean* distance $||x - x'||$:

$$K_{xx'} = exp(-\frac{||x-x'||^2}{2l^2})$$

The parameter l is the *length scale* which is a RBF kernel hyperparameter. Increasing values of l, in turn, increase the correlation between points that lie further apart. The variance and *length scale* determine the shape of the modelled objective function as well as the confidence of a prediction.
A *log-marginal likelihood* optimization technique determines the optimal value of the hyperparameter l during the training process of a GP.
With these tools, we can begin to define a probability distribution over an objective function.

The distribution that is represented by the GP, without having received any data or information about the objective function f, is called the prior distribution $P(f(x))$.
After observing some training data X and initial objective function evaluations Y, a GP prior distribution becomes a GP posterior function $P(f(x)|Y, X)$.
This posterior distribution has a (most likely nonzero) mean μ determined by the training data and covariance matrix Σ determined by the RBF kernel.
To summarize, given training data or prior information of the objective function $F_{x^{(1)}} = f(x^{(1)}), ..., F_{x^{(t-1)}} = f(x^{(t-1)})$, the conditional mean μ and the conditional variance σ^2 can be computed for a conditioned random variable F_x in the Gaussian Process:

$$F_x|F_x^{(1)} = f(x^{(1)}), ..., F_{x^{(t-1)}} = f(x^{(t-1)})$$

This posterior function can then make predictions about a objective function value f^* for new vector $x* \in \mathbb{R}^d$ from here on represented by x^*. This prediction $f^{(*)}$ is represented by the conditional mean μ_{x^*} and conditional variance Σ_{x^*} of vector x^*.
The key idea of a GP is to infer the distribution of the training data X together with new input x^*. We are interested in the conditional probability distribution, which is distributed normally under conditioning.
This attribute of the conditional distribution means that the resulting distribution $P(f^*|X, x*)$ is also Gaussian with mean μ^* and covariance matrix $Sigma^*$.
This mu^* and Σ^*, in particular, the standard deviation $\sigma*$ are the parameters used by the *expected improvement* algorithm to infer the improvement expected at vector x^*. Because x^* itself has a distribution, it has a probability associated with it. This probability is why we can use the term *expected*

improvement instead of only the improvement itself.

For a more detailed insight into Gaussian Processes, specifically GPs applied to machine learning, I recommend the book *Gaussian Processes for Machine Learning* by *C.E. Rasmussen C.K.I. Williams* [6].

2.2 Expected Improvement

In Bayesian Global Optimization, proposing new sampling points in the search space; to be evaluated by the objective function f is performed by acquisition functions. These functions make a trade-off between *exploitation* of the search space where it samples locations where the surrogate model (the Gaussian Process) predicts high objective function values and *exploration* of the search space where it samples areas where the uncertainty of predictions is high. The exploration is associated with higher values of the mean μ and the exploitation with higher values of the standard deviation σ. The goal of Bayesian Global Optimization is to optimize the acquisition function to sample the next vector that yields the highest *expected improvement* when will be evaluated in the objective function and added to the GP as new prior information about f.

The improvement of a function value $y \in \mathbb{R}^d$ can be defined as:

$$I(y) = max(0, y_{min} - y),$$

where $y_{min} = min(y^{(1)}, ..., y^{(t-1)}$ is the minimum of the already evaluated objective function values (the GP prior information).

The function value y of a candidate sample point x is considered to be a Gaussian random variable in our GP with mean μ_x and variance σ_x^2. The *expected improvement*, with regard to the GP model, for such a candidate point, can then be written as:

$$E(I(F_x|X, f(X))) = \int_{x=-\infty}^{x=x_{min}} I(y) PDF_{x|X,f(X)}(y) dx,$$

where $PDF_{x|X,f(X)}$ is the *probability density function* of the conditional probability distribution of F_x given prior information $X, f(X) : F_x|X, f(X)$

2.3 Multi-objective Optimization

When considering a multi-objective optimization problem consisting of m dimensions, multiple objective functions are to be minimized simultaneously. The solution of such an optimization problem is no longer an optimal point f_{opt} in two dimensional space but a vector $f_{opt} \in \mathbb{R}^m$ in m dimensional space. We define the set of optimal points as the *Pareto front*.

2.3.1 The Pareto Front

To define the Pareto front, we need to define *Pareto dominance order* $\prec$ on $\mathbb{R}^m$, with $\forall x, y \in \mathbb{R}^m : x \prec y (\forall i \in \{1, ..., m\} : x_i \leq y_i)$ and $x \neq y$.

The Pareto dominance order means is that if for all elements x_i of vector x it has a smaller or equal value than the element y_i of vector y, it is then said that x is *dominating* point y.

The Pareto front is a non-dominated subset of a set of vectors. In the case of multi-objective BGO, these vectors are evaluated objective function values. Specifically, in the two-dimensional case with objective functions $f_1(x)$ and $f_2(x)$, a Pareto optimal vector y is of dimensionality 2:
$y = (y_1, y_2)$ which represents a point in the objective space.

The set of all solution vectors $F = \{y_1, ...y_n\}$ contains a non-dominated subset of vectors that dominate all other solutions. This subset is the Pareto front and is defined as:

$$F_{nd} = \{y \in F | \nexists w \in F : w \prec y\}$$

An observer can then analyze this set and make trade-off decisions in favour of one of the objectives if deemed necessary.

Chapter 3

Expected Hypervolume Improvement

The *expected hypervolume improvement* algorithm, from now on, referred to as the EHVI algorithm is a generalization of the single-objective expected improvement algorithm for $m \geq 2$ objectives. We will only consider the bi-objective variant here.

The main idea behind the EHVI technique is sampling a new point where it has the highest expected improvement. However, the improvement measure from the single-objective *expected improvement* algorithm does not translate directly to problems with multiple objectives. One way to generalize Bayesian Global Optimization and the expected improvement algorithm to multi-objective optimization problems, is to compute the expected improvement of the hypervolume or hypervolume indicator.

3.1 The Hypervolume Indicator

The hypervolume indicator is a m-dimensional volume measure (*Lebesque measure*) λ_m of the subspace dominated by the Pareto front, bounded by a reference point r^m. In the bi-objective case, the hypervolume indicator is the area of the non-dominated subspace. The formal definition of the hypervolume indicator is given by:

$$hv(F_{nd}) = \lambda_m(f \in \mathbb{R}^d : \exists x \in F : x \prec f \wedge f \prec r),$$

where F is the set of non-dominated solutions in objective space, f is such a solution, and r is the reference point.

An illustration of a problem with two objective functions, is given in Figure 3.1 The Pareto front F_{nd} has nine points, these points are denoted by $f_i, i = \{1, ...9\}$. The shaded area represents the hypervolume indicator.

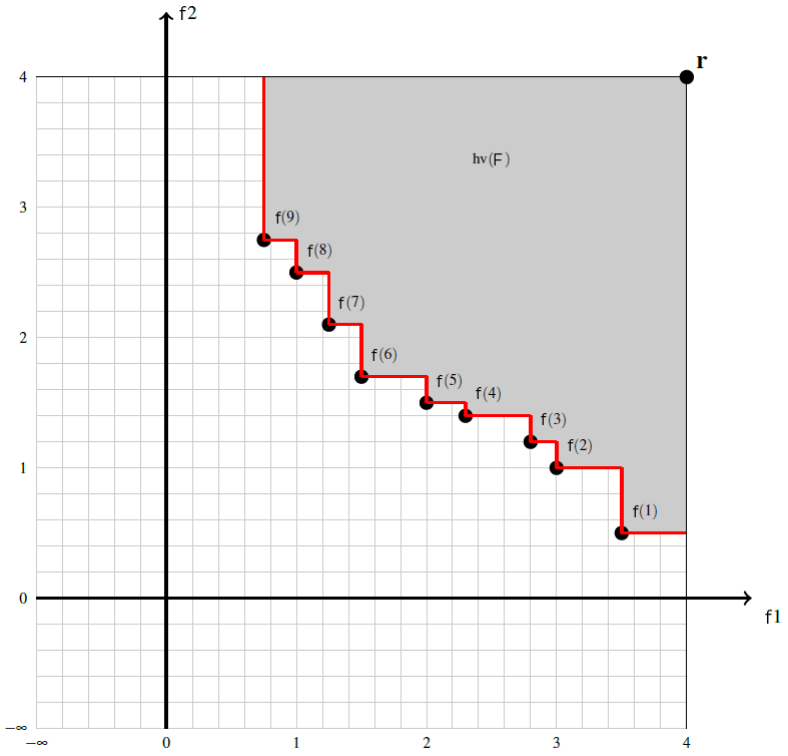


Figure 3.1: *The hypervolume indicator for a two-dimensional optimization problem*

The problem of finding approximations to the Pareto front is sometimes referred to as maximizing the hypervolume indicator. Each new non-dominated point that is added to the non-dominated subset F_{nd} of F increases the value of the hypervolume indicator.
With this in mind, we can think about using the hypervolume indicator as an improvement measure for multi-objective Bayesian global optimization.

3.2 Expected Hypervolume Improvement

The function of the EHVI algorithm is to determine a new sample vector x^m which has the highest expected improvement of the hypervolume indicator.
By calculating the hypervolume indicator over the set of non-dominated solutions, we get a measure of the best solution found up to iteration $t-1$. Generalizing the best function value found for the single-objective expected improvement to the set of non-dominated points (Pareto front) in the multiobjective case:

$$y_{min}^{t-1} \rightarrow F_{nd,t-1} = (y_1, ..., y_{t-1})_{nd}$$

The generalization of the improvement function is given by:

$$I_m(y, F_{nd,t-1}, r) := hv(F_{nd,t-1} \cup \{y\}) - hv(F_{nd,t-1})$$

The expected improvement determines the hypervolume indicator for the joint set of the current Pareto front $F_{nd,t-1}$ and predicted solution y for vector x computed by the Gaussian Process.
It then subtracts the hypervolume indicator of the current Pareto front to compute a difference measure.

With the improvement function in place, before computing the expected improvement we need to take a step back to the Gaussian Processes that are modelling our objective functions.
When predicting the expected improvement for two objective functions, there can be no correlation between the random variables of these separate objective function value predictions. Namely, want both objective functions to be modelled independently from each other. To achieve this, we will consider one Gaussian Process per objective function.
By doing this we can assume that there is no correlation between random variables from different Gaussian processes.

Considering the representation of a vector x by a Gaussian Process as a random two-dimensional variable,conditioned on previous information $X, F(X)$ where $F(X) = y^{(1)}, ...y^{(t-1)}$. This random variable has a distribution associated with it, so we can compute the *conditional probability density function* (PDF) of the joint probability $P(x|X, F(X)$ this gives us the probability of x falling within a range of values y.

When we then integrate the improvement $I_m(y, F_{nd,t-1}, r)$ of vector x and multiply this by the PDF associated with x we can compute the probability of a certain improvement value:

$$EI = \int_{y \in \mathbb{R}^d} I_m(y, F_{nd,t-1}, r) PDF_{x|X,F(X)}(y) dx$$

Computing the PDF of a vector x from a Gaussian process can be accomplished by using the mean μ and standard deviation σ of the predictive distribution of x and the GP prior X.

3.2.1 EHVI Efficiency

It is essential that the computation of the EHVI is fast and exact, so I will discuss some measures and definitions that the EHVI algorithm utilizes to achieve this.
One of these measures is a function $\Delta(y, F_{nd}, r)$ that computes the subset of vectors $\{y_1, ..., y_n\}$ which are exclusively dominated by objective vector $y \in \mathbb{R}^d$ and not by any elements in the Pareto front F_{nd}, and that dominate the reference point.
A formal definition of this function Δ:

$$\Delta(y, F_{nd}, r) = \lambda_m\{z \in \mathbb{R}^d | y \prec z \text{ and } z \prec r \text{ and } \nexists q \in F_{nd} : q \prec z\}$$

The introduction of this function Δ allows us to only evaluate for the vectors $\{y_1, ..., y_n\}$, which have a definite positive improvement value, and significantly reduce the computation needed.

For computing the hypervolume indicator, some improvement measures have been implemented as well. Firstly, we sort the Pareto front by the first coordinate and then define rectangles or strips over which the integration will take place.
Consider n points in the Pareto front F_{nd}, if we split the objective space into $n + 1$ disjoint strips $S_1, ..., S_{n+1}$ and introduce two sentinels for bounding the outermost rectangles $S1$ and Sn:
$y^{(0)} = (r_1, -\infty)$ and $y^{(n+1)} = (-\infty, r_2)$ we can formally define these strips:

$$S_i = \left(\begin{pmatrix} y_1^{(i)} \\ -\infty \end{pmatrix} \quad , \quad \begin{pmatrix} y_1^{(i-1)} \\ y_2^{(i)} \end{pmatrix} \right)$$

Now the improvement I_2 of a two-dimensional objective vector y becomes:

$$I_2(y, F_n, r) = \sum_{i=1}^{n+1} \lambda_2[S_i \cup \Delta(y)]$$

The introduction of these strips allows taking calculating the area ($\hat{I}_2$ is a two-dimensional area measure) of the intersection of a strip S_i and the subset of vectors $\Delta(y)$ found earlier reducing the integration region significantly.This intersection is non-empty if and only if $y = (y_1, y_2)$ lies within the rectangle with lower-left corner $(-\infty, -\infty)$ and upper right corner $(y_1^{(i-1)}, y_2^{(i)}))$.
The improvement is calculated by taking the sum of these area measures.

A description of the full mathematical rewriting of the EHVI formula can be found in the paper *A Multicriteria Generalization of Bayesian Global Optimization* by *Emmerich, Yang, Deutz,Wang and Fonseca* [1].

We will consider the final shape the EHVI formula takes on:

$$EHVI(\mu, \sigma, F_{nd}, r) =$$
$$\sum_{i=1}^{n+1} (\Psi(y_1^{(i-1)}, y_1^{(i-1)}, \mu_1, \sigma_1) - \Psi(y_1^{(i-1)}, y_1^{(i)}, \mu_1, \sigma_1)) * \Psi(y_2^{(i)}, y_2^{(i)}, \mu_2, \sigma_2)$$

in which $\Psi(a, b, \mu, \sigma)$ is defined by :

$$\Psi = \sigma * PDF(\tfrac{a-\mu}{\sigma} + (a - \mu) * CDF(\tfrac{b-\mu}{\sigma})$$

and $PDF(x)$ and $CDF(x)$ are given by $\frac{1}{\sqrt{2\pi}} * exp(\frac{-x^2}{2})$, and $\frac{1}{2} * (1 + erf(\sqrt{2})$ respectively.

Version of August 5, 2020– Created August 5, 2020 - 10:57

3.2.2 Finding the Maximizer of the EHVI Function

Now that we have defined the necessary functions and methods to compute the *expected hypervolume improvement* for a vector $x \in \mathbb{R}^2$, associated with a random variable and a prior distribution in a Gaussian Process, we can focus on the next task: finding the maximizer of the EHVI function.
The problem of finding the vector x that returns the highest EHVI value is essentially finding maximizer of the EHVI function. This itself an optimization problem where the objective function is the EHVI function.
It is also where the strength of this algorithm comes from since an evaluation of the EHVI function is cheap compared to an evaluation of the objective function for specific problems where Bayesian Global Optimization is often used.

When searching for the maximizer of a function, many different optimization techniques can be implemented; like grid-search which is computationally expensive when the optimization problem has a high dimensionality; or we can use another optimization technique to optimize the EHVI function.

Since we are dealing with vector input x, creating a uniform grid is not computationally efficient. Deciding on which optimizer would fit this problem well, there are a few requirements to keep in mind:

1. The optimizer should be capable of more exploration-based and more exploitation-based solution searching.
2. The optimizer should be computationally feasible; relatively fast convergence.

Any evolutionary optimization technique like *genetic algorithms*, *N+1ES*, *particle swarm optimization* algorithms, or other global optimization techniques could offer a good solution to the problem of finding the maximizer of the EHVI function. Chapter 3, *ESA's Pygmo Library*,will cover a comparison of the performance of a few of these algorithms.

3.3 Implementing EHVI in Python

For the Python implementation of the EHVI algorithm, the final formula for the EHVI function [1] as described in the previous section is used in the algorithm. Since we are only interested in solving bi-objective optimization problems, the dimensionality $m = 2$.

The algorithm can handle objective functions with multi-dimensional input, meaning the input x can be a vector $x = \{x_1, ...x_d\}$. However, this thesis will focus on optimization problems where x has two dimensions $x = \{x_1, x_2\}$.

The Gaussian Processes that act as the surrogate functions for the two objective functions are implemented using the Python library *SKlearn*, which has a method for Gaussian Process Regression. Using the default settings for these GP's the kernel function is the stationary *Radial Basis Function* (RBF) kernel which uses a *Euclidian* distance-based metric for correlations between the random variables in the GP.
The log-marginal likelihood algorithm *L-BFGS-B* performs the task of optimizing the hyperparameters of the kernel.

Running the algorithm, we start with some initial objective function evaluations $f_1(x_1), ..., f_1(x_n), f_2(x_1), ..., f_2(x_n)$ for some solution vectors $x_1, ..x_n$,the number of these initial points can be set by the user.
Having prior knowledge of the objective functions in the form of these evaluations is necessary for the functioning of our GPs as prior distributions of our objective functions, this can be considered our training data.

For the EHVI function, we declare some necessary variables that are used frequently by the PDF and CDF calculations to speed up computation, namely: $\sqrt{2}$, $\frac{1}{\sqrt{2\pi}}$.
For illustration a few lines of pseudo code for the EHVI function:

Algorithm 1 EHVI algorithm

function EHVI(P,r,μ_1,μ_2,σ_1,σ_2)
  $P \leftarrow sort(P)$ ▷ sort the Pareto front
  $A \leftarrow -\infty$ ▷ set sentinel value
  for $i = 0; i < n + 1; i++$ **do**
    $u1, u2, l1 \leftarrow S_i - boundaries$ ▷ define the boundaries of a strip
    $\Psi_1 \leftarrow$ EXIPSI(u1,u1,μ_1,σ_1) - EXIPSI(u1,l1,μ_1,σ_1) ▷ integration
    $\Psi_2 \leftarrow$ EXIPSI(u2,u2,μ_2,σ_2) - EXIPSI(u2,A,μ_2,σ_2) ▷ integration
    if $!S_{n+1}$ **then** ▷ if not the last strip
      $R \leftarrow remainder$
    end if
    $answer \leftarrow \Psi_1 1 * \Psi_2 + R$ ▷ final ehvi value
  end for
  return answer
end function
function EXIPSI(f_{max},cellcorner,μ,σ)
  $f_t \leftarrow f_{max} - \mu$
  $x_t \leftarrow \frac{cellcorner-\mu}{\sigma}$
  return $\sigma * PDF(x_t) + f_t * CDF(x_t)$
end function

3.3.1 Maximizing the EHVI Function

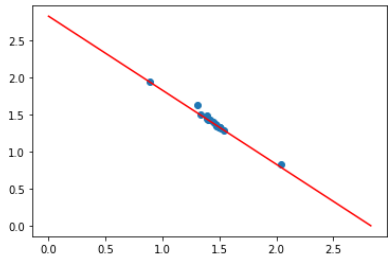
The pygmo library offers many optimization algorithms that can be used to optimize the EHVI function. Choosing the right algorithm for our problem is crucial for performance and convergence. The algorithms that are under consideration are:

- The *(N+1)-ES* simple evolutionary algorithm mutates its best solutions into one offspring each generation.
- The *SGA* simple genetic algorithm uses mutation, crossover, and selection techniques to create new solutions every generation.
- The *PSO* particle swarm optimization algorithm iteratively tries to improve a candidate solution.
- The *GACO* ant colony optimization algorithm locates optimal solutions by moving through the search space.

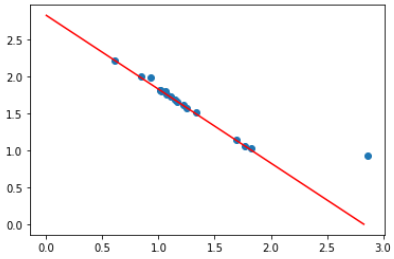
Version of August 5, 2020– Created August 5, 2020 - 10:57

- The *Simulated Annealing* algorithm which implements a temperature and cooling schedule which affects the acceptance of worse solutions.

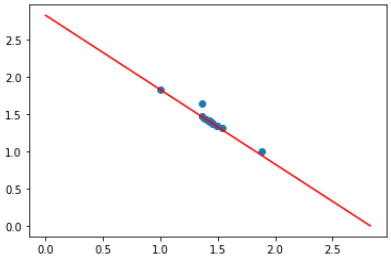
Running these algorithms in a simple test suite where the EHVI algorithm minimizes $f1 = ||x - 1||$, $f2 = ||x + 1||$ with the Pareto front represented by the line segment between $(2\sqrt{2}, 0)$ and $(0, 2\sqrt{2})$ we get the following results for the different optimizers of the EHVI function:



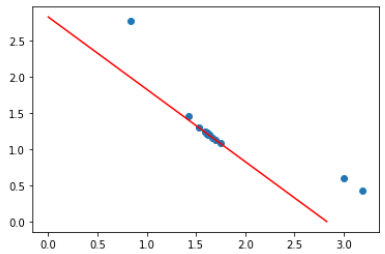
(a) *(N+1)-ES algorithm*



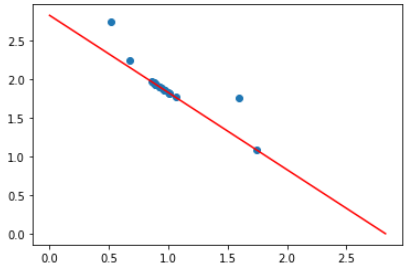
(b) *SGA algorithm*



(c) *PSO algorithm*



(d) *GACO algorithm*



(e) *Simulated Annealing algorithm*

For all the different algorithms, the EHVI algorithm seems to converge well on the optimal Pareto front. The distribution of the final non-dominated front is not ideal, but this does not seem to originate from the choice of optimizer for the EHVI function.
Going forward, we implement the SGA (simple genetic algorithm) as our optimizer.

For an extensive discussion of pygmo and its functionalities, I refer you to the next chapter.

With the optimizer in place, we can now find the maximizer of the EHVI algorithm. This maximizer is associated with vector x which is the next input to be evaluated by the objective functions resulting in a new point $y = (y_1, y_2)$ in the objective space. Adding vector x and y to the Gaussian process as training data results in a posterior distribution that better describes the objective functions for further prediction.
Pseudocode for the maximization loop:

Algorithm 2 Maximizing the EHVI function

```
1: function MAXIMIZE_EHVI(GP1,GP2,P,r)
2:     function MIN_OBJ_FUNC(x)
3:         μ1, σ1 ← GP1.predict(x)        ▷ calculate from Gaussian Process
4:         μ2, σ2 ← GP2.predict(x)        ▷ calculate from Gaussian Process
5:         ehvi_res ← EHVI(P, r, μ1, σ1, μ2, σ2)        ▷ run ehvi calculation
6:         return -EHVI_res                              ▷ for minimization
7:     end function
8:     X_new ← SGA(MIN_OBJ_FUNC(x : x))        ▷ optimizing the function
```

3.3.2 Main EHVI Loop

The main loop in the program describes the actions the algorithm performs every iteration. This loop consists of training the GPs on input and generating the current best non-dominated front by extracting it from the collection of solutions evaluated thus far. It then proposes a new solution vector x by optimizing the EHVI function using the simple genetic algorithm. Pseudocode for the main optimization loop:

Algorithm 3 Main optimization loop

function EVOLVE(X_{init})
  $X \leftarrow X_{init}$ ▷ set of initial vectors
  $Y \leftarrow (y_1, y_2)$ ▷ initial function evaluation for X
  for number of iterations **do**
    $GP1.fit(X, Y)$ ▷ train the GPs on X and Y
    $GP2.fit(X, Y)$
    $P \leftarrow non\_dominated\_front(Y)$
    $R \leftarrow P.ref\_point()$ ▷ compute r
    $X_{new} \leftarrow$ MAXIMIZE_EHVI($GP1, GP2, P, r$)
    $X \leftarrow X \cup X_{new}$ ▷ add the new point
    $Y \leftarrow Y \cup f(X_{new})$ ▷ evaluate new point
  end for
  return P

We now have all the tools needed to start applying the EHVI algorithm to multi-objective optimization problems with dimensionality $d = 2$.
The next section will have some graphs and plots of a numerical example to get a better understanding of the EHVI algorithm described above.

3.3.3 A Numerical Example

The numerical example implemented is a recreation of an example from the paper *A Multicriteria Generalization of Bayesian Global Optimization* by *Michael Emmerich, Kaifeng Yang, Andre Deutz, Hao Wang and Carlos M. Fonseca* [1]. The objective functions to be minimized were already introduced in the previous section as well:

$$f_1 = ||x - 1||, f_2 = ||x + 1|| \text{ with } x \in [-2, 2]X[-2, 2]$$

The optimal Pareto front is given by the line segment between $(2\sqrt{2}, 0)$ and $(0, 2\sqrt{2})$. First, the GPs are initiated with 15 random solution vectors and then we run the algorithm for 20 iterations.
The interesting metrics are then: the distribution of the mean values μ_1, μ_2 for each GP as well as the variances σ_1, σ_2. Furthermore, the distribution of the EHVI values might be the most informative metric to gain an insight into the algorithm. Using a uniform grid, we will evaluate these values for every vector $x = \{x_1, x_2\}$ in its domain. Plotting the values with x_1 on the x-axis and x_2 on the y-axis gives us the following graphs:

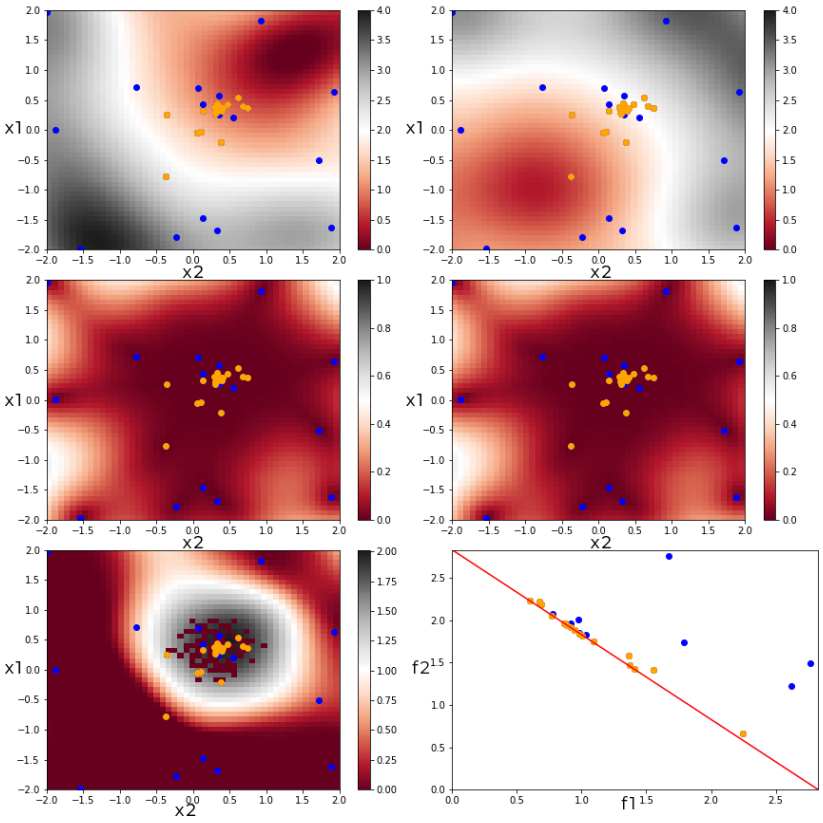


Figure 3.3: *Top: distribution of μ_1 and μ_2 Middle: distribution of σ_1 and σ_2 Bottom-left: distribution of the EHVI, Bottom-right: Objective space*

The lower right plot is of the solutions evaluated, the non-dominated points and the optimal Pareto front. The distributions of the mean and variance are what you might expect given the solutions. Interesting is the plot of the EHVI values; these values seem to be skewed very much towards exploitation since the highest values are centered in the middle of the current Pareto front. This unexpected result will be discussed further in Chapter 6 *Discussion*.

Chapter 4

ESA's PyGMO Library

PyGMO is a scientific library for (massively parallel) optimization, developed by the European Space Agency ESA. This library is used for evolving interplanetary spacecraft trajectories, designs for spacecraft parts and more. The PyGMO library is not limited to optimizing trajectories; it is built around the idea of providing a unified interface to optimization problems and algorithms that solve them and to make it easier to deploy them in massively parallel environments.

Within the library, there are many variants of optimization problems: problems that are constrained or unconstrained, single objective or multi-objective problems, continuous and integer problems, stochastic and deterministic problems. There are also many different algorithms available that solve these problems: local solvers, meta-heuristics, single and multi-objective algorithms.

The strength of PyGMO is the fact that in addition to the optimization problems and algorithms available, you can create a user-defined problem (UDP) and a user-defined algorithm (UDA) adhering to the format specified by PyGMO. You can implement algorithms and give them access to the resources that the library provides: optimization problems, parallel optimization, bench-marking, and utility functions.

The first step is implementing the EHVI algorithm as a UDA in the PyGMO API. This implementation allows us to benchmark it on specific multi-objective optimization problems, against similar algorithms that can solve multi-objective optimization problems which are already available in the library.

The fundamental principles of the PyGMO API are several different classes which together instantiate optimization problems, specify a population of candidate solutions to those problems, and evolve these populations using optimization algorithms.

4.0.1 Problem Class

The problem class implements a generic optimization problem; a user-defined problem can be passed as a PyGMO algorithm class to the problem class, which provides a single unified interface for all optimization problems.
An optimization problem can be created with the name of a user-defined optimization problem and its relevant parameters, for example:

```
prob = pygmo.problem(pygmo.rosenbrock(dim = 10))
```

This line of code creates a ten-dimensional variant of the*Rosenbrock* optimization problem. Through the interface of the problem class, we can now access information about this problem.

4.0.2 Algorithm Class

The algorithm class implements a generic optimization algorithm; a user-defined algorithm can be passed to this class.
A PyGMO algorithm is created using the name, and relevant parameters of a user-defined algorithm, these parameters are specified in the documentation of each algorithm. Since many optimization algorithms differ significantly in their approach, the parameters of the algorithms also vary greatly.

```
algo = pygmo.algorithm(pygmo.cmaes(gen = 100, sigma0 = 0.3))
```

This line of code creates an instance of the *Covariance Matrix Adaptation Evolutionary Strategy* (CMA-ES). Like with the problem class, we can access information about an algorithm.
Evolving a population is performed by calling the evolve method of the algorithm class. It takes a population as a parameter and returns the evolved population.

4.0.3 Population Class

The population class is a collection of candidate solutions for an optimization problem. It contains one problem as specified above and a number of decision vectors with associated fitness vectors. A solution is defined by a unique ID to allow the tracking of this individual.

Having specified a problem and an algorithm as we did earlier for the *Rosenbrock* problem and the CMA-ES algorithm, we can now create a population:

```
pop = pygmo.population(prob, size = 10)
```

Which constructs a set of ten solutions in the *Rosenbrock* problem, causing ten fitness evaluations. We can inspect the full population at any time or extract specific information from it like the decision vector or fitness vector values. We can add a solution to a population as well using:

```
pop.push_back(x)
```

The associated fitness vector is then automatically calculated by the associated problem and is added to the information of this solution.

With these three classes as building blocks, we are now able to implement the EHVI algorithm a user-defined algorithm.

4.1 EHVI as User-defined Algorithm

Implementing the *expected hypervolume improvement* algorithm as a user-defined algorithm in PyGMO is a straightforward task; we need to define elements of the EHVI algorithm as elements of the PyGMO API.

First, we need to define a class *py_bgo_ehvi* with all elements that make up the EHVI algorithm. We specify an init function with the algorithm's parameters:

```
def __init__(dim, sa_maxiter)
```

This function also imports the necessary dependencies and libraries as well as instantiating the Gaussian processes and other variables used by the different methods of the class.

The main optimization loop of the EHVI algorithm is renamed to the *evolve* method of a PyGMO algorithm, as described before. From the PyGMO population that is passed as a parameter, we extract the solution vectors as the initial points X and the associated fitness values as the initial objective function evaluations Y:

```
X = pop.get_x()
Y = pop.get_f()
```

The bounds of the problem can be extracted by:

```
prob.get_lb() and prob.get_ub()
```

In addition, there are several utility functions that PyGMO offers as well:

```
P = pygmo.non_dominated_front_2d(Y)
```

This function extracts the Pareto front from a set of fitness vectors.

```
R = pygmo.nadir(P)
```

The nadir function computes a valid reference point by taking the largest values in the Pareto front and requiring this reference point to be slightly bigger than these values.

```
pop.push_back(new_point)
```

The push_back functionality stores a new point to our population and automatically computes the fitness value for this point.

4.2 The EHVI Function as User-defined Problem

Instead of using external libraries to compute the maximizer for the EHVI function, we implement the function that calculates the *expected hypervolume improvement* as a user-defined problem in PyGMO. With this implementation, we can find the maximizer of the EHVI function using existing optimization algorithms.
This implementation is straightforward; defining the function as a PyGMO problem class; its fitness function returns the EHVI value for the input x. A population is initiated with some random solutions in the search space (within the bounds of the parent optimization problem) with their respective fitness values, which are calculated by the EHVI function.

Other utility functionalities that PyGMO offers like plotting and measures to compute the convergence and performance of an algorithm are introduced in the next chapter: *Bench-marking* where we will compare the EHVI algorithm to other multi-objective optimization algorithms like the *Non-dominated Sorting Genetic Algorithm* (NSGA2).

Chapter 5

Bench-marking the EHVI Algorithm

This chapter will discuss a method for bench-marking algorithms using the PyGMO scientific library, and the set up of the bench-mark and the indicators that are used to determine the performance of the EHVI algorithm. The EHVI algorithm's performance will be bench-marked against other similar multi-objective optimization algorithms available in the PyGMO library.

5.1 ZDT Test Suite

There are many optimization problems available in the PyGMO library. For this bench-mark experiment, we will be using the ZDT test suite. The ZDT test problems get their name from the three individuals that developed these problems: *Zitzler, Deb and Thiele*.
The optimization test suite was designed for the purpose of bench-marking optimization algorithms.
In their paper, the authors describe a set of six scalable, unconstrained two-objective optimization problems. For each of the six ZDT optimization problems, the optimal Pareto front is known so for any non-dominated front an algorithm produces we can measure its distance to this optimal front.

The objective functions for all ZDT problems are specified as the same combination of thee functions f_1, g, h:

$$\begin{aligned} &Minimize\, T(x) = (f_1(x_1), f_2(x)) \\ &subject\, to\, f_2(x) = g(x_2, ..., x_m) * h(f_1(x_1), g(x_2, ..., x_m)), \\ &\quad where\, x = (x_1, ..., x_m) \end{aligned} \tag{5.1}$$

The problems differ in their$f1, g, h$ functions as well as in the number of elements m in the input vector x.
Each problem in the ZDT suite introduces a particular difficulty for multi-objective optimization and converging on the optimal Pareto front.

We will only be testing the EHVI algorithm on the first three problems in the ZDT test suite, so for the problem descriptions for problems 4, 5, and 6, we don't go into as much detail as for the first three. Some specifications of the six ZDT problems:

ZDT1 is a box-constrained continuous $n > 1$ multi-objective optimization problem. It has a convex optimal Pareto front specified by:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, ..., x_m) &= 1 + 9 * \sum_{i=2}^{m} \frac{x_i}{m-1} \\ h(f_1, g) &= 1 - \sqrt{\frac{f1}{g}} \end{aligned} \tag{5.2}$$

In this problem the number of variables $m = 30$, and $x_i \in \{0, 1\}$. The optimal Pareto front is found for $g(x) = 1$.

ZDT2 is the counterpart to the ZDT1 problem with a non-convex optimal Pareto front specified by:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, ..., x_m) &= 1 + 9 * \sum_{i=2}^{m} \frac{x_i}{m-1} \\ h(f_1, g) &= 1 - (\frac{f_1}{g})^2 \end{aligned} \tag{5.3}$$

Again $m = 30$ and $x_i \in \{0, 1\}$ and the optimal Pareto front is found for $g(x) = 1$.

ZDT3 represents discreteness in optimization problems, meaning that the optimal Pareto front consists of multiple non-contiguous convex parts specified by:

$$\begin{aligned} f_1(x_1) &= x_1 \\ g(x_2, ..., x_m) &= 1 + 9 * \sum_{i=2}^{m} \frac{x_i}{m-1} \\ h(f_1, g) &= 1 - \sqrt{\frac{f_1}{g}} - \frac{f_1}{g} Sin(10\pi * f_1) \end{aligned} \tag{5.4}$$

As for the first two problems $m = 30$, $x_i \in \{0, 1\}$. But introducing the sine function causes the Pareto front to be discontinuous without creating a discontinuity in the parameter space. So the problem remains a continuous problem.

ZDT4 has $21e9$ local optimal Pareto fronts and tests optimization algorithms ability to deal with multimodality. With $m = 10$, $x_i \in \{0, 1\}$, and $x_2, ...x_m \in \{-5, 5\}$, with the global optimal Pareto front for $g(x) = 1$, and the best local Pareto front for $g(x) = 1.25$.

ZDT5 is a deceptive optimization problem where x_i represents a binary string. With $m = 11$, $x1 \in \{0, 1\}^{30}$, and $x_2, ...x_m \in \{0, 1\}^5$, with the optimal Pareto fronts for solutions where $g(x) = 11$.

ZDT6 implements the difficulties of non-uniformity of the search space; the optimal Pareto fronts are distributed non-uniformly along the global Pareto front; the density of solutions is at its lowest near the optimal Pareto front and higher further away from the front. With $m = 10$, $x_i \in \{0, 1\}$, and the optimal Pareto front where $g(x) = 1$.

For a more detailed description of the ZDT problems, we refer you to the original publication: *Comparison of Multi-objective Evolutionary Algorithms: Empirical Results by Zitzler, Deb and Thiele* [2].

5.2 Measures of Performance

When comparing optimization algorithms, there is always a notion of performance. The performance indicators for single-objective problems and algorithms are less complex and easier to compute than for multi-objective problems and algorithms.
We define the measures of performance for multi-objective optimization and in particular the ZDT problem suite here:

- The distance of the non-dominated front computed by an optimization algorithm to the optimal Pareto front. This distance should be minimized.

- The non-dominated front should be well distributed (even uniformly distributed). This uniform distribution can be achieved by comparing the hypervolume indicators of the non-dominated front with that of the optimal Pareto front or that of other algorithms when benchmarking and comparing different algorithms.

- For each of the objective functions, the nondominated front should cover a wide range of values. This measure should be maximized.

- In the particular case of Bayesian Global Optimization, since its strength lies in the comparatively low number of objective function evaluations, we are also comparing the number of these evaluations each algorithm performs.

5.3 Other Algorithms in the Bench-mark

Pygmo offers a significant collection of multi-objective optimization algorithms which can be used to bench-mark the EHVI algorithm. From this collection three evolutionary or genetic algorithms we're chosen:
a *Multi-objective Hypervolume-based Ant Colony Optimizer* (MACO)[3], a *Non dominated Sorting Genetic Algorithm* (NSGA2)[5], and a *Multi-Objective Evolutionary Algorithm by Decomposition* (MOEAD)[4] on the same problems as our Expected Hypervolume Improvement (EHVI) algorithm.

5.4 Bench-mark Set-up

The bench-mark will make use the first three optimization problems from the *Zitzler, Deb and Thiele* (ZDT) test problem suite available in PyGMO. These problems are often used for benchmarking multi-objective optimization algorithms on two objectives, by introducing difficulties that might arise in optimization problems, as described in the previous sections.

We will be running four optimization algorithms (EVHI, MACO, NSGA2 and MOEAD) for different population sizes $[32, 64, 128]$ (since the NSGA2 algorithm's population size has the requirement of being a multiple of 4). For each population size, we will be running the algorithms three times with a different fixed seed each time.
Important to note is that for the performance measures to be accurate, the four different algorithms are run on the same initial populations.
We achieve this by instantiating a population with a fixed seed as a parameter.

Since the optimal Pareto front is known for the ZDT problems, we use the distance of the non-dominated front computed by an algorithm to this optimal front as the first measure of convergence and performance. This metric allows us to check how well a non-dominated front covers the optimal Pareto front of a problem. In PyGMO, the ZDT suite implements this metric as *p_distance*, which is a method you can call on the non-dominated front of the final evolved population.

We will also be computing the hypervolume indicator for the final populations and compare them to get an indication of the spread and distribution of the non dominated front of the different algorithms.
If the hypervolume indicator of one population is lower than that of another population, evolved by a different algorithm, it indicates that the points in the non-dominated front are not as well distributed, i.e., the points are more concentrated in one region of the front.
The hypervolume in pygmo can be computed by calling the *hypervolume* method on a population.

Since the three algorithms NGSA2, MACO, and MOEAD, use a generations parameter, which is the number of generations (iterations) the algorithm performs, we will choose this to be $gen = 250$. The EHVI algorithm uses a different iteration parameter, for the sake of computation time, we set this parameter to a manageable number. Instead of iterating until we

have a Pareto front with the same amount of solutions as the initial population size. Especially for population sizes 64 and 128, this process can take a very long time. The iteration parameter is set to iterate until there are 32 solutions in the final Pareto front. This way, the EHVI algorithm is more in line with the other algorithms that keep their population size constant. Although the EHVI algorithm will run for more iterations than if the iteration parameter was set to a specific value, we do maximize the *p_distance* metric and distribution of the front this way.

This choice for the iteration parameter does affect the results for greater population sizes $[64, 128]$, the other algorithms will have more points in their final non-dominated front which may contribute to a better-distributed front for these algorithms compared to that of the EHVI algorithm.
We will calculate the average of the *p_distance* and *hypervolume* indicator values for each population size over the three runs.
The results will be plotted in the fitness space, showing the initial population and the non-dominated fronts found for each algorithm and population size.

With this setup and the code that implements this benchmark, we are now in a position to run this experiment and calculate the performance metrics for the four different multi-objective optimization algorithms. The results and conclusion thereof will be discussed in chapters 6 and 7: *Results* and *Discussion*.

Chapter 6

Results

The experiment that will be performed is set up as described in the previous chapter. The experiment consists of running multiple optimization algorithms on the first three problems of the ZDT test suite.

The MOEAD, MACO and NSGA2 algorithm will each perform 250 generations of optimization whereas the EHVI algorithm will run until there are 32 solutions in the Pareto front.
We expect the EHVI algorithm to converge on the Pareto front in a significantly less amount of objective function evaluations, which is the strength of Bayesian Global Optimization compared to other optimization techniques.

However, the distribution of the Pareto front computed by the EHVI algorithm is expected to be less uniform than that of the other algorithms, since we've seen in testing and developing this algorithm that the values that the EHVI calculation produces tend to be higher closer to other non-dominated points. The exploration element of the algorithm might be a bit lacking in comparison to its exploitation element, which seems to perform very well.
In addition, the fact that the other algorithms generate more solutions in their computation of the Pareto front, but running the EHVI algorithm until it has 64 or even 128 solutions in the non-dominated front is computationally expensive. The computation time stems from the optimization of the EHVI function each iteration using an SGA (Simple Genetic Algorithm).
Why this is the case, we will touch on later in chapter 7: *Discussion*.

The first problem in the ZDT suite has a concave Pareto front, and the results of nine runs are shown in the graph below:

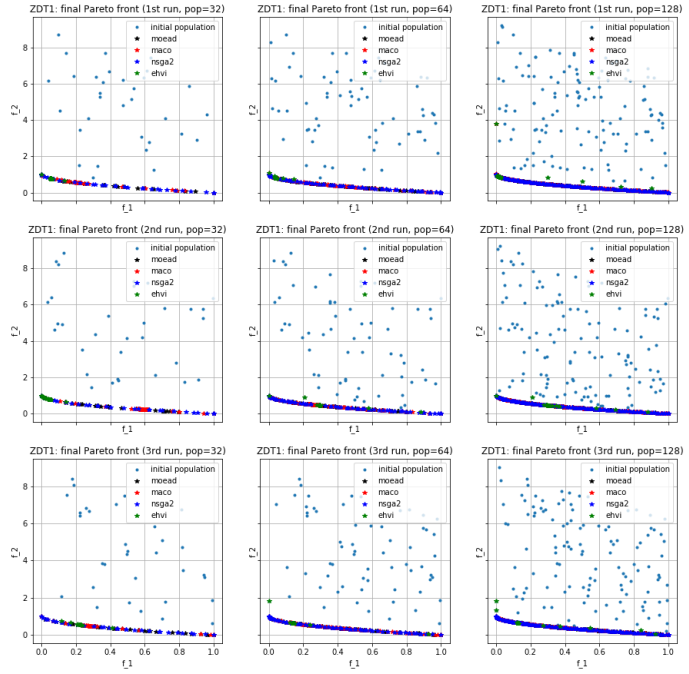


Figure 6.1: *MACO, MOEAD, NSGA2, and EHVI on the ZDT1 problem*

The initial population is plotted alongside the final non-dominated front of the different algorithms.
You can see that the non-dominated front of the EHVI algorithm (in green) is not as uniformly distributed as that of the other algorithms. The points tend to be more clustered together, whereas, with the other optimization algorithms, the points in the Pareto front have a spread across the length of the optimal front. Specifically, for population sizes 64 and 128, this seems to be the case since the EHVI algorithm only produces 32 points.

Looking at the mean performance and convergence metrics, over the different runs, that we introduced in the previous chapter (the hypervolume indicator and the P_distance metric):
We can see that there is a significant difference in hypervolume between the EHVI algorithm and the other three algorithms, which can be attributed

ZDT1	Hypervolume Indicator	p_distance
MOEAD [32,64,128]	[13.24, 13.32, 14.08]	[8.70e-05, 4.55e-05, 2.01e-05]
MACO [32,64,128]	[13.21, 13,32, 14.08]	[4.14e-03, 4.22e-03, 2.64e-02]
NSGA2 [32,64,128]	[13.25, 13.32, 14.08]	[7.28e-06, 3.71e-06, 2.34e-07]
EHVI [32,64,128]	[12.42, 12.85, 13.80]	[3.43e-03, 2.10e-02, 7.40e-02]

to the difference in the distribution of points on the Pareto front.
The convergence metric *p_distance* shows us that the EHVI algorithm converges well on the optimal Pareto front known for the ZDT1 problem. The MACO algorithm only slightly outperforms it, but the MOEAD and NSGA2 algorithm do tend to converge significantly better overall.

For the second problem in the ZDT test suite, we get the following plots and performance metrics:

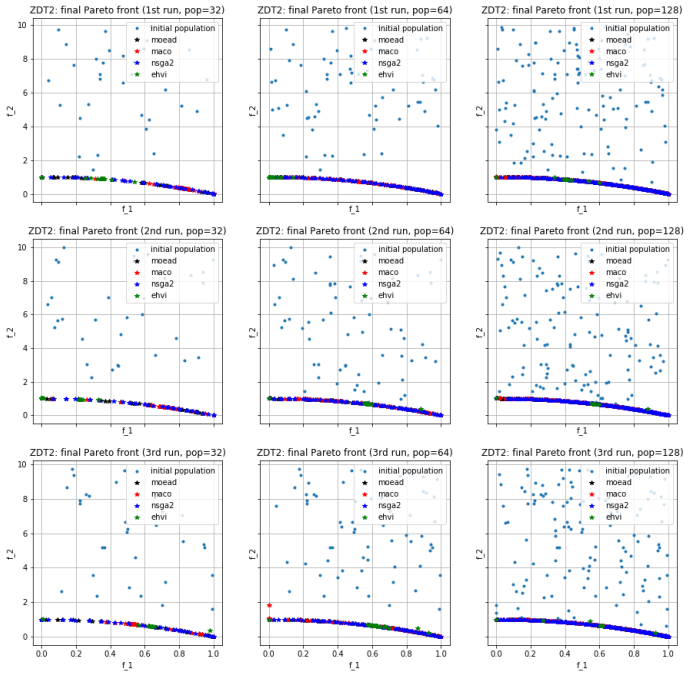


Figure 6.2: *MACO, MOEAD, NSGA2, and EHVI on the ZDT2 problem*

ZDT2	Hypervolume Indicator	p_distance
MOEAD [32,64,128]	[14.70, 14.78, 14.80]	[3.54e-05, 2.08e-05, 8.61e-06]
MACO [32,64,128]	[14.68, 14.78, 14.80]	[1.06e-03, 1.41e-02, 5.68e-04]
NSGA2 [32,64,128]	[14.70, 14.78, 14.81]	[8.48e-06, 2.02e-12, 2.35e-12]
EHVI [32,64,128]	[14.17, 14.34, 14.46]	[2.94e-03, 4.81e-03, 4.95e-03]

For clarity, we've also added a graph where only the EHVI solutions are plotted to get a better indication of the final Pareto front that it has been computed.

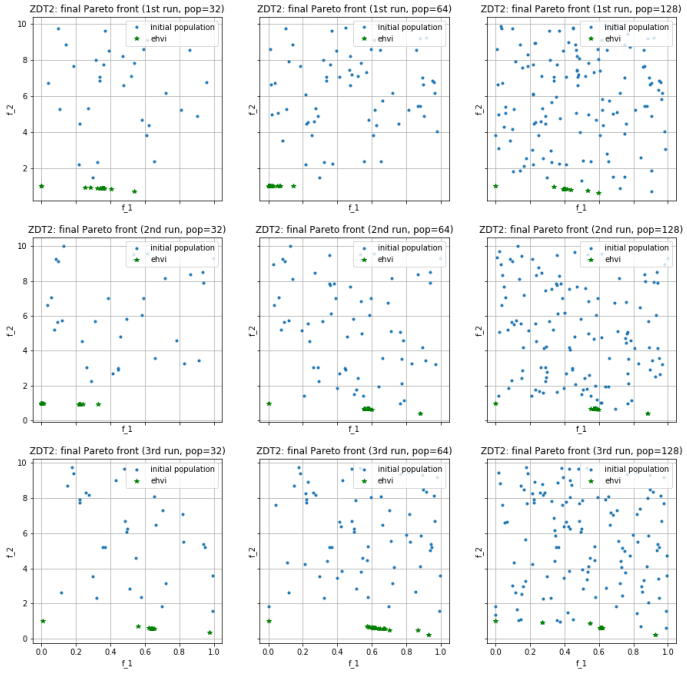


Figure 6.3: *EHVI on the ZDT2 problem*

The distribution of the non-dominated front computed by the EHVI algorithm seems to have improved slightly in comparison to the ZDT1 problem. Its hypervolume indicator in more on par with that of the other algorithms. However, in terms of the convergence, *p_distance* metric the EHVI algorithm outperformed by the MOEAD and NSGA2 algorithms but remains on par with the MACO algorithm.

The third problem in the ZDT suite is a particular problem since it implements a non-contiguous Pareto front and tests the algorithm's ability to deal with such a front. Like with the other ZDT problems we can expect the non-dominated front of the EHVI algorithm to be less uniformly distributed, this can result in the algorithm not being able to deal with a non-contiguous Pareto front as well as the other algorithms.

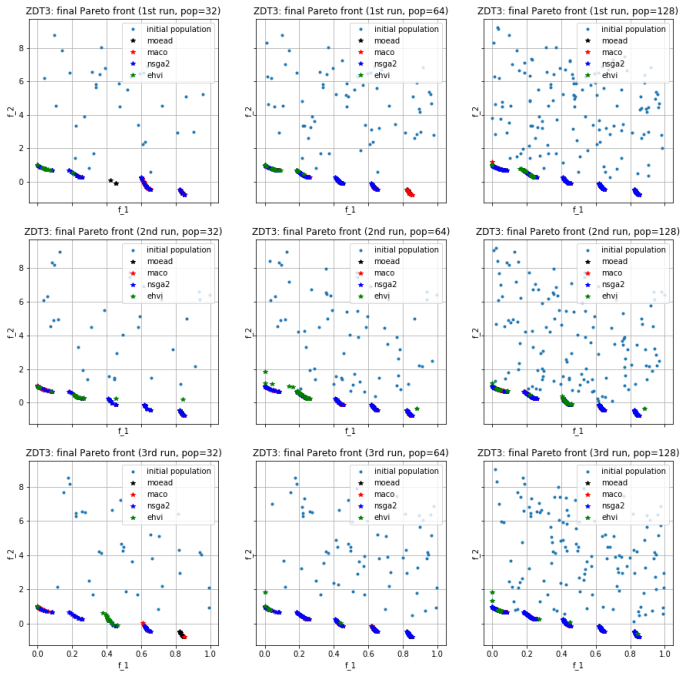


Figure 6.4: *MACO, MOEAD, NSGA2, and EHVI on the ZDT3 problem*

ZDT3	Hypervolume Indicator	p_distance
MOEAD [32,64,128]	[8.71, 8.78, 9.21]	[2.68e-04, 1.25e-04, 6.40e-05]
MACO [32,64,128]	[8.63, 8.79, 9.22]	[8.30e-03, 5.80e-04, 3.07e-03]
NSGA2 [32,64,128]	[8.68, 8.78, 9.22]	[5.53e-11, 4.51e-13, 3.99e-14]
EHVI [32,64,128]	[8.28, 8.34, 8.90]	[3.11e-02, 3.93e-02, 3.12e-02]

Indeed we can see that the Pareto front computed by the EHVI algorithm is centered around one or two sections of the optimal front. Still, not all algorithms seem able to deal with a non-contiguous front either. Especially

in the computation with a population size of 32, this is clear.
The performance metrics show this as well; all other algorithms outperform both the hypervolume indicator and *p_distance* metric of the EHVI algorithm.

But the strength of the EHVI algorithm comes from the number of objective function evaluations it performs. For the ZDT1 problem, these evaluations are:

ZDT1	Objective Function Evaluations
MOEAD	32128
MACO	32128
NSGA2	32128
EHVI	172

Because the MOEAD, NSGA2 and MACO algorithms have similar inner workings and the same number of generations they perform the number of function evaluations is the same for these algorithms. The EHVI algorithm performs more than two orders of magnitude less objective function evaluations than the other algorithms, but still yielding comparative results. We will discuss these results further in the next chapter: Discussion.

Chapter 7

Discussion and Future

We will now discuss the results presented in the previous chapter and try to gain insight into why the EHVI algorithm seems to choose exploitation over exploration and what the future of this algorithm is within ESA's pygmo library.

7.1 Discussion of the Results

Although the EHVI algorithm performs worse overall on the first three problems in the ZDT test suite, it does hold its own against the professionally implemented algorithms available in the PyGMO library. Moreover, it might compute a Pareto front that is less uniformly distributed than the others. Still, it does so in more than two orders of magnitude less objective function evaluations, which is the strength of Bayesian Global Optimization. The ZDT problems we looked at are not particularly problems where one might choose BGO over other optimization techniques. I hypothesize that a problem with computationally expensive objective function evaluations the EHVI algorithm will outperform the other algorithms.
For the future, testing on a problem where Bayesian Global Optimization performs best and comparing the performance of different algorithms against the EHVI algorithm is seems to be an interesting and important experiment.

7.1.1 The EHVI algorithm's Tendency for Exploitation Over Exploration

The EHVI algorithm relies heavily on the Gaussian process that acts as the posterior distribution of the objective functions. Choosing different kernels and their hyperparameters can significantly influence how the distribution of these functions is represented. Because we utilize an existing implementation for the GPs, namely the *Scikit* Gaussian Process library in Python, there is less control and flexibility in the creation and computation of the posterior distributions. We chose the stationary RBF kernel to generate the covariance matrices of our GPs, which implements the *length scale* parameter, which tells the GP how points are correlated about the Euclidean distance between them. This parameter is optimized internally by the Broyden-Fletcher-Goldfarb-Shanno algorithm (BFGS).

Lack of insight into this hyperparameter can result in points becoming correlated too much if they are close together. This correlation is represented in the covariance matrix and used by the GP to make predictions about new solutions as well as in the calculation of the expected hypervolume improvement. The choice of kernel and its hyperparameters could contribute to the EHVI function returning lower values for solutions further away from known solutions although their variance might be high. For the future, implementing a Gaussian Process, its kernel and the associated hyperparameters without relying on external Python libraries can be a good step to gain more control over the algorithm and manage more aspects of it yourself.

Another possible explanation for the exploitation favoured behavior of the EHVI algorithm is the choice of optimizer for finding the maximizer of the EHVI function. If this optimizer itself is more focussed on exploitation, the expected hypervolume improvement search space might not get searched as thoroughly as we might want, resulting in getting stuck in local optima. To find out if this is the case we'd need to do an extensive experiment with different optimizers and different values of their hyperparameters. For future implementation, finding the best optimizer for this problem is key to the algorithm's performance.

7.1.2 Future of the EHVI Algorithm in PyGMO

The PyGMO library is a professionally developed library that can be used by anyone who has an interest in the field of optimization. What this means is that all the algorithms implemented in this library are tested and debugged extensively.
The performance of the EHVI algorithm still lacks compared to the other algorithms, and it has not nearly been tested as much as the standards of ESA demand. The EHVI algorithm as of yet is not ready for an official integration within the PyGMO library, but I think a lot of progress has been made towards that end. Anyone who might continue this work can follow the topics discussed in this thesis and pinpoint the possible origin of the weaknesses and performance issues that presented themselves and start from there.

Bibliography

[1] Michael Emmerich, Kaifeng Yang, André Deutz, Hao Wang and Carlos M. Fonsenca. *A Multicriteria Generalization of Bayesian Global Optimization*. Multiobjective Optimization and Decision Analysis (MODA) Research Group LIACS, Faculty of Science, Leiden University, The Netherlands, 2015.

[2] Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele. *Comparison of Multiobjective Evolutionary Algorithms: Empirical Results*.

[3] Manuel López-Ibáñez. *Multi-objective Ant Colony Optimization*.

[4] Qingfu Zhang, Hui Li. *MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition*.

[5] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. *A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II*.

[6] C.E. Rasmussen C.K.I. Williams. *Gaussian Processes for Machine Learning*. Massachusetts Institute of Technology, 2006