UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Multi-Objective Parallelization of Efficient Global Optimization

by

Carla Grobler

A dissertation submitted in partial fulfillment
of the requirements of the degree

Master of Engineering

in the

Department of Mechanical and Aeronautical Engineering
Faculty of Engineering and Built Environment and Information Technology

University of Pretoria
South Africa

10 July 2016

To my loving husband, Paul Grobler.

Acknowledgments

During the process of research, there are often many people, whom without, the work would not have been possible. The following list of people have been instrumental in the completion of this research project, and for them, I am very grateful.

- I would like to thank the CSIR, who provided the funding a facilities for this project and also give special thanks to my supervisors at the CSIR, John Monk, and Kaven Naidoo. They offered help, expertise, and encouragement throughout the year.
- I would also like to thank my colleges at the CSIR, and would like to make special mention of Johan Heyns for intentionally following up on the progress of my project. I am very appreciative of the advice you have given me throughout this research.
- I would like to thank my office mates at the University of Pretoria, Paul, Amin, and Suzanne, for the time that we could share, and for your emotional support.
- To my supervisors at the University of Pretoria, Prof. Schalk Kok, and Dr. Nico Wilke, I have learnt an enormous amount from you. Thank you for your strong intellectual contributions and guidance, not only in the project but also in my life choices.
- I would like to thank my family for their support, and especially mention my parents, Onno and Marike Ubbink. I am a product of the guidance, opportunities, and encouragement you have given me.
- Paul, my husband, thank you for always being along for the ride. Thank you for your love, support, and encouragement. I love you.
- Leaving the best for last, I would like to thank the Creator of everything. You give meaning to everything I spend time doing. Let Your kingdom come.

Abstract

Design optimization is a subject field where mathematical algorithms are used to improve designs. Analyses of designs using computational techniques often require significant computing resources, and for these problems, an efficient optimization method is needed. Efficient Global Optimization (EGO), first proposed by Jones et al. [25] is an optimization method which aims to use few function evaluations when optimizing a design problem. In this study, we use a multi-objective strategy to parallelize EGO.

EGO is part of a set of algorithms called surrogate optimization methods. A set of initial designs are analyzed and then a response surface is fitted to the evaluated designs. In each iteration, EGO selects the set of design variables for which the next analysis will be performed. It makes this decision based on two opposing criteria. EGO will either decide to sample where the predicted objective function value is low, an exploitation approach, or where there is high uncertainty, an exploration approach.

In each iteration, the classical EGO only selects one design per iteration. This selected design vector is either a result of exploitation or exploration based on a measure referred to as maximum Expected Improvement (EI). However, the modern day computing environment is capable of running multiple different analyses in parallel. Thus, it would be advantageous if EGO would be able to select multiple designs to evaluate in each iteration.

In this research, we treat EGO's inherent selection criteria to either exploit or explore as a multi-objective optimization problem, since each criterion can be defined by a separate objective function. In general multi-objective optimization problems don't only have one solution, but a set of solutions called a Pareto optimal set. In our proposed strategy multiple designs from this Pareto optimal set are selected by EGO to be analyzed in the subsequent iteration. This proposed strategy is referred to as Simple Intuitive Multi-objective ParalLElization of Efficient Global Optimization (SIMPLE-EGO).

We start our study by investigating the behaviour of classical EGO. During each iteration of EGO, a new design is selected to be evaluated. This is performed by finding the maximum of the Expected Improvement (EI) function. Maximizing this function initially proved challenging. However, by exploiting information regarding the nature of the EI function, the maximization problem is simplified significantly, and the robustness of finding the maximum is enhanced. More importantly, solving this maximization problem robustly, dramatically improves the convergence behaviour once a local basin has been found.

We compare our SIMPLE-EGO method to a multi-objective optimization algorithm (EGO-MO) published by Feng et al. [16]. We first investigate the behaviour of EGO, EGO-MO, and SIMPLE-EGO. Thereafter the convergence performance of these methods is quantified.

As expected the parallelization of both SIMPLE-EGO and EGO-MO lead to faster convergence on a range of test functions compared to classical EGO, which only sampled one point per iteration. The convergence characteristics of SIMPLE-EGO and EGO-MO are also markedly different. We conclude with a discussion on the advantages and disadvantages of the investigated methods.

List of Symbols

Alpha Numeric Symbols

Symbol	**Description**
$\mathbf{1}$	Vectors of ones of length n_s
a	Scale Parameter used when selecting points on the Pareto front for EZ-EGO
b	Distance parameter used when selecting points on the Pareto front for EZ-EGO
c	Self contained variable used when selecting points on the Pareto front for EZ-EGO
E	Expected value
$\boldsymbol{f}$	Multi-objective cost function
f	Cost function
f_Y	Regression function
f_{Y_j}	Basis functions of the regression function f_Y (for Kriging this is often low order polynomial terms)
g	Inequality constraints in general optimization problem formulation
h	Equality constraint in general optimization problem formulation
k	Self contained parameter of Generalized Expected Improvement Function
k_Y	Number of basis functions for the underlying regression function f_Y
m	Number of inequality constraints in general optimization problem formulation
n	Number of dimensions
n_f	Number of cost functions
n_s	Number of evaluated designs
n_i	Number of points per iteration
N	Normal distribution
P	Probability
p_h	Smoothness parameter of the Kriging spacial correlation function

p	Number of equality constraints in general optimization problem formulation
$\boldsymbol{r}$	Spatial correlation vector
T	Self contained variable of Generalized Expected Improvement Function
$\boldsymbol{R}$	Spatial correlation matrix
v	Variable substituted in the derivation of Expected Improvement
w	Weighting variable used in the Weighted Expected Improvement Function
w_e	Variable substituted in the derivation of Expected Improvement
$\boldsymbol{X}_{\text{exp}}$	Vector of sampled design space values
$\boldsymbol{x}$	Design variable vector
$\boldsymbol{y}_{\text{exp}}$	Vector of sampled Cost function values
y_Y	Specific realization of the random variable Y
y	Ordinary Kriging predicted function value
y_{min}	Minimum sampled cost function value
Y	Regression function (Random Variable)
Z	Deviation from the underlying regression function

Greek Symbols

Symbol	**Description**
α	Multi-objective weighted sum parameter
$\boldsymbol{\alpha}_{\text{lin}}$	Multi-objective spacing vector used when selecting points on the Pareto front for EZ-EGO
$\boldsymbol{\alpha}_{\text{scaled}}$	Multi-objective spacing vector used when selecting points on the Pareto front for EZ-EGO
β	Regression coefficients of basis functions of Kriging
ϵ_N	Random error (noise) on sampled data
ϵ	Tuning parameter in the Kushner infill sampling criteria
θ	Curve fitting parameter of spacial correlation function of Gaussian Process. (Importance or activity of variable)
μ	Mean of Gaussian process
π	Pi
σ	Kriging local standard deviation also referred to as uncertainty
σ_z	Kriging Process Variance
Φ	Normal cumulative density function

ϕ	Normal probability density function

Subscripts

Symbol	**Description**
h	Gaussian process dimension index
i, j	Element index of a vector
k	Index used in the Generalized Expected Improvement function

Superscripts

Symbol	**Description**
$\hat{}$	Refers to estimated values
$*$	Refering to a general x^* location in the design domain
$**$	Refering to a general x^{**} location in the design domain different to the location of x^*
$\boldsymbol{X}_{\text{exp}}^{(i)}$	Bracket superscript i indicating the i^{th} sample in the experimented $\boldsymbol{X}_{\text{exp}}$ list
T	Transpose

Abbreviations

Abreviation	**Description**
CDF	Cumulative Distribution Function
DACE	Design and Analysis of Computer Experiments
DE	Differential Evolution
DOE	Design of Experiments
EGO	Efficient Global Optimization
EGO-MO	Efficient Global Optimization - Multi-Objective
EI	Expected Improvement
EZ-EGO	Elemental Simultaneous EGO
GA	Genetic Algorithm
GEI	Generalized Expected Improvement Function
LHS	Latin Hypercube Sampling
MLE	Maximum Likelihood Estimator
MSE	Mean Square Error
NSGA-II	Non-dominated Sorting Genetic Algorithm
PDF	Probability Density Function
PI	Probability of Improvement

List of Figures

Contents

Chapter 1

Introduction

1.1 Design optimization

Design optimization, the topic of the present study, is a field which pertains to improving engineering solutions. The world we live in is full of significant accomplishments. For instance, we have been able to design cars and aircraft, which can transport people further and faster than ever before. We have managed to set up world wide networks and produce wireless communication. However, the world's resources are not unlimited and with a growing population, there is a need for better utilization of these limited resources.

Engineering problems are typically underspecified, resulting in a number of possible solutions. Hence, there is a need to select the better solutions from these possible solutions. To improve designs, engineers often manually iterate through different design parameters. However, the present study pertains to mathematical design optimization. Formally, this is defined as finding the best solution (design vector), from a set of feasible solutions [49]. The best solution is the solution that minimizes a chosen objective function, and feasibility is defined through a set of constraints that must be satisfied. Optimization algorithms systematically improve the objective function for feasible design vectors until the algorithm is unable to improve on the best solution.

1.2 Optimization Strategies

A number of different optimization strategies exist that can be classified in a number of ways. Figure 1.2.1 depicts three popular classes of optimization algorithms. Classical gradient based approaches require both function value and gradient information of the objective function and the constraint functions. The other classes, stochastic optimization and surrogate methods, typically only require function values.

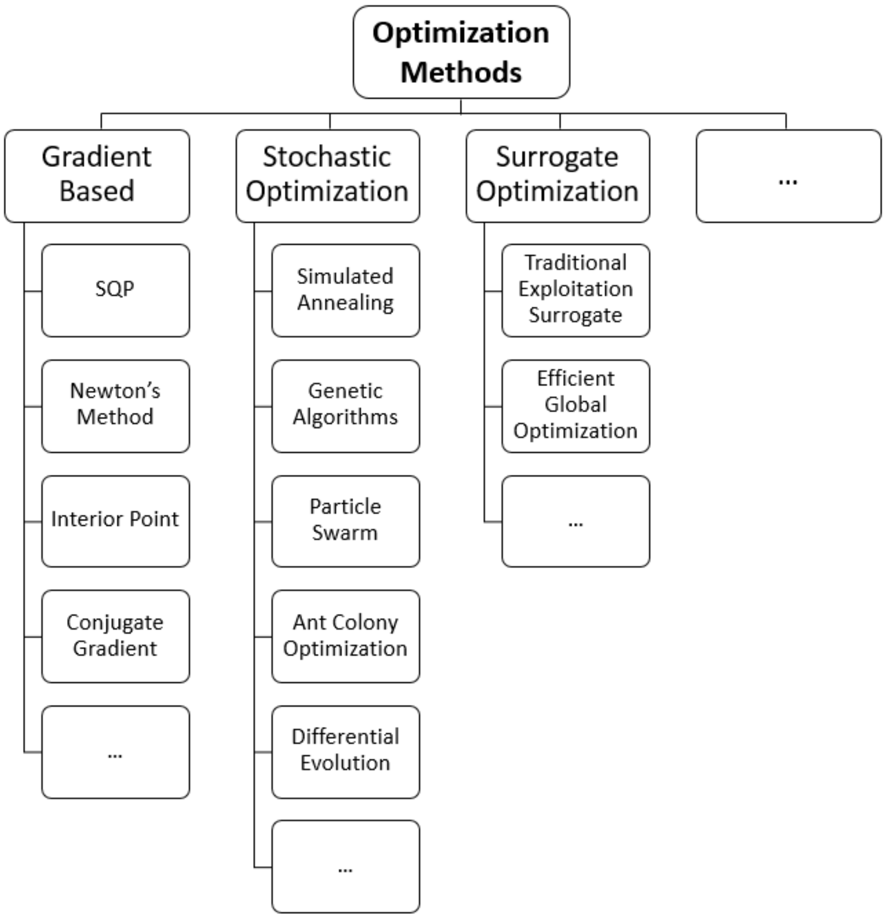


Figure 1.2.1: Different Optimization Methods

Gradient based optimizers proceed by computing the objective function value and the gradient of the objective function for a specific design vector. It then uses this information to evaluate a subsequent design along a descent direction. Gradient based optimizers are prone to converge to the minimum in the local basin of attraction in which the algorithm was initialized. Hence, to use the methods to solve multi-modal problems usually requires multiple random re-starts. Gradient based optimizers are sequential algorithms that evaluate one design at a time within an optimization run. A potential speed-up when utilizing parallel computing architectures is to approximate the required gradients using finite differences. In addition, independent optimization runs as part of a re-start strategy can be run efficiently in parallel.

Stochastic optimization algorithms are methods which perform many function evaluations in the design space to find the feasible design with the lowest function value. In contrast to gradient based approaches, these algorithms are able to find minima outside the local basin of attraction in which they were initialized. However, the number of function evaluations required by these methods is orders of magnitude higher than gradient based approaches [1]. Examples include Differential Evolution (DE) and Genetic Algorithms (GA) [1].

The strategies discussed so far assume that a reasonable number of function evaluations are practical to compute. However, in modern day design, computer aided analysis has become commonplace. It is not unusual for a single analysis to take a couple of hours to compute on a cluster of computers. To make such design problems tracktable to solve, surrogate models or response surfaces are constructed that approximate the objective and constraint functions of the actual problem. These surrogate models are computationally efficient to evaluate. Another benefit of surrogate methods is that designs can be evaluated independently and in parallel. Surrogate models only require a limited number of function evaluations to construct an approximation of the actual functions. The surrogate is then optimized instead of the expensive objective function. The accuracy of the surrogate may then be improved by performing additional real function evaluations. Optimization is performed again, and this cycle can repeat until convergence. The curse of dimensionality does not effect all algorithms equally and surrogate models are limited in the number of design variables they can handle. Despite this limitation, surrogate optimization remains popular and is studied in this research.

1.3 Standard design optimization formulation

The standard mathematical formulation for an optimization problem is [1]:

Minimize the objective function

$$f(\boldsymbol{x}) = f(x_0, x_1, x_2, \ldots, x_{n-1}), \tag{1.3.1}$$

subject to p equality constraints

$$h_i(\boldsymbol{x}) = h_i(x_0, x_1, x_2, \ldots, x_{n-1}) = 0; \quad i = 1, 2, \ldots, p \tag{1.3.2}$$

and subject to m to inequality constraints

$$g_i(\boldsymbol{x}) = g_i(x_0, x_1, x_2, \ldots, x_{n-1}) \leq 0; \quad i = 1, 2, \ldots, m, \tag{1.3.3}$$

where $\boldsymbol{x}$ represents the n-dimensional vector of design variables, which we refer to as the design vector in this study.

1.4 Multi-objective optimization

Many cases exist where the design problem can rather be formulated by simultaneous optimization of conflicting criteria. An unconstrained multi-objective optimization problem is defined by the following vector-valued objective function

$$\text{Minimize} \quad \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), f_3(\boldsymbol{x}), ..., f_{n_f}(\boldsymbol{x})), \tag{1.4.1}$$

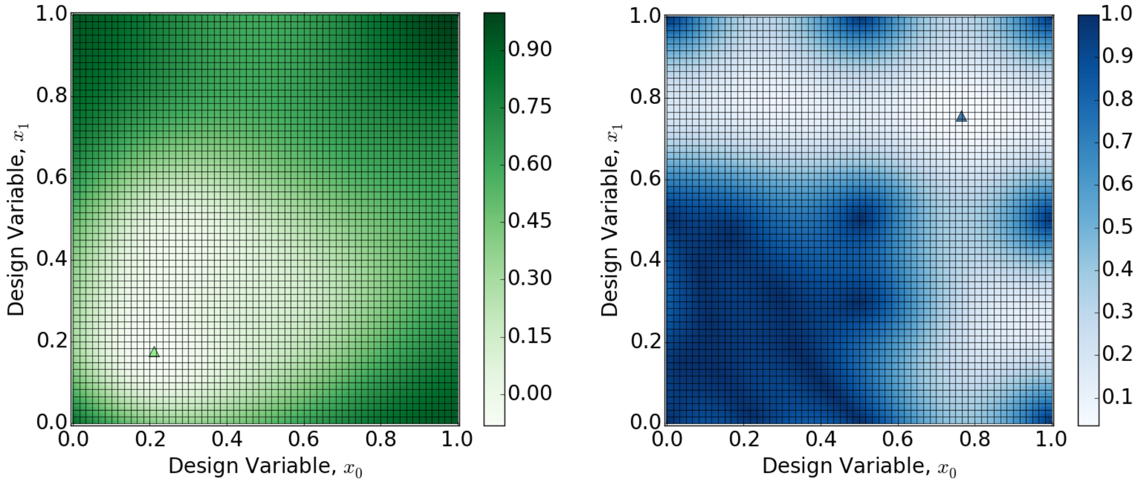
where $\boldsymbol{f} = (f_1, f_2, ..., f_{n_f})$ is the n_f scalar valued objective functions or criteria. In this study, the multi-objective optimization problem is encountered when deciding where to sample the design domain. One obvious choice is to sample in the vicinity of the lowest function value to date (behaviour described as exploitation in optimization literature). The other obvious choice is to sample in locations where no functions evaluations have been performed (behaviour described as exploration in the optimization literature). The multi-objective optimization problem for these two objective functions is defined by

$$\text{Minimize} \quad f_1(\boldsymbol{x}) - \text{Optimize exploitation} \tag{1.4.2}$$

$$f_2(\boldsymbol{x}) - \text{Optimize exploration.} \tag{1.4.3}$$

1.4.1 Set of Optimal Solutions

Multi-objective optimization problems generally have a set of optimal solutions. To illustrate, consider the two objective functions depicted in Figure 1.4.1. The minima of these functions are indicated with the triangles. Since the optimum of one function is not the optimum of the other, we need a more sophisticated concept, Pareto optimality, to define the solution.



(a) Objective Function f_1 **(b)** Objective Function f_2

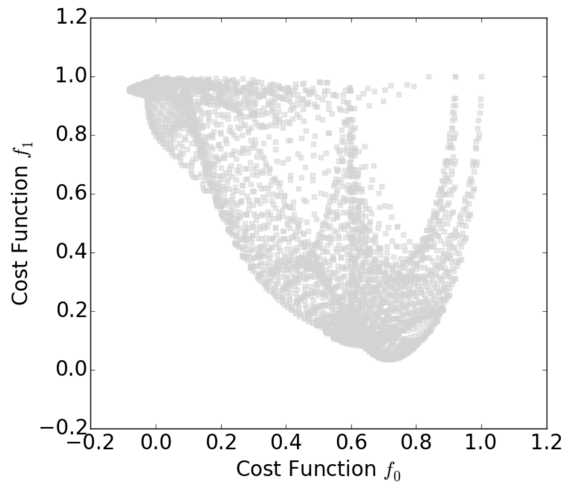
Figure 1.4.1: The figure presents the contour plots of the two objective functions of a multi-objective optimization problem. The triangles indicate the minimum objective function value. Notice how the optimum of these two bjective function do not lie in the same place. The gridlines indicated are used to generate the criterion space plot presented in Figure 1.4.2.

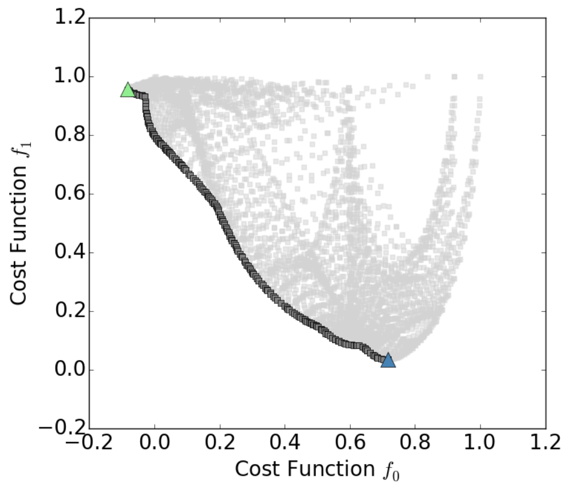
The Pareto optimal set is a set of design solutions, which occur if there is no other design in the design space which reduces at least one of the objective functions, without increasing another [1]. Formally this is defined as:

> A design $\boldsymbol{x}^*$ in the feasible design space S is Pareto optimal if and only if there does not exist another design $\boldsymbol{x}$ in the set S such that $\boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{f}(\boldsymbol{x}^*)$ [1] with at least one $f_i(\boldsymbol{x}) < f_i(\boldsymbol{x}^*)$.

Reconsider Figure 1.4.1, which also contains fine gridlines. Both functions are evaluated at each of these gridline intersections. The functions values are then plotted against each other in Figure 1.4.2 (a), for each of the gridline intersections. The definition of Pareto optimality is then applied, and the Pareto optimal designs are plotted using square symbols in Figure 1.4.2 (b). The extreme locations of the Pareto optimal curve coincide with the global minima of each of the objective functions, again indicated using triangles. Since the two axes of this plot are the two objective functions (the optimization criteria), this is known as the Pareto optimal set plotted in criterion space.

[1]Where vector inequalities are applied between each corresponding vector element, for instance $\boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{f}(\boldsymbol{x}^*)$ implies $f_1(\boldsymbol{x}) \leq f_1(\boldsymbol{x}^*)$ and $f_2(\boldsymbol{x}) \leq f_2(\boldsymbol{x}^*)$

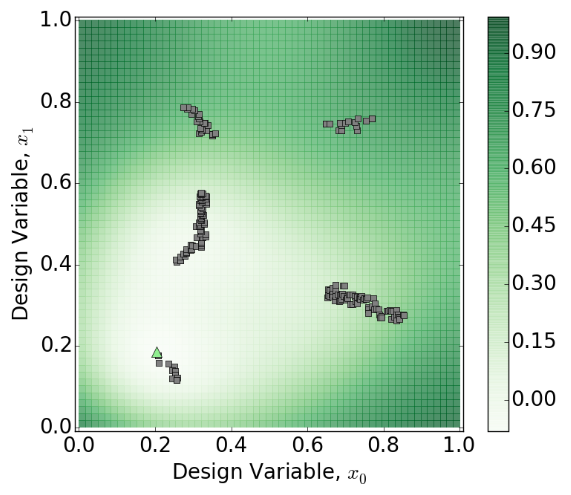


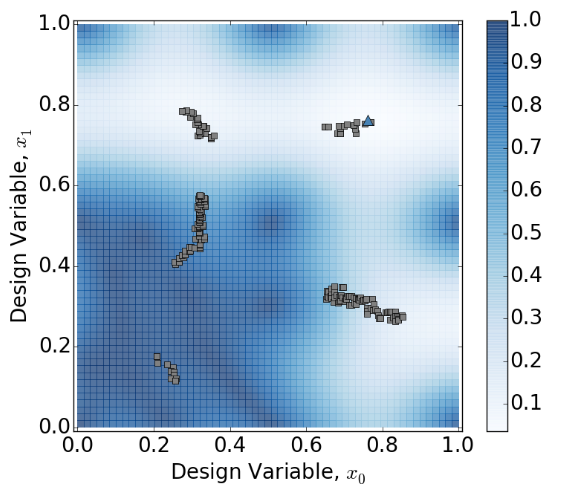


(a) Pareto Front not Indicated **(b)** Pareto Front Indicated

Figure 1.4.2: Using the gridlines plotted in Figure 1.4.1 the value the two objective functions at the gridline intersections are plotted against each other. (a) indicates only the values generated using the gridlines, while (b) also indicates the Pareto front found with NSGA-II. The triangles indicate the minimum function value for each of the two objective functions.

The Pareto optimal set can also be plotted in the design space. Figure 1.4.3 plots all the Pareto optimal designs from Figure 1.4.2 (b) in the design space, superimposed with the objective function contours of f_1 and f_2.





(a) Objective Function f_1 **(b)** Objective Function f_2

Figure 1.4.3: The Pareto optimal set from Figure 1.4.2 can be mapped back to the design space the Pareto optimal designs are indicated.

1.4.2 Solving Pareto Optimal Sets

An intuitive method to solve a multi-objective optimization problem is to rewrite the multi-objective problem as a single objective problem. This is called scalarization. One such method is called the weighted sum method. Consider the multi-objective optimization problem formulation stated in Equations (1.4.2) and (1.4.3). The weighted sum formulations is written as

$$\text{Minimize} \quad f_{\text{Weighted Sum}} = \alpha f_1(x) + (1-\alpha) f_2(x), \tag{1.4.4}$$

where α now is a trade-off between the two objective functions. If $\alpha = 1$ only the objective function f_1 is optimized, while if $\alpha = 0$ only objective function f_2 is optimized. If the user selects values for α, different solutions along the Pareto front are found.

A limitation of the weighted sum method is that it cannot find Pareto optimal designs that lie on the non-convex parts of the Pareto front. Figure 1.4.4 illustrates a large non-convex part on the Pareto front. Mathematical details about this limitation can be found in Caramia and Dell'Olmo [7].

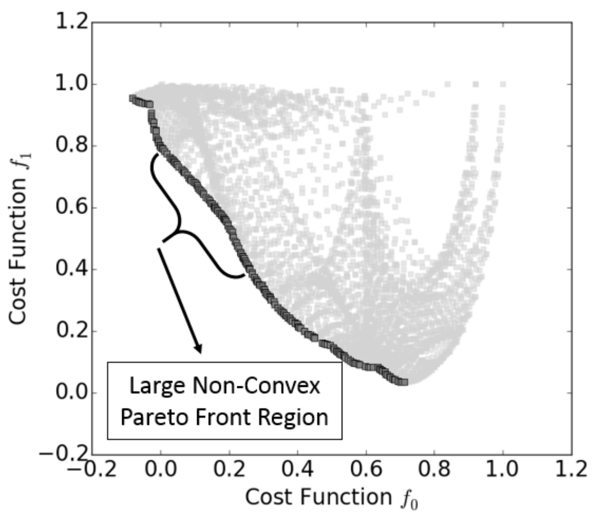


Figure 1.4.4: A portion of Pareto front which is non-convex and cannot be solved by the weighted sum method is indicated.

Given the limitations of the weighted sum method, other methods have been developed. Arora [1] lists a number of methods for multi-objective optimization, such as a Multi-objective Genetic Algorithm, a weighted max-min method, and a weighted global criterion method.

Genetic algorithms are iterative optimization algorithms that fall into the catagory of evolutionary optimization discussed in Section 1.2. The advantages of evolutionary algorithms are that they are able to solve continuous, non-continuous, and discrete optimization problems. Furthermore, evolutionary algorithms not only exploit the information they have available, they also explore the domain. This generally prevents them from getting stuck in the closest local basin of attraction like a gradient based optimization method would have done. The disadvantages of evolutionary algorithms include that they require many function evaluations to solve a problem, and they generally do not reach the same accuracy as gradient based optimization methods.

In this research, at least one of the objective functions for the multi-objective problem we routinely want to solve is highly multi-modal. Both objective functions are cheap to evaluate. Therefore the genetic multi-objective algorithm NSGA-II [13] is suitable to solve the multi-objective problems in this thesis. We use an implementation available in the computational package PyGMO [5]. NSGA-II aims to ensure that designs are evenly spread out over the Pareto front.

1.5 Surrogate Optimization: Exploitation and Exploration

As discussed in Section 1.2, surrogate based optimization strategies aim to use few function evaluations in order to find the global optimum of a objective function [3, 10, 14, 22, 23, 28, 33, 46, 58]. Surrogate optimization methods start with an initial sample of designs that are first evaluated and then used to construct a response surface. The accuracy of the response surface is improved by evaluating additional designs and using these responses to update the response surface. Subsequent designs are selected based on two criteria; exploitation and exploration. The surrogate optimization method can select the subsequent design vector where the response surface predicts low function values. This is called an exploitation approach. Or alternatively, it can select the subsequent design where the response surface is uncertain about the value of the actual function value. This is called an exploration approach.

This exploitation versus exploration approaches can be explained by using an analogy of a family vacation. A family has previously had a few family vacations at a range of different destinations. They had good experiences at certain vacation locations and bad experiences at other locations. There are also a number of locations where they have never been. The family has to plan their next vacation. They can go back to a location with known good experiences, hence they exploit previous knowledge and the risk of having a bad experience is low. Alternatively, they can decide to visit a new destination, to explore new possibilities. In this case, they have a higher risk of having a bad experience, but there is also the potential of a much better experience than previous holidays.

Traditional surrogate optimization methods are exploitation based. Usually, an initial sample of designs is selected, and a response surface is then fit to the actual responses computed at these designs. An optimization algorithm is then used to find the minimum of the response surface. The actual function is then evaluated at the design that corresponds to the minimum of the response surface. This conventional method does not explore beyond the designs evaluated initially. If the global minimum basin is not described adequately by the current response surface, it is unlikely that this method will find the global minimum.

Efficient Global Optimization (EGO) is a specific surrogate optimization method that was first proposed by Jones et al. [25]. It falls into the class of Bayesian Optimization first coined by Mockus et al. [34]. EGO uses a Kriging model to interpolate between the responses of the evaluated designs. Kriging is a unique regression method, which not only predicts the function value at unsampled design vector, but also gives an estimation of the uncertainty at any design vector in the domain. It was named after Danie Krige, a South African mining engineer and is also called Gaussian process regression [35]. EGO takes both exploration (low predicted function values) and exploitation (high uncertainty) into account when deciding where to sample next. The function used to quantify where to sample next is known as an infill sampling criterion (also known as an acquisition function). The EGO infill sampling criterion is known as Expected Improvement (EI), and this function quantifies the likelihood of finding a design with a lower functions value than the current best design. The infill sampling criterion, EI, was extended by Schonlau et al. [45] to Generalized Expected Improvement (GEI), which allows the user to tune to emphasis on exploitation and exploration. Since this study focuses on EGO, a detailed background on Kriging and EGO is given in Chapter 2.

1.6 Parallel variants of EGO

Since the development of EGO, the objective of computers has decreased, and their capabilities have increased. Many laptops have 8 processors and are able to perform multiple computational analyses simultaneously. Computing clusters have many more cores and are able to perform even more simultaneous analyses. The traditional EGO approach only selects one design per iteration to sample next. Strategies where EGO is adapted to select multiple designs per iteration are called parallelization strategies.

The first variant of parallel EGO algorithms selects a new design by maximizing the infill sampling criterion. The response surface is then updated by assuming the function value returned by the actual function will equal the response surface function value. The updated infill sampling criterion is maximized yet again, and this process repeats until the required number of new designs for this iteration is found. These designs are then evaluated in parallel. Examples of such EGO parallelization strategies are the Kriging Believer strategy proposed by Schonlau et al. [44, 45] and variations of this method by Ginsbourger et al. [18].

Sóbester et al. [47] proposed another parallel sampling strategy where the multiple local maxima of the EI function are found and selected to sample in the current iteration. Ponweiser et al. [38] developed this method further by adding clustering to determine how design vectors are selected.

Viana et al. published a series of papers which proposes using multiple different surrogates when running EGO [53, 54, 55, 56]. Each surrogate would use a different model with a different minimum. They used Gaussian Process Regression, a Radial Basis Neural Network, a Linear Shepard model, and a variety of Support Vector Regression models. Only Gaussian Process Regression directly provided a measure of uncertainty and therefore the uncertainty measure of the Gaussian Process Regression was used throughout. The number of samples which can be added in each iteration is limited by the number of surrogates selected.

Of particular interest in this study, is the multi-objective parallelization of EGO. The EI function has two terms, and many papers have labeled these two terms as exploitation and exploration terms respectively [24, 38, 43, 47]. Sóbester [48] therefore argued that if a linear weighting of these two terms is introduced (Weighted Expected Improvement), it should give the user control over biasing the search towards exploration or towards exploration. As a future endeavor, he suggested using this weighted EI function to perform multi-objective optimization. This idea was furthered in the works of Feng et al. [15, 16]. They treated the two terms of the EI function as a multi-objective optimization problem. They then selected some of the Pareto optimal designs to sample in the subsequent iteration. This method is called EGO-MO.

1.7 Research Aim

The research aim in this thesis is to develop a new parallel EGO algorithm that is also inspired by the multi-objective nature of exploitation and exploration. Primarily we would like to investigate the behaviour of this algorithm, and also compare our new algorithm to a recently published method. The new algorithm simplifies parallel EGO to its bare essentials by using predicted function value as exploitation indicator, and the Kriging standard deviation as exploration indicator. The new algorithm is called Simple Intuitive Multi-objective ParalLElization of Efficient Global Optimization (SIMPLE-EGO).

1.8 Thesis overview

Chapter 2 gives background on Kriging and the classical EGO algorithm. A reader who is familiar with EGO may choose to skip ahead to Chapter 3 where we investigate and discuss the behaviour of EGO. In Chapter 4 we investigate multi-objective optimization of EGO and compare our SIMPLE-EGO method with a multi-objective parallelization strategy proposed by Feng et al. [16]. We also present a short discussion of EGO-MO in Chapter 4. Finally in Chapter 5 we conclude this report.

Chapter 2

Background on Efficient Global Optimiztion (EGO)

2.1 EGO Section Outline

The steps associated with the EGO algorithm are outlined in Figure 2.1.1. This chapter details the steps that respectively address the initial design of experiments, the fitting of the Kriging surrogate, different infill sampling criteria and finally the stopping criteria of the algorithm. A reader familiar with these topics may continue reading in Chapter 3.

2.2 Initial Design of Experiments (DOE)

The first step of EGO is to select an initial set of designs to sample. Ideally the design vectors should be distributed evenly over the design space and space filling designs are used. A variety of design of experiment (DOE) methods are available and include factorial, fractional factorial, central composite, latin hypercube sampling (LHS) and LP$_\tau$ [48]. Jones et al. [25] recommended using LHS [32], while a more recent source [2] suggested using Centroidal Voronoi Tessellation.

Jones et al. [25] proposed an initial sample size of $11n-1$ designs, where n is the number of dimensions of the design space. In 2005, Sóbester et al. [48] investigated the impact of initial sample size. They found that the ideal sample size varies depending on the nature of the objective function.

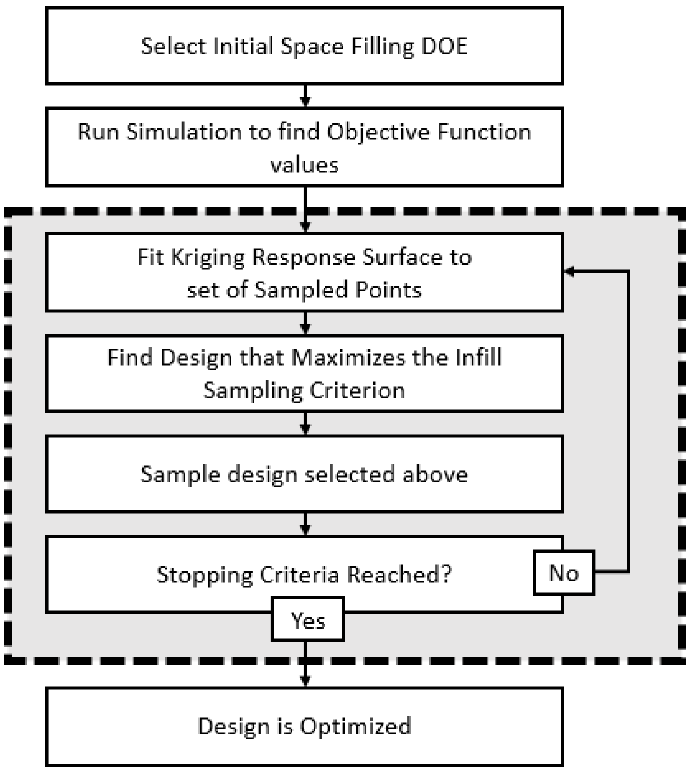

Select Initial Space Filling DOE
Run Simulation to find Objective Function values
Fit Kriging Response Surface to set of Sampled Points
Find Design that Maximizes the Infill Sampling Criterion
Sample design selected above
Stopping Criteria Reached?
No
Yes
Design is Optimized

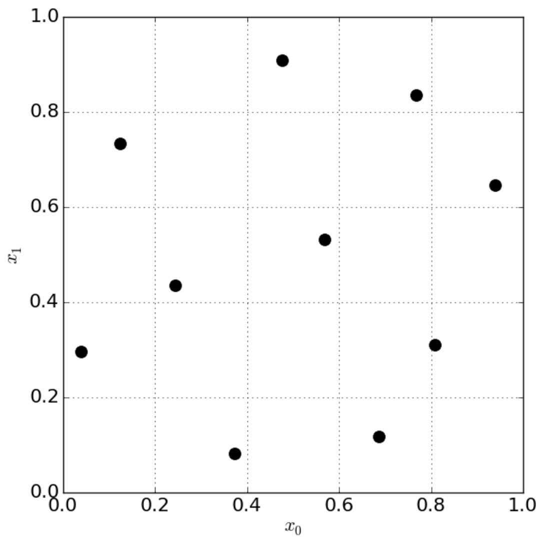
Figure 2.1.1: EGO Algorithm

To generate a LHS design we used the `optimumLHS` function in the `lhs` package [8] implemented in the R-Project [39]. Figure 2.2.1 shows a LHS DOE for a 2D problem. The selected design vectors are plotted as black dots. Figure 2.2.1 (a) shows a DOE for 10 design vectors while Figure 2.2.1 (b) shows a DOE for 50 design vectors. As shown, the designs vectors are approximately equispaced, and cover the design space uniformly. Additional information on the settings we used is available in Appendix C.

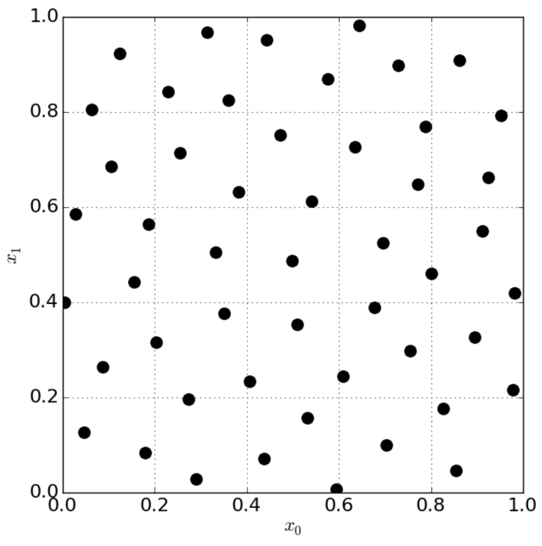
2.3 Gaussian Process Regression (Kriging)

In EGO, Kriging is used to approximate the objective function from the DOE evaluated designs. Kriging [11, 12, 20, 30, 40, 50] is a powerful regression method, which was first used in geology and named after Danie Krige, a South African Mining Engineer. Later the method was also published by Matheron [31]. This field then became known as geostatistics.

At the end of 1980 Sacks et al. [41] took Kriging into a new direction, when they applied



(a) 10 Design Vectors



(b) 50 Design Vectors

Figure 2.2.1: Latin Hypercube Sampling (LHS) Design of Experiments (DOE)

it to Computer Experiments and called this Design and Analysis of Computer Experiments (DACE) [42]. DACE applications differ fundamentally from Geostatistics since computer experiments are deterministic. This means if the same design is evaluated twice by a computer, the same result is expected, where with geostatistical experiments, measurement noise exists. In addition the nature of the gold deposit landscape is vastly different to a typical objective function for a design optimization problem. This section first describes the general regression method, then it outlines how Kriging works following the presentation of Sasena [43].

2.3.1 General Regression

In general, regression methods take on the form $Y(\boldsymbol{x}) = f_Y(\boldsymbol{x}) + \epsilon_N(\boldsymbol{x})$ where $f_Y(\boldsymbol{x})$ is some trend function, and $\epsilon_N(\boldsymbol{x})$ is the random error associated with the output. The capital letter $Y(\boldsymbol{x})$ refers to a stochastic function where $y_Y(\boldsymbol{x})$ refers to a specific realization of $Y(\boldsymbol{x})$. The random error is usually assumed to be normally distributed $N(0, \sigma^2)$ for all $\boldsymbol{x}$. Figure 2.3.1 shows a noisy dataset with a regression function fit through the data.

Kriging makes the same root assumption, i.e. function can be decomposed into the

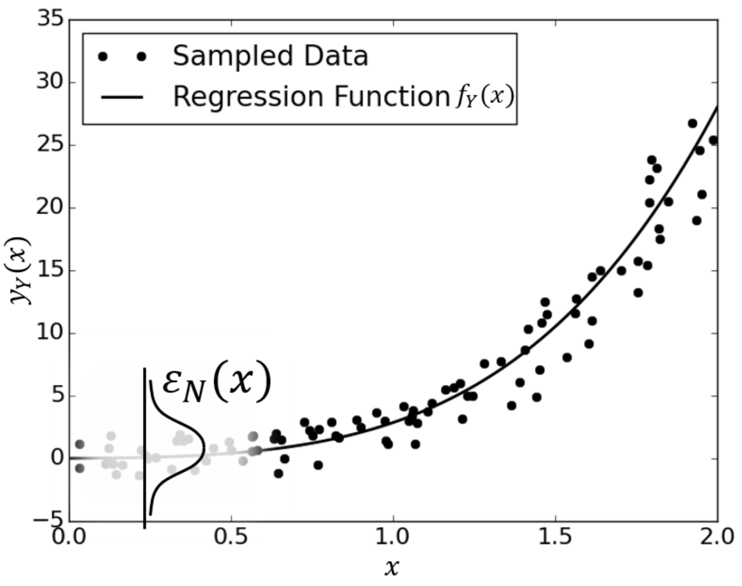


Figure 2.3.1: Example of Regression of data with Random Errors

sum of some trend function and a deviation from that trend. A dataset generated by deterministic computer experiments is not stochastic but deterministic. Therefore a deterministic function is now decomposed as $Y(\boldsymbol{x}) = f_Y(\boldsymbol{x}) + Z(\boldsymbol{x})$, where $f_Y(\boldsymbol{x})$ is the mean response with $Z(\boldsymbol{x})$ the deviation from the mean response.

Kriging makes the assumption that deviations from the mean, $Z(\boldsymbol{x})$, are correlated. If two $\boldsymbol{x}$ values are close together, then their deviation from the regression function, $f_Y(\boldsymbol{x})$, should also be similar, as illustrated in Figure 2.3.2.

Hence, in Kriging the predicted function value, $y_Y(\boldsymbol{x}^*)$, for a specific design is influenced significantly by analyzed designs that are close to it, and lesser influenced by sampled designs further away. The next section describes how Kriging estimates $f_Y(\boldsymbol{x})$ and $Z(\boldsymbol{x})$.

2.3.2 Kriging estimate of the mean response

In Kriging the mean function $f_Y(\boldsymbol{x})$ is a low order polynomial. The response $Y(\boldsymbol{x}$ is given by

$$Y(\boldsymbol{x}) = \sum_{j=1}^{k_Y} \beta_j f_{Y_j}(\boldsymbol{x}) + Z(\boldsymbol{x}), \tag{2.3.1}$$

where f_{Y_j} are the k_Y polynomial terms (or basis functions) of the mean function $f_Y(\boldsymbol{x})$ and β_j are the corresponding regression coefficients. $Z(\boldsymbol{x})$ is assumed to be a Gaussian

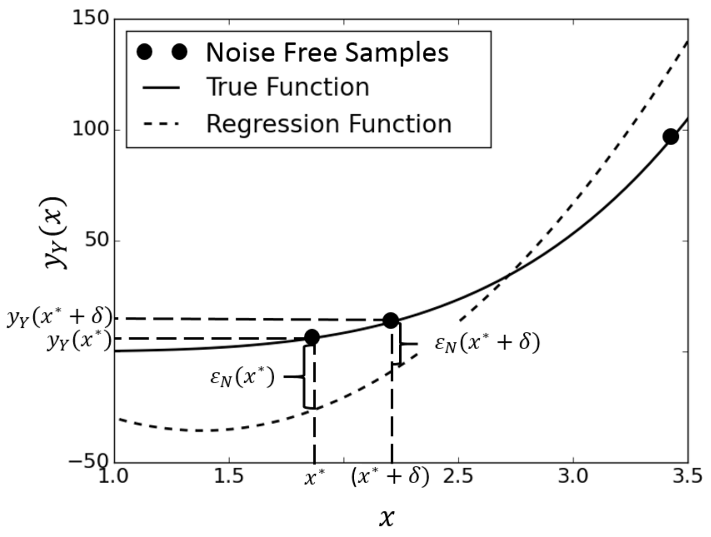


Figure 2.3.2: Kriging is based on the assumption two data points which are close together will have a correlated deviation from a trendline. The figure illustrates how the data points that close together have similar deviation from the trendline ϵ_N.

process with zero mean i.e. $E(Z(\boldsymbol{x})) = 0$.

EGO assumes the regression function $f_Y(\boldsymbol{x})$ is a constant, which is also known as ordinary Kriging. For the remainder of this report we refer to this constant regression function as the process mean, $\hat{\mu}$. The process mean, $\hat{\mu}$, and variance, VAR, are given by

$$\hat{\mu} = \frac{\mathbf{1}^T \boldsymbol{R}^{-1} \boldsymbol{y}_{\text{exp}}}{\mathbf{1}^T \boldsymbol{R}^{-1} \mathbf{1}} \tag{2.3.2}$$

$$\text{VAR} = \hat{\sigma}_z^2 = \frac{(\boldsymbol{y}_{\text{exp}} - \mathbf{1}\hat{\mu})^T \boldsymbol{R}^{-1} (\boldsymbol{y}_{\text{exp}} - \mathbf{1}\hat{\mu})}{n_s}. \tag{2.3.3}$$

Here the ˆ distinguishes the true process mean μ and variance σ from the estimated process mean $\hat{\mu}$ and estimated process variance $\hat{\sigma}$. The number of sampled design vectors is given by n_s, $\mathbf{1}$ is a vector of n_s ones, $\boldsymbol{y}_{\text{exp}}$ is the vector of sampled function values and $\boldsymbol{R}$ is the spatial correlation matrix of size $n_s \times n_s$. The (i, j) entries of $\boldsymbol{R}$ are given by

$$R[(\boldsymbol{X}_{\text{exp}})^{(i)}, (\boldsymbol{X}_{\text{exp}})^{(j)}] = \sum_{h=0}^{n-1} \exp\left(-\theta_h |(\boldsymbol{X}_{\text{exp}})_h^{(i)} - (\boldsymbol{X}_{\text{exp}})_h^{(j)}|^{p_h}\right), \tag{2.3.4}$$

where n is the number of dimensions of $\boldsymbol{x}$ and $\boldsymbol{X}_{\text{exp}}$ contains the sampled design vectors. The parameters $\theta_h > 0$ and $0 < p_h < 2$. $\boldsymbol{r}(\boldsymbol{x})$ is the spatial correlation vector where

$r_i(\boldsymbol{x}) = R[\boldsymbol{x}, (\boldsymbol{X}_{\text{exp}})^{(i)}]$. The predicted function value, $\hat{y}(\boldsymbol{x})$ is computed using

$$\hat{y}(x) = \hat{\mu} + \boldsymbol{r}^T \boldsymbol{R}^{-1}(\boldsymbol{y} - \mathbf{1}\hat{\mu}), \tag{2.3.5}$$

while the mean square error (MSE) is given by

$$\text{MSE}[\hat{y}(x)] = \hat{\sigma}^2(x) = \hat{\sigma}_z^2(1 - \boldsymbol{r}^T \boldsymbol{R}^{-1}\boldsymbol{r}). \tag{2.3.6}$$

Figure 2.3.3 shows the effect of different θ on the correlation function. The larger θ the larger the radius of designs which effect each other. The parameter, p_h, is a measure of how smooth the function is. For DACE and EGO, an Gaussian correlation function is used, meaning p_h is chosen as two for all h. In general, a different choice of θ_h and p_h are possible for each dimension of $\boldsymbol{x}$.

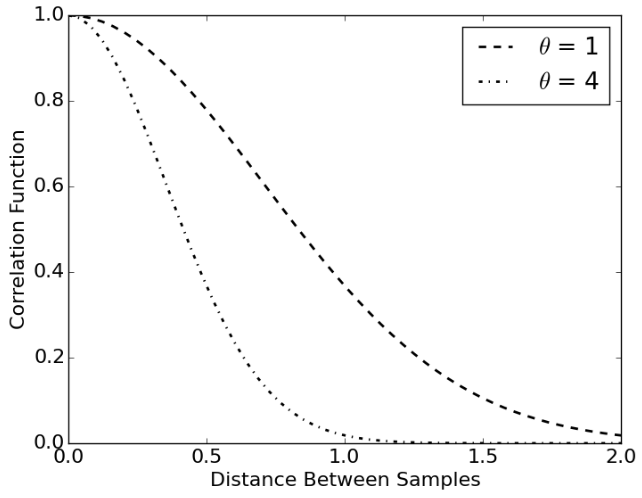


Figure 2.3.3: Effect of θ on the Gaussian Correlation Function. The larger θ is the larger the radius of influence of a specific data point.

For the remainder of the report, we will not refer to $\hat{y}(\boldsymbol{x})$, $\hat{\sigma}(\boldsymbol{x})$ and $\hat{\mu}(\boldsymbol{x})$, but to $y(\boldsymbol{x})$, $\sigma(\boldsymbol{x})$ and $\mu(\boldsymbol{x})$ as we assume our estimation of these parameters are correct.

2.3.3 Parameter Estimation

The range parameters $\boldsymbol{\theta}$ are usually estimated by the Maximum Likelihood Estimator (MLE), that is given by

$$\text{MLE}(\boldsymbol{\theta}) = \frac{1}{(2\pi)^{n/2}((\sigma_z(\boldsymbol{\theta}))^2)^{n/2}|\boldsymbol{R}(\boldsymbol{\theta})|^{1/2}} \exp\left(-\frac{(\boldsymbol{y}_{\text{exp}} - \mathbf{1}\mu(\boldsymbol{\theta}))^T \boldsymbol{R}(\boldsymbol{\theta})^{-1}(\boldsymbol{y}_{\text{exp}} - \mathbf{1}\mu(\boldsymbol{\theta}))}{2(\sigma_z(\boldsymbol{\theta}))^2}\right). \tag{2.3.7}$$

Sasena [43] noted that in some cases the MLE may not converge and they opted to minimize the cross validation error to estimate $\boldsymbol{\theta}$. Although the MLE statistical assumption of zero mean variance is not always true for optimization functions, we would like our computational implementation to be the comparable to Jones et al. [25] and Feng et al. [16]. For this reason we use MLE in this research.

2.3.4 Underlying Regression Function

In Figure 2.3.4 two different underlying regression functions, $f_Y(\boldsymbol{x})$, are used to approximate the data points. Both a constant and a quadratic underlying regression function are shown. The figure shows that for interpolation the underlying regression function does not have a significant impact. However, in the extrapolated regions the two methods differ significantly, and thus EGO should not be used to extrapolate data.

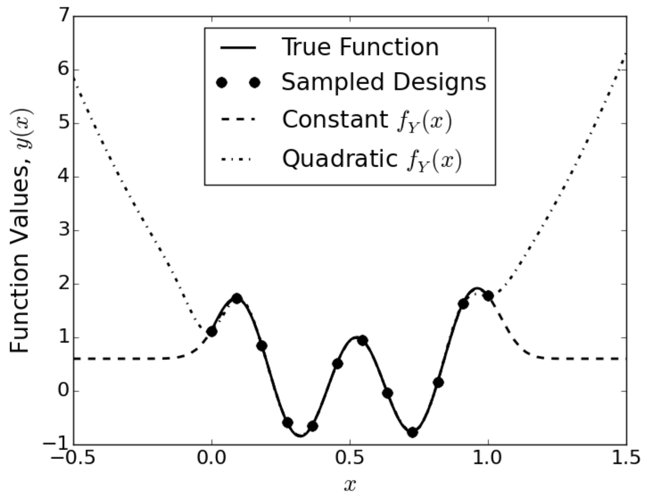
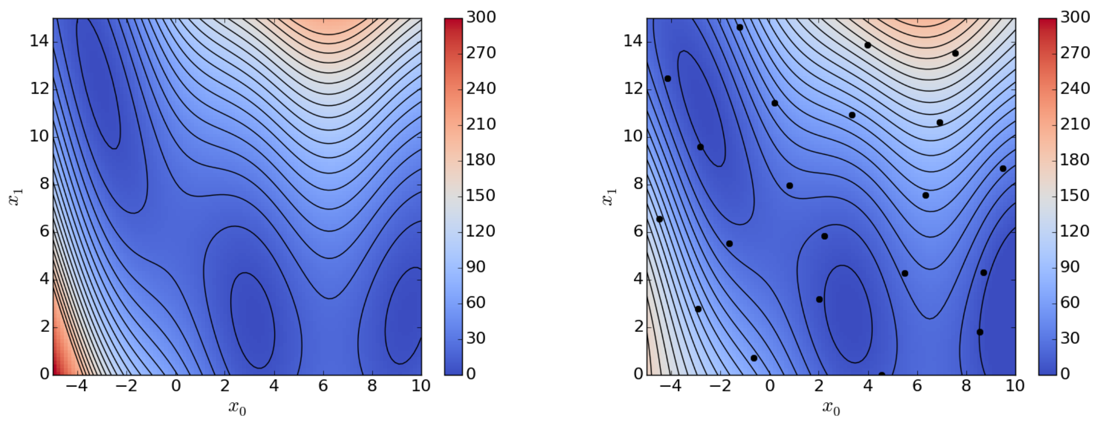


Figure 2.3.4: Effect of the Underlying Correlation Function $f_Y(\boldsymbol{x})$, showing how Kriging performs well for interpolation, but should not be used for extrapolation.

2.3.5 Kriging Performance

At the time when EGO was published, Kriging was compared to other regression methods, such as a quadratic surface, and a thin plate spline. Kriging outperformed these two methods by far. However, with recent advances in the field of Machine Learning, many robust regression methods are available. Examples are Radial Basis Functions and Support Vector Regression. Figure 2.3.5 shows how well Kriging can approximate the Branin Function while using a DOE consisting of 21 design vectors.



(a) **(b)**

Figure 2.3.5: (a) Contours of the true Branin function, and (b) contours of the Kriging Approximation of the Branin Function using a 21 sample DOE

2.3.6 Numerical Instabilities

One of the challenges faced in implementing EGO is the ill-conditioning of the correlation matrix $\boldsymbol{R}$. There are two reasons for the ill-conditioning. The first happens as a result of approximating a smooth function. In such a case the design vectors will be highly correlated, which implies that each column in the correlation matrix will be a column of ones. Thus the matrix $\boldsymbol{R}$ will be excessively collinear. The second takes place when two sampled design vectors are close together. Then two columns in the correlation matrix, $\boldsymbol{R}$, are almost the same making $\boldsymbol{R}$ highly ill-conditioned.

Jones et al [25] suggested using singular value decomposition as discussed in the book *Numerical Recipes*. Sasena [43] use a nugget to get rid of this ill-conditioning. A nugget is also what is used to fit Kriging to noisy data. As discussed in Sasena [43], we use a nugget of 10^{-12} to treat numerical instabilities in this thesis.

2.3.7 Kriging Uncertainty Estimates: Standard Deviation

Kriging allows the standard deviation of an interpolated function value to be estimated. It serves as a measure of uncertainty and is leveraged by EGO. Figure 2.3.6 shows an example of the Kriging standard deviation estimates. The Kriging function closely fits the true function. Note that where the design vectors are further apart, the standard deviation estimate is higher. Some authors have noted that Kriging underpredicts the standard deviation [18] [25].

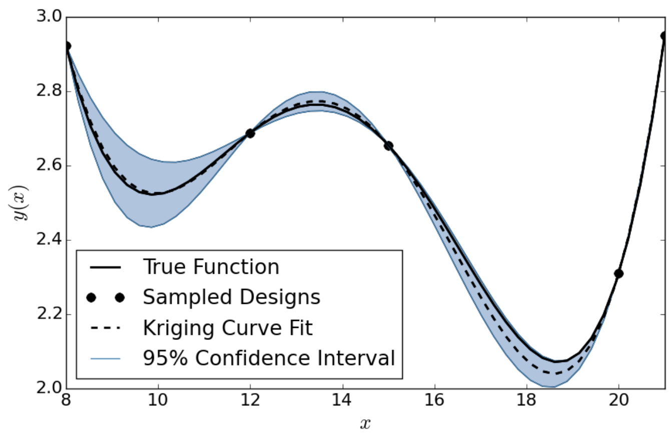


Figure 2.3.6: Kriging Uncertainty Estimates in the form of Standard Deviation. The graph shows the 95% confidence interval based on the Kriging standard deviation.

2.4 Infill Sampling Criteria

Recall from Section 1.5 that the two criteria of exploitation and exploration are used by EGO to select the next design. The mechanism that EGO utilizes to achieve this, is to maximize the infill sampling criteria, which is also often referred to as an acquisition function [29]. Infill sampling criteria are scalarized functions indicating where to select the next design sample. In Sections 2.4.1 to 2.4.4 we discuss some of the prominent infill sampling criteria, and in Section 2.4.5 we describe how we find the maximum and provide a visual example of how EGO progresses.

2.4.1 Probability of Improvement (PI)

The probability of improvement infill sampling criterion was first published by Harold Kushner in 1964 [28]. At the stage it was named the Kushner Criterion. The mathematical expression for the Kushner Criterion is given by

$$\begin{aligned}\text{Kushner}(\boldsymbol{x}) &= \text{P}(y < y_{\text{min}} - \epsilon) && (2.4.1)\\ &= \Phi\left(\frac{(y_{\text{min}} - \epsilon) - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right), && (2.4.2)\end{aligned}$$

where P is the probability, y_{min} refers to the current lowest sampled value, Φ is the cumulative distribution function (CDF) of a normal distribution, and ϵ is a tolerance. ϵ determines how much lower than y_{min} the predicted objective function value, $y(\boldsymbol{x})$, must be before the difference is considered significant. In each iteration the aim is to find $\boldsymbol{x}^*$, that maximizes Kushner$(\boldsymbol{x})$.

A special case of the Kushner criterion is when $\epsilon = 0$. Watson and Barnes [57] refer to this criterion as *Locating the threshold-bounded extremes*, while other authors refer to it as Probability of Improvement [44, 52]. For the remainder of this report, this sampling criterion will be named Probability of Improvement (PI) defined as

$$\begin{aligned}\text{PI}(\boldsymbol{x}) &= \text{P}(y(\boldsymbol{x}) < y_{\text{min}}) && (2.4.3)\\ &= \Phi\left(\frac{y_{\text{min}} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right). && (2.4.4)\end{aligned}$$

This criterion can be classified as an exploitation infill sampling criterion as it puts more emphasis on sampling where the response surface predicts low function values [44].

2.4.2 Expected Improvement (EI)

Expected Improvement (EI) is the acquisition criterion on which EGO is built. The maximum of EI defines the next design to be evaluated. The EI criterion considers both exploitation and exploration. EI is defined by

$$\text{E}[I(x)] = \text{E}[\max(y_{\text{min}} - y(\boldsymbol{x}), 0)], \tag{2.4.5}$$

where E is the expected value. Using Equation (2.4.5), it can be shown (Appendix A, [4, 44]) to be equivalent to

$$\text{EI}(x) = (y_{\text{min}} - y(\boldsymbol{x}))\Phi\left(\frac{y_{\text{min}} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right) + \sigma(\boldsymbol{x})\phi\left(\frac{y_{\text{min}} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right), \tag{2.4.6}$$

where ϕ and Φ are the probability density function (PDF) and cumulative distribution function (CDF) of the standard normal distribution respectively. EI expresses the likelihood that a function value at an unknown sample location will be less than the best function value sampled to date. EI elegantly combines two locations where this is likely to occur: in the vicinity of the current best sampled location, or where there has been no sampling (hence large standard deviation) but reasonably low function values are predicted there.

Although the original derivation of EI does not depart from a perspective of exploitation and exploration, it is seemingly attractive to interpret each of the EI terms as such. Many authors have labeled the two terms as an exploitation term and an exploration term [24, 38, 43, 47].

The first term in Equation (2.4.6) contains the quantity $(y_{\min} - y(\boldsymbol{x}))$, which measures how much the predicted function value is smaller than the best function value sampled to date. Hence the first EI term is positive if the predicted objective function value is smaller than the best function value sampled to date. Hence it seems reasonable to interpret term one of EI as an exploitation measure.

The second term in Equation (2.4.6) is proportional to the product of the uncertainty $(\sigma(\boldsymbol{x}))$ and ϕ. In locations where the standard deviation is large and the predicted function value is reasonably low, this second term is positive. Again, it seems reasonable to interpret term two of EI as an exploration measure.

Note, however, that Jones et al. [24] reported the following relationships for EI:

$$\frac{\partial \mathrm{EI}}{\partial y} = -\Phi\left(\frac{y_{\min} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right) < 0, \tag{2.4.7}$$

$$\frac{\partial \mathrm{EI}}{\partial \sigma} = \phi\left(\frac{y_{\min} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right) > 0. \tag{2.4.8}$$

This means that EI increases when $y(\boldsymbol{x})$ decreases, and increases when $\sigma(\boldsymbol{x})$ increases. The implication of Equations (2.4.7) and (2.4.8) is that the maximum of EI lies in the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto optimal set. This intuitive result is also formally proven by Ginsbourger [18].

Although this result has been known for some time, it seems that its significance has been overlooked. The fact that the maximum EI design lies in the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto optimal set, directly suggests two objective functions for a multi-objective EI formulation. The first objective is to minimize the predicted objective function value $y(\boldsymbol{x})$. The second objective is to minimize $-\sigma(\boldsymbol{x})$, i.e. maximize the uncertainty. Hence the first objective exploits, and the second objective explores. This is an intuitive result, and presents a simpler multi-objective formulation as compared to the two EI terms. The new parallel EGO algorithm proposed in this thesis, SIMPLE-EGO, is based on this simpler multi-objective formulation.

2.4.3 Weighted Expected Improvement (WEI)

An extension to EI was suggested by Sóbester et al. [48] called Weighted Expected Improvement (WEI). Since numerous authors have labeled the two EI terms as exploitation and exploration respectively, the idea behind the WEI criterion was to give the user additional control over the importance of each term. This criterion is given by

$$\mathrm{WEI}(x) = w(y_{\min} - y(\boldsymbol{x}))\Phi\left(\frac{y_{\min} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right) + (1-w)\sigma(\boldsymbol{x})\phi\left(\frac{y_{\min} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right), \quad (2.4.9)$$

where w is a user selected weight between zero and one. If $w = 1$ is selected the resulting infill criteria is somewhat similar to the Probability of Improvement criterion. Note that WEI can be directly interpreted as the weighted sum formulation of a multi-objective formulation, as discussed in Section 1.4.2. Here the multi-objective formulation would contain the two EI terms as the two objective functions. This is precisely the multi-objective formulation of EGO proposed by Feng et al. [16], EGO-MO.

2.4.4 Generalized Expected Improvement (GEI)

In the original EGO paper, Jones et al. [25] noted that Kriging underpredicts the standard deviation. This leads to EGO first exploiting close to the current predicted objective function minimum, until the uncertainty in the region becomes very low. Then only does EI perform a global search [25]. To address the problems associated with the under prediction of standard deviation, Schonlau [44] suggested the Generalized Expected Improvement (GEI) function. This function has a tuning parameter, which allows the user to decide to put more emphasis on global exploration or local exploitation. The derivation can be found in Schonlau [44]. Since GEI is not used in this thesis, it is not detailed any further.

2.4.5 Maximizing the Infill Criteria (Acquisition Function)

In each iteration the infill sampling criterion maximum must be found. To find this maximum can prove challenging. Most of the infill criteria are multi-modal, and have large areas with value of zero. These two characteristics render gradient based optimizers unsuitable, unless a multi-start method is used. Jones et al. [25] suggested using a branch and bound algorithm. Sasena [43] decided to use the DIRECT [24] algorithm. Finding the maximum of EI is more challenging than we expected. We give details of how we found the maximum of EI in Section 3.3.

Figure 2.4.1 shows how EGO progresses by maximizing the EI function. In the first iteration, a random initial sample is selected and a Kriging surface is fit. Both the true function and the predicted function are plotted. The infill sampling criterion value is shown in green. The first three iterations are shown. In each iteration, the design vector with the maximum infill sampling criterion value is selected and the true objective function value for this design is evaluated.

2.5 Stopping Criteria

Different authors have different perspectives about how and when EGO should terminate. Jones et al. [25] terminated when the maximum EI fell below a certain value, while Sóbester et al. [47] recommended to run EGO for the available time budget.

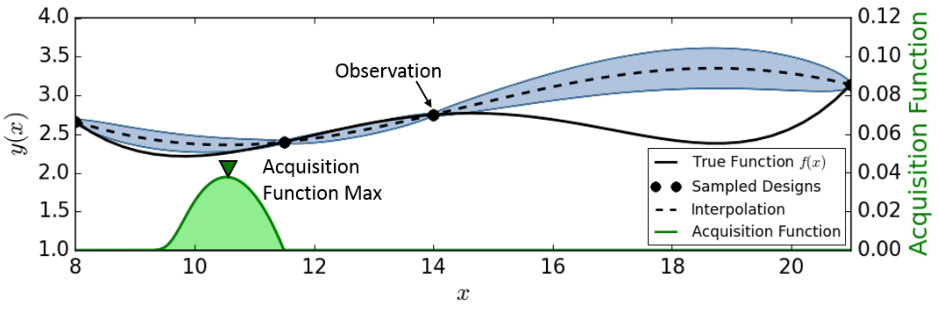
Jones et al. [25] proposed terminating EGO when the maximum EI reaches 1% of the maximum sampled value. Schonlau [44] proposed terminating algorithms using GEI as when GEI$^{1/g}$ reaches 1% of the maximum sampled value.

Henkenjohann and Kunert [21], and Bichon [4] noted this stopping criteria does not take the range of sampled values into account. Due to this phenomena, for this research, both the domain $\boldsymbol{x}$, and the sampled $\boldsymbol{y}_{\text{exp}}$ values are scaled between zero and one.

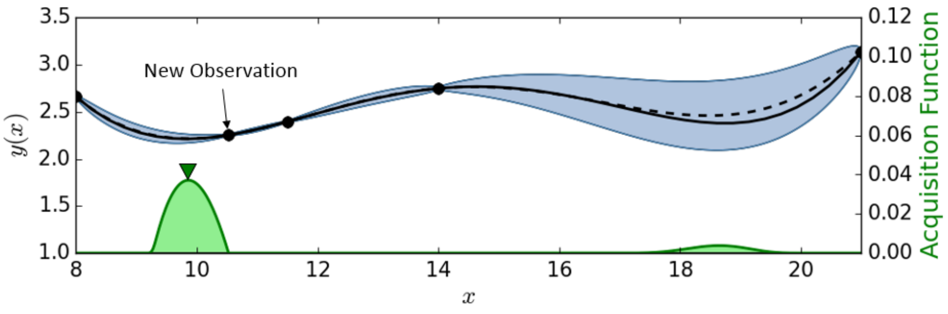
In this research, we are specifically interested in what happens to EGO if it is run for an extended amount of time as one would if following Sóbester et al. [47] recommendation and we do not use termination criteria.

2.6 Computational Implementation

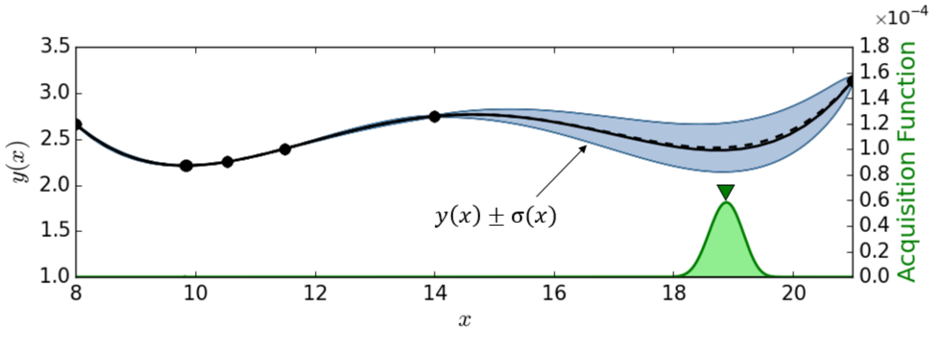
For this research, we have implemented the surrogate methods as described above in Python. Scikit-learn [36] provides a computationally efficient Kriging implementation.



(a) First Iteration



(b) Second Iteration



(c) Third Iteration

Figure 2.4.1: EGO progressing using EI as infill sampling criterion (also known as an acquisition function). In each iteration the design which maximizes the infill sampling criterion is selected. Kriging is refit, and the process repeats.

Furthermore, optimization of the infill sampling criteria is preformed using DE [1] as provided by the Scipy toolbox [26]. More on the computational implementation of this project can be found in Appendix C.

2.7 EGO Problem Dimensionality

As briefly mentioned in Section 1.2 EGO is limited by the number of design variables it can compute and authors typically recommend using not more than 10 design variables [43]. Many recent works benchmarked their adaptations of EGO on mostly 2D to 6D objective functions [2, 9, 16, 56]. However, Boukouvala and Ierapetritou [6] included a test problem with 9 design variables and Feng et al. [15] applied EGO to an Aerodynamic optimization problem with 14 design variables. Although it falls outside the scope of the current research, strategies to extend EGO to higher dimensional space include performing sensitivity studies to reduce the number of design variables, and decomposition of a problem into a series of smaller problems which are more tractable to solve. An interested reader may find a further discussion of how EGO can be adapted for higher dimensional space in Sasena [43].

2.8 Chapter Conclusions

In this chapter we discussed all of the building blocks necessary to implement EGO. We can see that in each iteration EGO performs a trade-off between exploitation and exploration.

We would like to use the inherent multi-objective characteristics of EGO to suggest a parallelization strategy based on the multi-objective nature of EGO. But first we investigate the behavior of classical EGO in the next chapter.

Chapter 3

Classical EGO Characteristics and Behaviour

3.1 Introduction

The aim of this research is to investigate a multi-objective parallelization strategy for EGO. Before this is done, the characteristics and behaviour of EGO are investigated on three scalable unconstrained test functions.

At each iteration of EGO, EI is maximized to determine the next design to be evaluated. We demonstrate that this EI maximization problem is more challenging than literature suggests. Heuristics are suggested to solve the EI maximization problem more robustly.

The chapter is concluded by investigating the exploitation and exploration behaviour of EGO.

3.2 Test Functions

EGO is designed to optimize problems where objective function evaluations are expensive to evaluate. However, in this study we use computationally cheap test functions to allow a detailed study of EGO. By reporting the number of required function evaluations (hypothetical sequential implementation) or the number of required iterations (hypothetical parallel implementation) it gives the reader an indication of how EGO would perform on realistic objective functions. Hence the computational cost of the EGO algorithm is ignored in comparison to the computational cost required to evaluate designs.

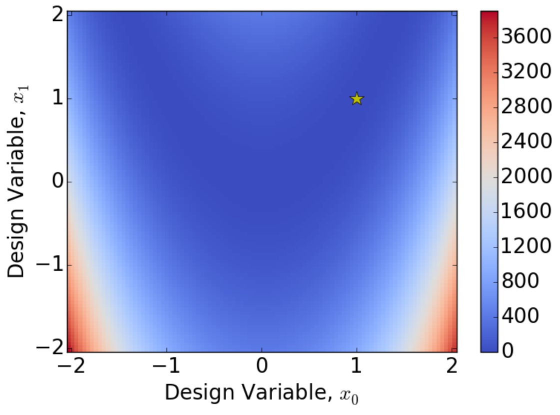
The three test functions are the n-dimensional Rosenbrock, Styblinsky-Tang, and Rastrigin functions. Each test function is considered in 2D, 3D, and 4D.

3.2.1 Rosenbrock

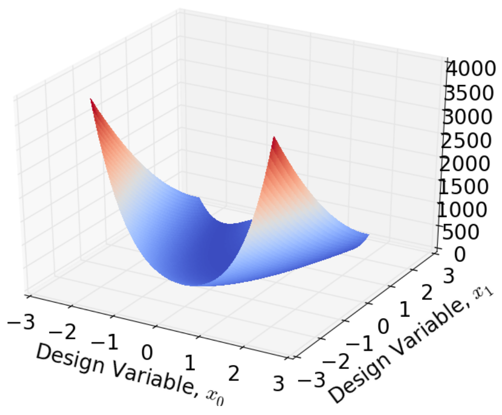
The Rosenbrock Function [51] is often referred to as a banana or valley function and is given by

$$f_{\text{Rosen}}(\boldsymbol{x}) = \sum_{i=0}^{n-2} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right], \tag{3.2.1}$$

where n is the number of dimensions. We evaluate this objective function on $x_i \in$ [-2.048, 2.048]. It has one global minimum of $f_{\text{Rosen}}(1., ..., 1.) = 0$. It is a unimodal objective function in 2D and 3D, and it is multi-modal for $n > 3$ [27]. The valley is easy to find, but progress along the valley to locate the global minimizer is difficult for some algorithms [37]. The 2D version of the Rosenbrock objective function is presented in Figure 3.2.1.The yellow star shows the location of the global optimum.



(a) Contour Plot



(b) Plot

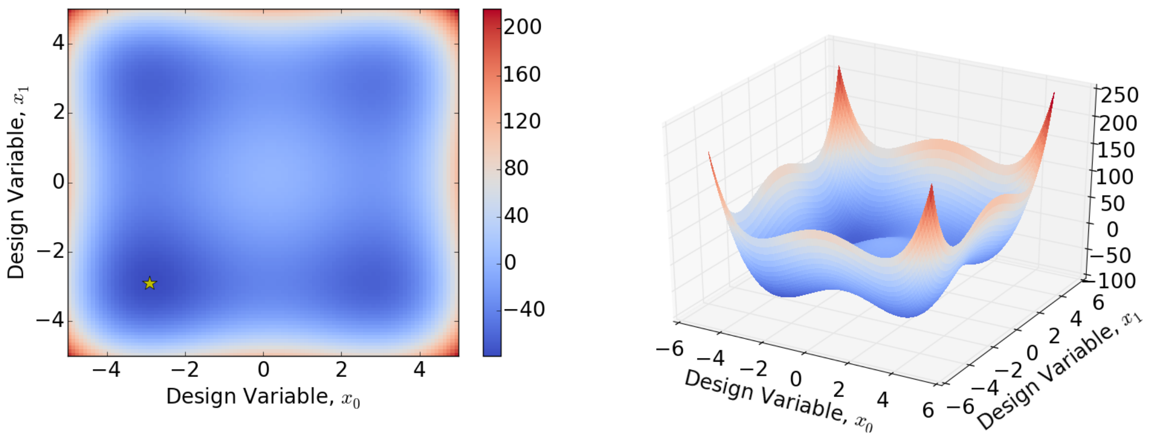
Figure 3.2.1: 2D Rosenbrock Objective Function. Yellow star indicates the global minimum.

3.2.2 Styblinsky-Tang

The Styblinsky-Tang Function [51] is a multi-modal objective function and has 2^n local minimums, where n is the number of dimensions. The function is given by

$$f_{\text{Styb-Tang}}(\boldsymbol{x}) = \frac{1}{2}\sum_{i=0}^{n-1}\left(x_i^4 - 16x_i^2 + 5x_i\right). \tag{3.2.2}$$

This objective function is evaluated in the hypercube $x_i \in$ [-5,5]. It has one global mimimum of $f_{\text{Styb-Tang}}(\boldsymbol{x}^*) = -39.16616570377016n$, where $x_i^* = -2.903533757901 72752$ for all dimensions. This objective function is presented in Figure 3.2.2. The yellow star shows the location of the global optimum.



(a) Contour Plot **(b)** Plot

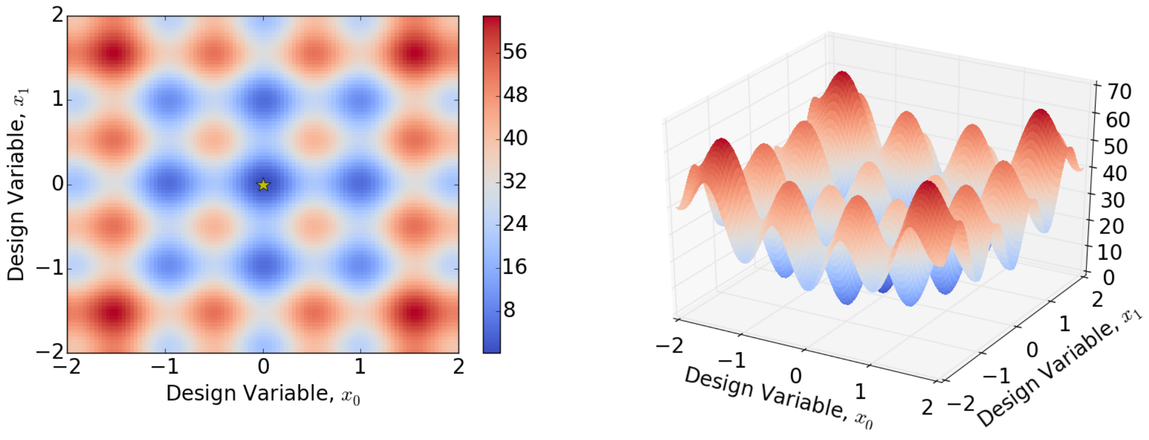
Figure 3.2.2: Styblinsky-Tang Objective Function. Yellow star indicates the global minimum.

3.2.3 Modified Rastigrin Function

The Rastrigin function [51] is a highly multimodal function with many local minima, but only one global minimum. We decided to modify the function slightly, by reducing the domain and increasing the intensity of the parabola. Our modified version of the Rastrigin function is given by

$$f_{\text{Rast}}(\boldsymbol{x}) = 10 + \sum_{i=0}^{n-1}\left[5x_i^2 - 10\cos(2\pi x_i)\right]. \tag{3.2.3}$$

The modified Rastrigin function has 5^n local minima, where n is the number of dimensions. This objective function is evaluated in the hypercube of $x_i \in$ [-2,2]. It has one global mimimum of $f_{\text{Rast}}(0,0) = 0$. This objective function is presented in Figure 3.2.3. The yellow star shows the location of the global optimum.



(a) Contour Plot **(b)** Plot

Figure 3.2.3: Rastrigin Objective Function. Yellow star indicates the global minimum.

3.3 Maximizing EI

Finding the maximum of the EI function is required at each iteration, to locate the next sample design to be evaluated. This maximization problem is much more challenging than suggested in literature, due to

1. the highly multi-modal nature of EI [48],
2. the small basins of attraction of potential maximizers, and
3. for a significant portion of the design domain, the EI function value is close to zero.

Different authors have used different strategies to find the maximum of EI. Jones et al. [25] used a branch and bound algorithm and the definitions

$$\frac{\partial \text{EI}}{\partial y} = -\Phi\left(\frac{y_{\text{min}} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right) < 0 \tag{3.3.1}$$

and

$$\frac{\partial \text{EI}}{\partial \sigma} = \phi\left(\frac{y_{\text{min}} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right) > 0. \tag{3.3.2}$$

Knowing that the decrease of $y(\boldsymbol{x})$ and increase of $\sigma(\boldsymbol{x})$ leads to higher values of EI, Jones et al. [25] used these two values as proxies for EI. They implemented a branch and bound algorithm. In each bounded region, they found the minimum of $y(\boldsymbol{x})$ and the maximum of $\sigma(\boldsymbol{x})$. They used the location of these minimums and maximums and computed EI at these locations. They compared the values and selected the maximum EI value.

Subsequent authors have opted not to use the proxies Jones et al. [25] proposed, but rather opted to use a brute force optimization approach using EI as the objective function. Recognizing the difficulties, Sóbester et al. [48] used a gradient based optimizer with 1000 restarts. Feng et al. [16] used Differential Evolution (DE) with a population size of 20 for 2D test problems. (Typically DE requires population sizes between 10 and 20 for a 2D problem.) We downloaded a EGO code by Muller [35] and found they use a Genetic Algorithm to maximize EI.

Before we ran EGO with the intent to generate data, we wanted to make sure that our implementation of EGO is able to find the maximum of EI. Our initial instinct was to use DE. We used the Scipy [26] computational implementation of DE. The benefit of this implementation is that is allows for a BFGS gradient-based approach to refine the best solution obtained by DE by setting a "polish" argument. The aim is to find the population size that would be sufficient to robustly find the maximum, however we learned that only using DE to optimize EI is not sufficient. The reason is that there is a small basin of attraction for a maximizer of EI (local or global) around the evaluated design with the lowest function value. The difficulty of finding this solution is alleviated by directly using this information.

To illustrate, consider the modified 2D Rastrigin function we defined in Section 3.2.3. We would like to investigate the characteristics of EI after EGO has been run for some iterations, and test whether a given maximization strategy is capable of finding the maximum EI. Therefore we run EGO on this objective function for 44 iterations using DE with a population size of 100 to find the maximum EI in each iteration. After the 44 iterations we are left with a set of sampled designs, $\boldsymbol{X}_{exp}$ and $\boldsymbol{y}_{exp}$. This set of sampled design vectors, is used to set up a EI maximization benchmark problem.

Figure 3.3.1 shows the predicted objective function values, standard deviation, and the EI values plotted in 2D. A legend is presented in Figure 3.3.2. To generate these figures we used a find grid of 120 by 120 points. Note the areas where EI are at its largest, are in the regions where standard deviation are also large. There is no perceivable spike in EI function in the region where the objective function is at its minimum ($\boldsymbol{x} \approx (0.5; 0.5)$). However, when we zoom in around this predicted global minimum, depicted in Figure 3.3.3, the global maximum of the EI function is in fact located here.

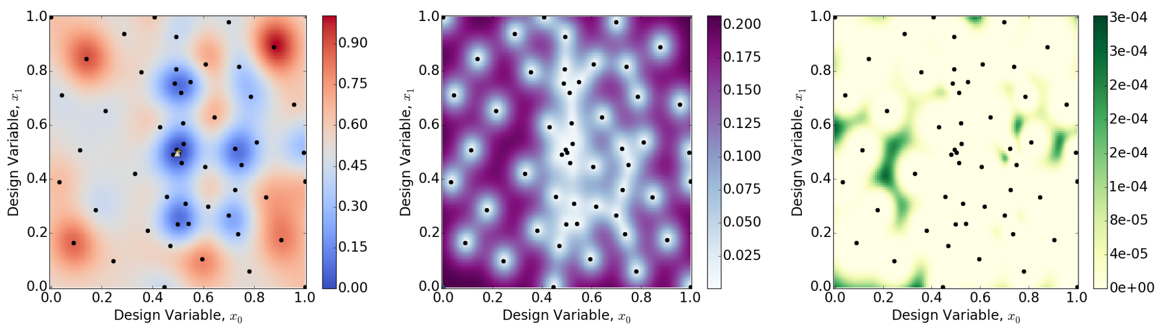
As depicted the basin where EI has a value higher than that of the exploration basins is less than 0.01% of the domain's surface area, while the region around the basin has EI values approaching zero. It is this characteristic which renders EI near impossible to optimize without using additional information.

Since the exploitation basins are located at the global minimum of the Kriging function, we can use this additional information to augment the search for the maximum of EI. In addition, the EI basins of attraction based on exploration are relatively large. Hence an evolutionary strategy using a large population size, many iterations and multi-start should be able to locate these maximizers. The following strategy is therefore proposed to find the maximum of EI:

1. Use DE (population size = $20 \times n$) to find the minimum predicted objective function value. Use a gradient-based method to refine the final result.
   (a) Reduce each one of the design variable's range to the 2% around the minimum predicted objective function value.
   (b) Run DE (population size = $20 \times n$) on EI in this reduced design domain. Use a gradient-based method to refine the final result.
2. Run DE (population size = $50 \times n$) on EI in the full design domain. As before, use a gradient-based method to refine the final result.
3. Compare the results from 1(b) and 2 and select the result with the maximum EI value.

We tested two methods on the problem described above. Only using DE (population size = 100) found this maximum one out of ten times, but the method described in the list above found the maximum of EI ten out of ten times.

Additionally a convergence comparison of EGO is performed on our 3D modified Rastrigin test function, using only DE (population size = 150) to maximize EI or using our proposed method to maximize EI. The results are averaged over the same set of



(a) $y(\boldsymbol{x})$ **(b)** $\sigma(\boldsymbol{x})$ **(c)** EI

Figure 3.3.1: Domain Parameter Plots at the 44th iteration of EGO optimizing the modified Rastrigin function. Note that in this figure, where IE is at its highest the uncertainty is also high. However, (c) does not capture all of the characteristics of EI, therefore Figure 3.3.3 presents a zoomed in section of EI. A legend is presented in Figure 3.3.2.

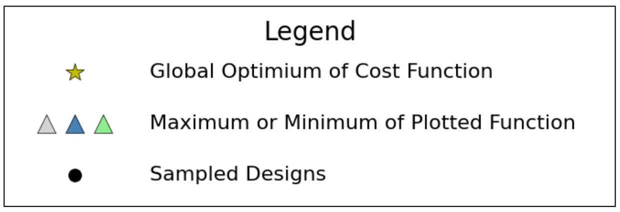

Legend
Global Optimium of Cost Function
Maximum or Minimum of Plotted Function
Sampled Designs

Figure 3.3.2: Legend

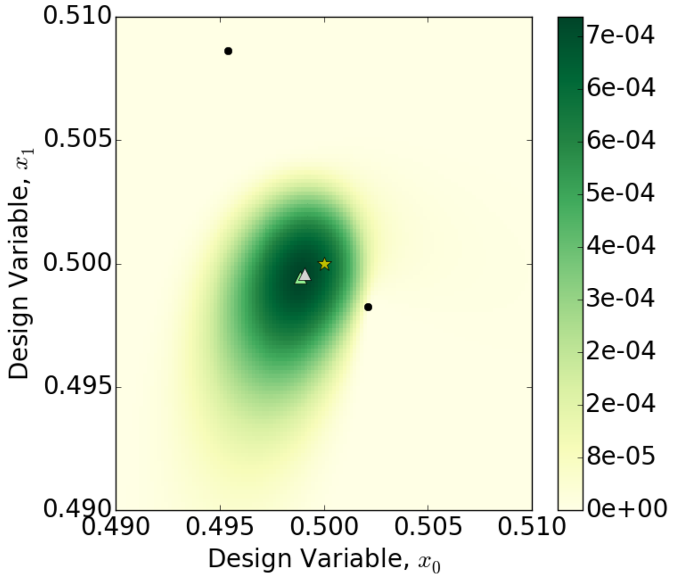


Figure 3.3.3: Zoomed Section of EI in the vicinity of the minimum Kriging value. The maximum EI value in this small basin of attraction is higher than that of the relatively larger basins in Figure 3.3.1.

20 DOEs and are depicted in Figure 3.3.4. The depicted average convergence error is defined by

$$\text{Average}\left(\log_{10}(\Delta y_{\min})\right) = \text{Average}\left[\log_{10}\left(\frac{y_{\min} - f_{\text{true min}}}{f_{\text{true max}} - f_{\text{true min}}}\right)\right]. \tag{3.3.3}$$

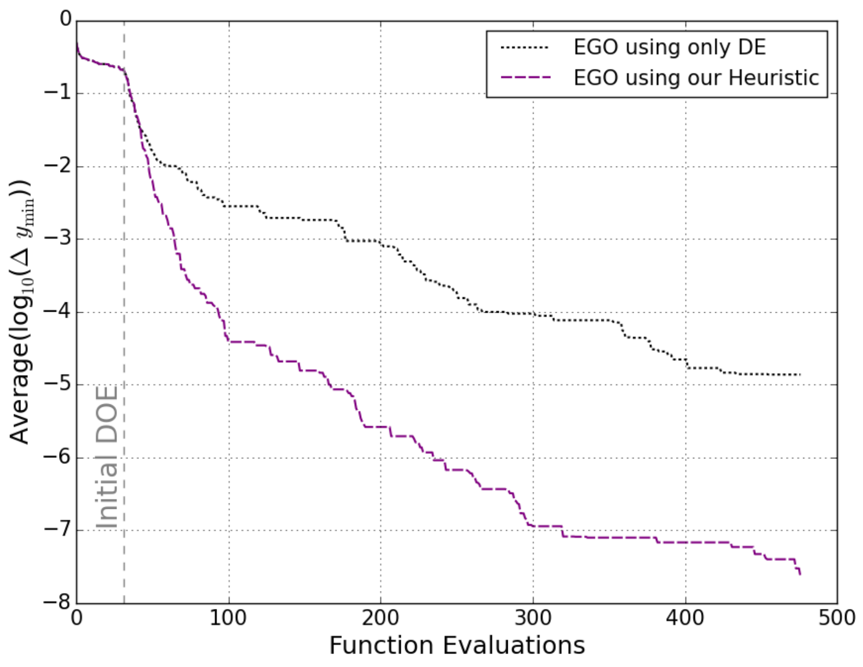


Figure 3.3.4: Convergence Comparison for two Methods Used to Solve EI. The method performing the additional search converged to a higher accuracy than the method only performing the single DE search.

(We give a more in depth overview of our definitions of convergence in Section 4.5). EGO using only one DE run per iteration to maximize EI was only converged to an accuracy of 10^{-5} in the 480 function evaluations, while our proposed method converged to an accuracy of 10^{-7}. This clearly demonstrates the impact that robust maximization of EI has on the performance of EGO.

3.4 EGO Behaviour

3.4.1 Visualization Method

A visual study is now used to investigate the exploitation and exploration behaviour of EGO. We ran EGO on three different 2D objective functions described in Section 3.2.

In each iteration we plot the predicted objective function, the standard deviation, the EI, and the Pareto front. We also map the Pareto optimal designs back to the design domain.

To make the figures more recognizable, we plot each one of the design domain parameters in a different color scheme. Figure 3.4.1 presents a legend to the figures. These color schemes are used throughout this report.

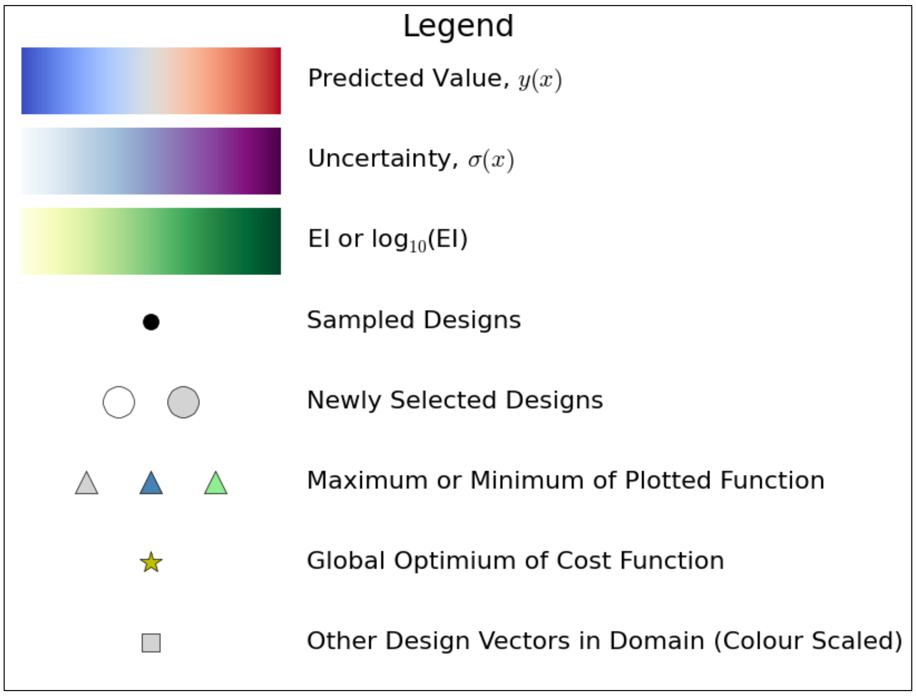
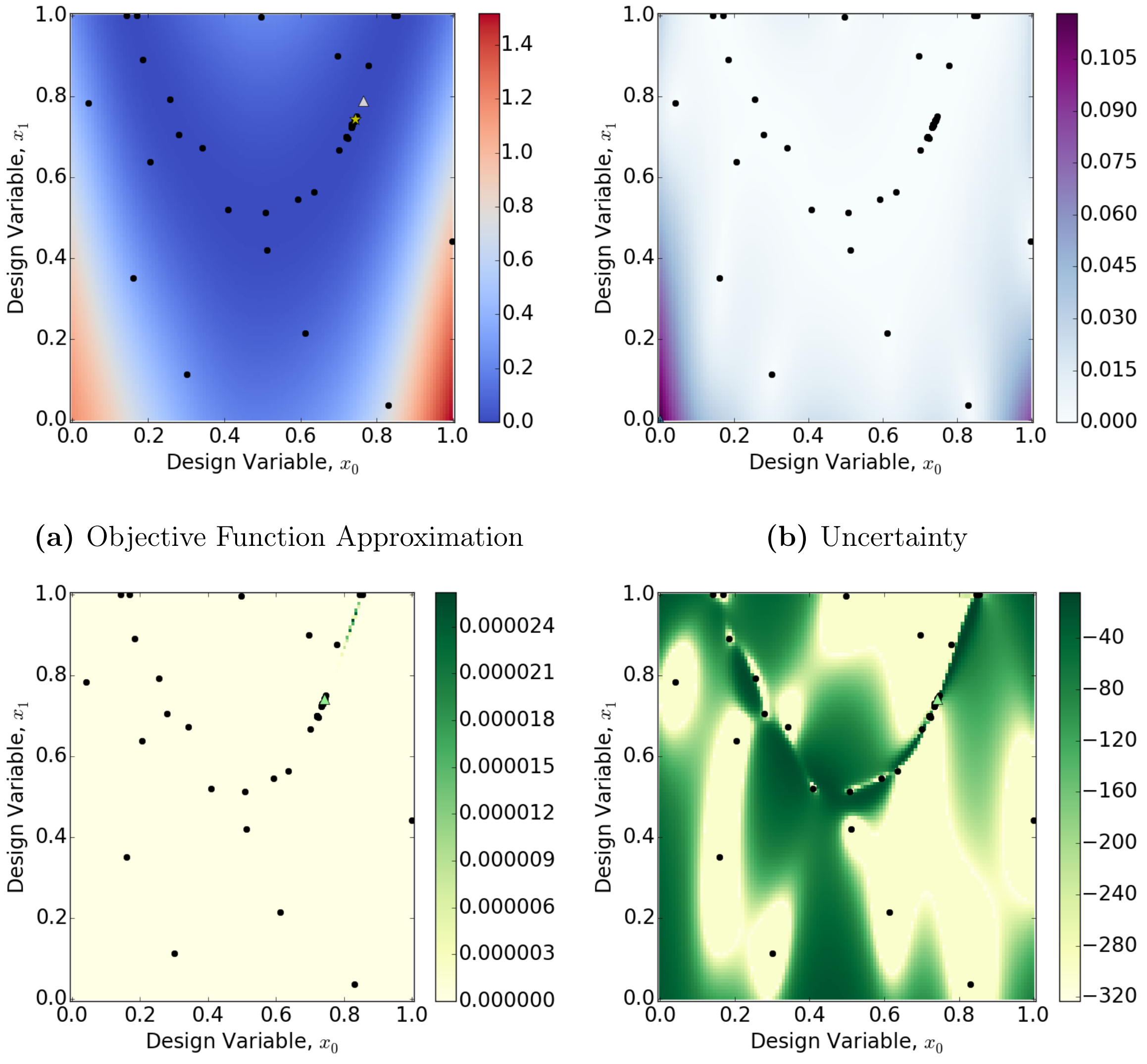

Legend
Predicted Value, $y(x)$
Uncertainty, $\sigma(x)$
EI or $\log_{10}$(EI)
Sampled Designs
Newly Selected Designs
Maximum or Minimum of Plotted Function
Global Optimium of Cost Function
Other Design Vectors in Domain (Colour Scaled)

Figure 3.4.1: Legend for Figures

We have compiled the complete visualization of EGO's progression, and all these figures are included in an electronic appendix (See Appendix B). Here we summarize the essential details.

Figure 3.4.2 (a) depicts the predicted objective function, (b) the standard deviation, (c) the EI, and (d) the $\log_{10}$(EI), all for the 41st iteration. In each figure, the small black circles represent previously sampled designs.

Each function has a maximum or minimum which is represented by a triangle. In Figure 3.4.2 (a), the gray triangle indicates the minimum of the predicted objective function. The blue triangle in the bottom left corner of Figure 3.4.2 (b) indicates the maximum standard deviation ($\sigma(\boldsymbol{x})$). The green triangles in Figures 3.4.2 (c)-(d) indicates the maximum of the EI function, and this becomes the next design to be evaluated.



(a) Objective Function Approximation

(b) Uncertainty

(c) EI

(d) $\log_{10}(\text{EI})$

Figure 3.4.2: Domain Parameter Plots after 41 iterations on the Rosenbrock function. Legend is presented in Figure 3.4.1.

The global optimum of the Rosenbrock function is indicated with a yellow star. In Figure 3.4.2 (a) sampled designs in the vicinity of the global optimum are evident. We also observe that EGO has not sampled extensively in the areas where the predicted function values are high, but it has sampled extensively in the valley. Figure 3.4.2 (b) demonstrates that the standard deviation is very low close to previously sampled designs, and higher when further away.

As we have discussed in Section 2.5, in each iteration we rescale sampled values, $\boldsymbol{y}_{\text{exp}}$, to fit between zero and one. Thus in Figure 3.4.2 (a) the predicted objective function values range between 0 and 1.4. (Rescaling the sampled values between certain bounds before the response surface is fit does not necessarily mean the predicted function values will also fall in these bounds). We also scaled the design domain of each objective function to fall inside the hypercube between zero and one.

As discussed in Section 1.5, we know that in each iteration EGO must perform a trade-off between exploitation and exploration. For this reason, it is helpful to also visualize the design space information in the criterion space. The two objective functions used to construct the criterion space plot are

$$f_{\text{explore}}(\boldsymbol{x}) = y(\boldsymbol{x}) \tag{3.4.1}$$

$$f_{\text{exploit}}(\boldsymbol{x}) = -\sigma(\boldsymbol{x}). \tag{3.4.2}$$

We use the negative of the standard deviation, $-\sigma(\boldsymbol{x})$, because exploration would mean to sample at the maximum of the standard deviation. Figure 3.4.3 shows the criterion space plot for the two objectives defined in Equations (3.4.1) and (3.4.2). As indicated on the plot, the two extremes of the Pareto front represent the minimum of the predicted function and the maximum standard deviation respectively. All the criterion space design vectors which do not fall on the Pareto front are also plotted in the background. The square markers with the dark border constitute the Pareto front, while the markers without a border constitute the design vectors and criterion space values generated with the fine grid in the design domain.

The green triangle in Figure 3.4.3 indicates the maximum EI value, which must fall on the Pareto front [18]. It is this design that will be sampled in the subsequent iteration and it is depicted on the Pareto front to get a sense of the exploit or explore behaviour of EGO in the current iteration.

Differentiating between exploitation or exploration is not trivial. Qualitatively we would like to know when EGO exploits, and when EGO explores. We decided to label iterations that select designs with a predicted function value approximately coinciding with the predicted minimum function value, $y(\boldsymbol{x})$, as exploitation iterations. The other iterations are considered to be exploration iterations.

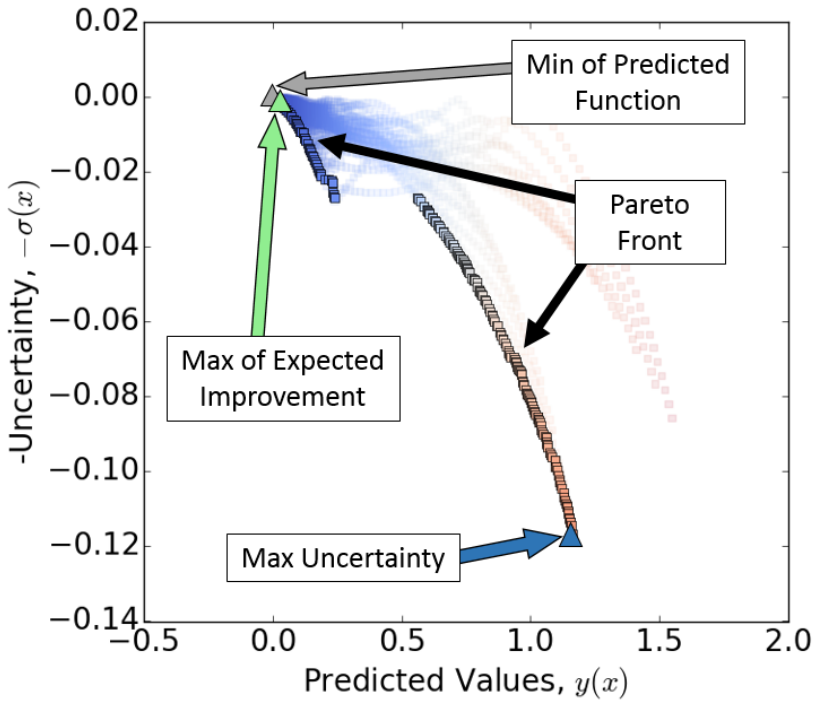


Figure 3.4.3: Criterion space plot of $y(\boldsymbol{x})$ and $-\sigma(\boldsymbol{x})$. As in Figure 1.4.2 a fine grid is used to plot the spread of the points, while the Pareto Optimal set is plotted as outlined squared. Indicated on the figure are the maximum standard deviation (as measure of uncertainty), the minimum of the predicted function, and the maximum EI value.

Figures 3.4.4 (a)-(c) depict three Pareto fronts generated using EGO, and these are used to illustrate our reasoning. In Figure 3.4.4 (a) we display an iteration where EGO selected a design approximately coinciding with the minimum of $y(\boldsymbol{x})$. This we label as exploitation. In Figure 3.4.4 (c) the selected design almost coincides with the maximum standard deviation and we labeled this as strong exploration. In Figure 3.4.4 (b) the selected design lies approximately in the middle of the Pareto front. We decide to label these design selections as weak exploration. Note that the design selected in Figure 3.4.4 (b) has a predicted function value of 0.3. In the case of Figure 3.4.4 (b) the predicted function values range between approximately zero and one. The selected design is a third of the objective function range from the current predicted minimum, therefore categorizing it as an exploration iteration.

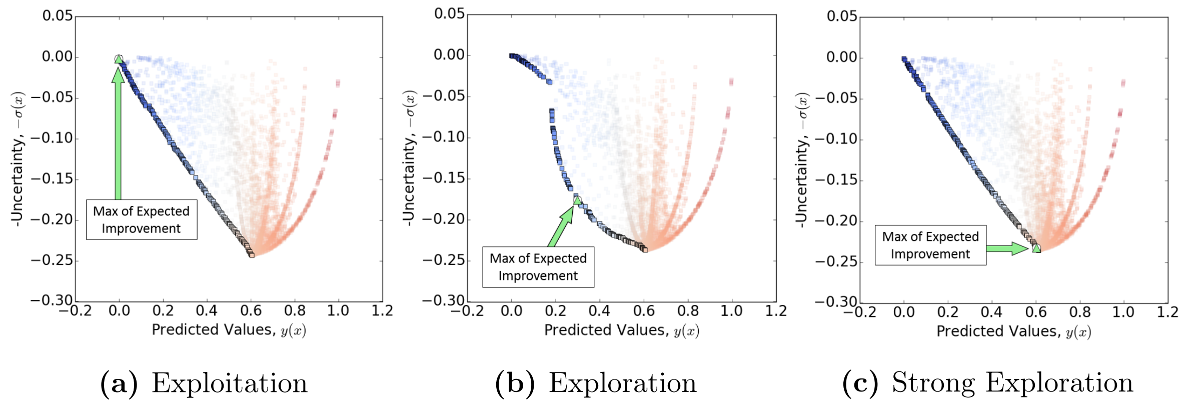


Figure 3.4.4: Using the same topography as Figure 3.4.3, sub-figures (a), (b), and (c) indicate three different cases where designs are selected.

In Figures 3.4.3 and 3.4.4 we color-scaled the markers on the Pareto front to match the color scaling of the predicted function values. In Figure 3.4.5 we plot the Pareto front designs in the design domain. The color scaling is intended to help the reader to relate the designs from the criterion space to design space. We also indicate the selected designs with a large white circle. For classical EGO the selected design corresponds with the maximum EI. However, when more design vectors are selected per iteration, it depends on how the algorithm developer decides select the designs.

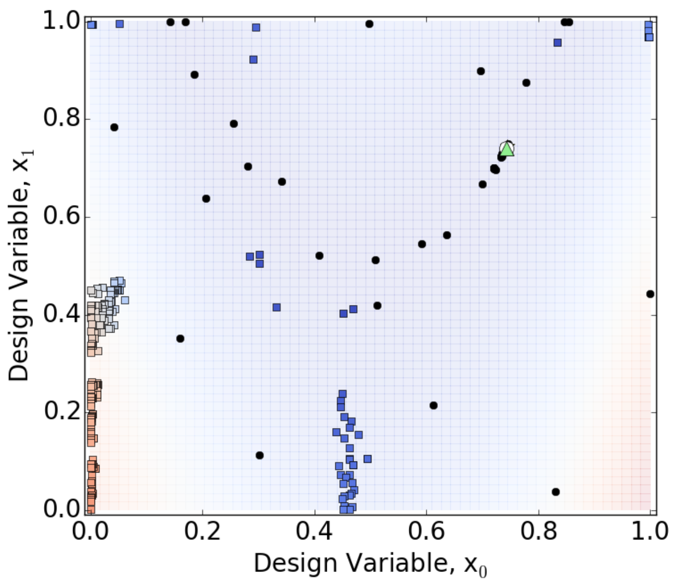


Figure 3.4.5: Exploitation Exploration Pareto Optimal Designs can be mapped back to the Design Domain as illustrated in Section 1.4.

3.4.2 Results and Discussion

The qualitative numerical experiments conducted in this chapter has indicated that EGO has the following behaviour:

1. Phase 1: Objective function is not yet well approximated. Exploitation designs are added, but because the additional sample contributes to the accuracy of objective function, the location of the predicted objective function changes in each iteration. The result is exploitation design vectors which help with the overall accuracy of the response surface.
2. Phase 2: The predicted objective function has become more accurate, and EGO has found a local basin and exploits this basin for a few iterations.
3. Phase 3: EGO has explored this local basin sufficiently, and if the standard deviation in the rest of the domain is still high, EGO now starts to explore. If EGO finds another basin with low predicted objective function values, EGO repeats phase 2 for the new basin.

Our qualitative assessment corresponds with Schonlau's [44] assessment that EGO will usually first exploit then explore. Furthermore, it occurs often that in one iteration EGO will exploit followed immediately with an iteration where EGO explores. In general, the transition in EGO between exploitation and exploration is abrupt. In Chapter 4 we use our understanding of the bare basics of EGO to propose a parallel sampling strategy for EGO.

Chapter 4

Multi-objective Parallelization of EGO

4.1 Introduction

Classical EGO only selected one set of design parameters to sample in each iteration. In general, the modern day computing environment allows simultaneous evaluation of different sets of design parameters. For instance, clusters can run several simultaneous function evaluations, while even some laptops have 8 or 16 cores and are also capable of doing parallel runs.

Additional sampling is the method which EGO employs to increase its knowledge about the problem. The rate at which EGO's knowledge about the problem increases would also increase if EGO were to select multiple samples per iteration.

In this chapter an existing multi-objective parallelization of EGO, EGO-MO [16] is first discussed to provide a reference point. We then propose our own parallelization strategy for EGO, that is based on the exploitation and exploration objective funtions on which EI is based. The proposed algorithm is named Simple Intuitive Multi-objective ParalLElization of Efficient Global Optimization (SIMPLE-EGO).

4.2 Comparison Method: EGO-MO

EGO-MO [16] uses the speculated exploitation-exploration characteristics of the two terms of the EI function. The algorithm starts by solving the Pareto front of the multi-objective optimization problem defined by:

$$\text{Minimize} \quad f_{\text{EI1}}(\boldsymbol{x}) = -(y_{\min} - y(\boldsymbol{x}))\Phi\left(\frac{y_{\min} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right), \tag{4.2.1}$$

$$f_{\text{EI2}}(\boldsymbol{x}) = -\sigma(\boldsymbol{x})\phi\left(\frac{y_{\min} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right). \tag{4.2.2}$$

Figure 4.2.1 gives a summary of the EGO-MO algorithm. EGO-MO uses a Multi-Objective Evolutionary Algorithm to find a Pareto front. Having found the Pareto optimal set, designs on the Pareto front that are closer than some Euclidean distance to previously sampled designs are discarded. Thereafter, EGO-MO starts selecting the Pareto front designs to sample. EGO-MO first selects the two extreme Pareto front designs. To select the remaining designs, EGO-MO uses c-means clustering. Once the Pareto front designs have been clustered, EGO-MO selects a design from each cluster that maximizes the Euclidean distance from the other designs in the domain.

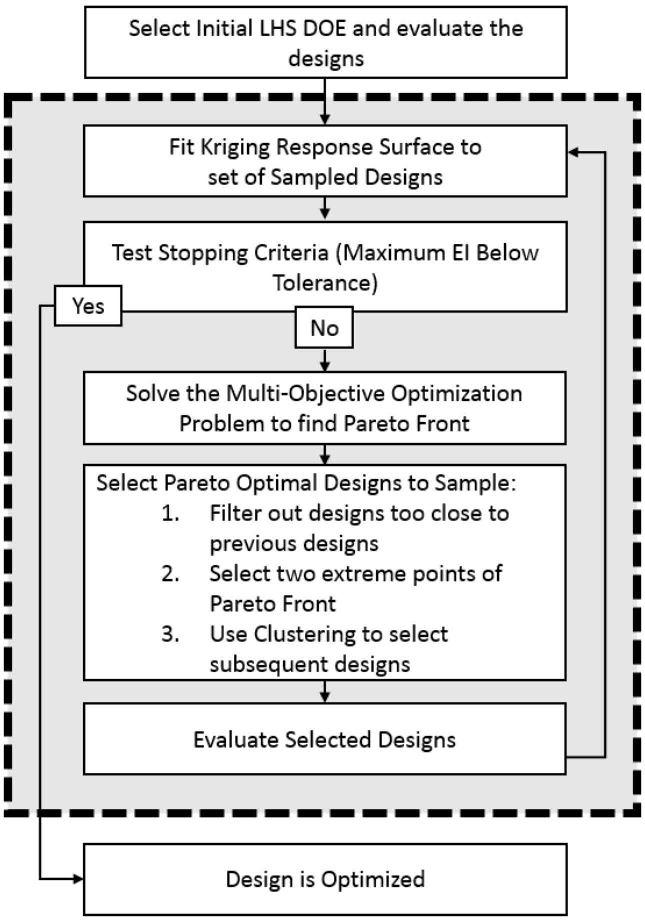
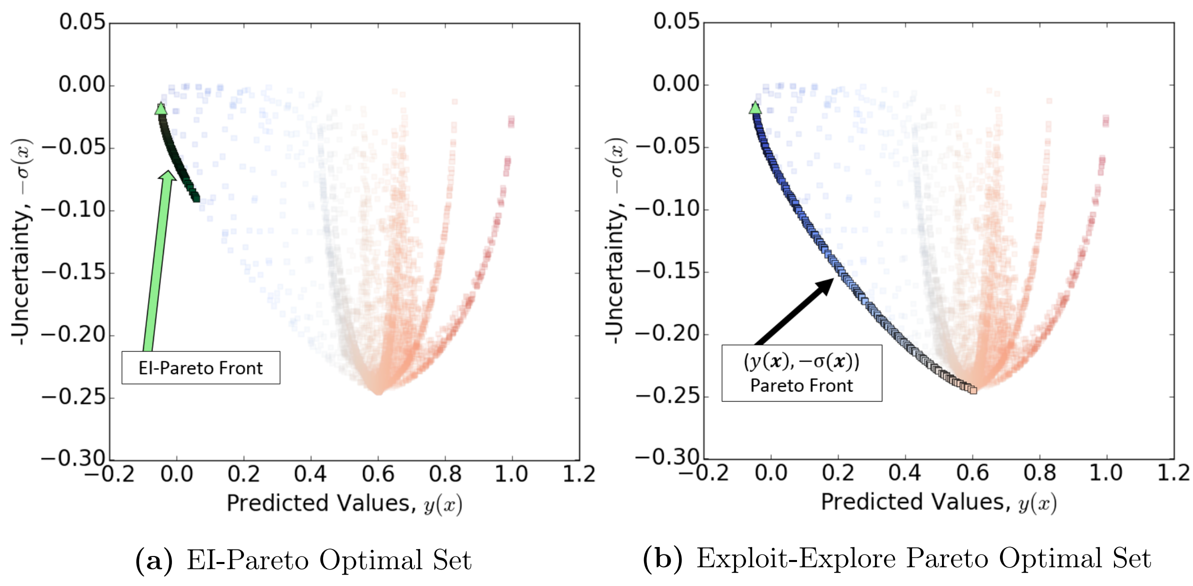

Select Initial LHS DOE and evaluate the designs
Fit Kriging Response Surface to set of Sampled Designs
Test Stopping Criteria (Maximum EI Below Tolerance)
Yes
No
Solve the Multi-Objective Optimization Problem to find Pareto Front
Select Pareto Optimal Designs to Sample:
1. Filter out designs too close to previous designs
2. Select two extreme points of Pareto Front
3. Use Clustering to select subsequent designs
Evaluate Selected Designs
Design is Optimized

Figure 4.2.1: EGO-MO Algorithm

In our implementation of EGO-MO, we followed this same procedure. However, instead of c-means clustering, we use k-means clustering as implemented by the Sklearn toolbox [36].

We present a more in depth analysis of the behaviour and characteristics of EGO-MO in Appendix D, but we present a short summary of our results here.

In each iteration EGO intelligently selects a design to sample which lies on the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front. For the most part, EGO-MO follows this behaviour. To differentiate between the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ and the Pareto front solved by EGO-MO this text refers to the EGO-MO Pareto front as the EI-Pareto front. The EI-Pareto front usually also lies on the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front, but covers a smaller portion of the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front. Figure 4.2.2 (a) illustrates the EI-Pareto front, while (b) illustrates the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front for the same response surface. Both the plots are in the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ criterion space. Note the EI-Pareto optimal designs covers a reduced section of the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front.



(a) EI-Pareto Optimal Set **(b)** Exploit-Explore Pareto Optimal Set

Figure 4.2.2: $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Criterion Space Plot. It can be seen that the EI-Pareto front only covers a small portion of the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front. Also plotted on the figure are the function values generated by a find grid as explained in Figure 1.4.2.

As a result the design vectors selected by EGO-MO are often clustered. We discuss and illustrate this in Appendix D. Clustering can be a desirable feature, but in some cases the selected design vectors are clustered where the predicted objective function values are high. Such behaviour is typically is not desirable.

In Section 3.3 we discussed the difficulties associated with finding the Maximum of EI. EGO-MO encounters the same challenges. As the algorithm progresses the EI terms reduce to zero and towards the later stages, the EI-multi-objective optimization problem becomes difficult to solve. Figure 4.2.3 illustrates a criterion space plot after EGO-MO has sampled 220 designs. In this case NSGA-II was unable to solve the EI-Pareto optimal set (yellow-green color scaling), but was able to find the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front (blue-red color scaling). More details are presented in Appendix D.

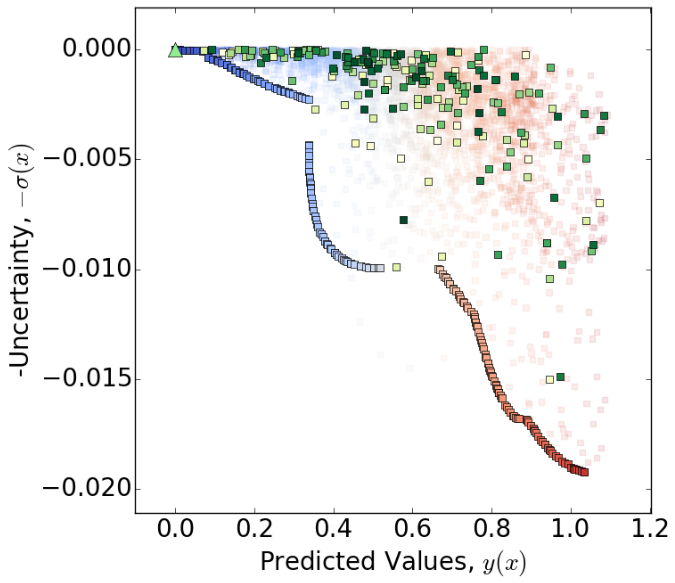


Figure 4.2.3: NSGA-II unable to solve EI-Pareto Front: The outlined blue and red colour scaled squares indicate the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front which was solved by NSGA-II. The outlined yellow and green colour scaled squares indicate the output from NSGA-II trying to solve the EI-Pareto front. It is evident NSGA-II is no longer capable of solving the EI-Pareto front. Also plotted on the figure are the function values generated by a find grid as explained in Figure 1.4.2.

To summarize, both the EI-Pareto front and the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front include the maximum EI design. The major differences between the two criterion spaces, is that the EI-Pareto front covers small region of the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front. Furthermore, towards the end of an optimization run, the EI Pareto front becomes difficult to solve, while the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front can still be solved. Despite these drawbacks EGO-MO performs well as an optimization algorithm and is used as a comparison for SIMPLE-EGO.

4.3 SIMPLE-EGO

Recall from Section 2.4.2 that Ginsbourger [18] proved that the maximum EI design is in the $(y(\boldsymbol{x}); -\sigma(\boldsymbol{x}))$ Pareto optimal set. This result directly suggests the two objective functions for a multi-objective EGO algorithm: minimize the predicted function value $y(\boldsymbol{x})$ while maximizing the uncertainty $\sigma(\boldsymbol{x})$. It is intuitive to label these two objective functions as explore and exploit objective functions. Compared to the EGO-MO objectives (the two terms of the EI function), the exploit-explore interpretation is more direct. Our proposed objective functions are also directly available from any standard Kriging implementation.

4.3.1 Solving and selecting Pareto front designs

The Pareto optimal set is solved using a multi-objective optimization algorithm called Non-dominated Sorting Genetic Algorithm (NSGA-II) [13]. In each iteration of SIMPLE-EGO, we solve for 200 Pareto optimal designs approximately equispaced along the Pareto front in criterion space.

The next step is to develop a strategy to select designs from the Pareto optimal set, to be evaluated in the subsequent iteration. Consider the Pareto front depicted in Figure 4.3.1 (a reprint of Figure 3.4.3). For convenience the legend explaining the colour scheme is reprinted in Figure 4.3.2. Notice that the two extremes of the Pareto front are the minimum predicted function value, and the maximum standard deviation. The maximum EI design is also indicated. For SIMPLE-EGO, we always select the designs associated with the minimum predicted function value, the maximum standard deviation, and the maximum of EI. In Sections 4.3.2 and 4.3.3 we discuss how the remainder of the designs are selected.

4.3.2 Design Selection

Often users will have in depth knowledge about the problems they are optimizing. Their different objective functions will require algorithms with different characteristics. It would be ideal from a user's perspective to be able to control the main characteristics of the chosen algorithm using simple tuning parameters. This section discusses how we biased SIMPLE-EGO to have more explorative or exploitative characteristics, using

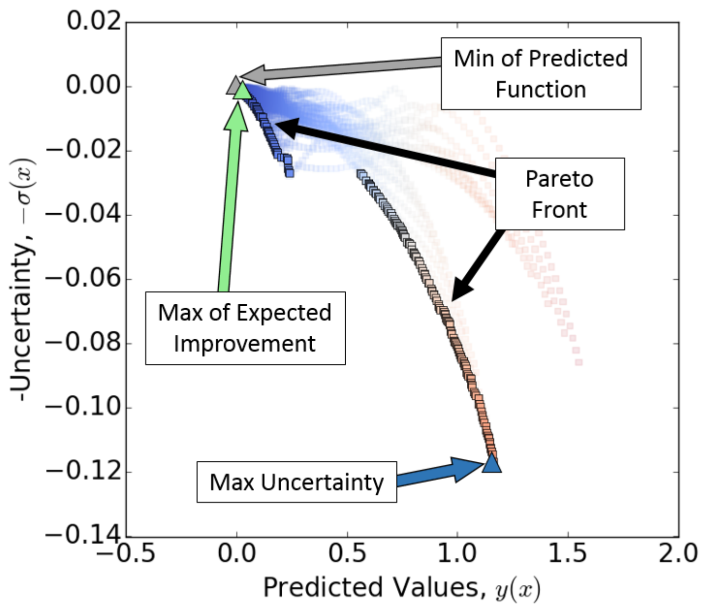


Figure 4.3.1: Reprint of Figure 3.4.3: Criterion space plot of $y(\boldsymbol{x})$ and $-\sigma(\boldsymbol{x})$. As in Figure 1.4.2 a fine grid is used to plot the spread of the points, while the Pareto Optimal set is plotted as outlined squared. Indicated on the figure are the maximum standard deviation (as measure of uncertainty), the minimum of the predicted function, and the maximum EI value.

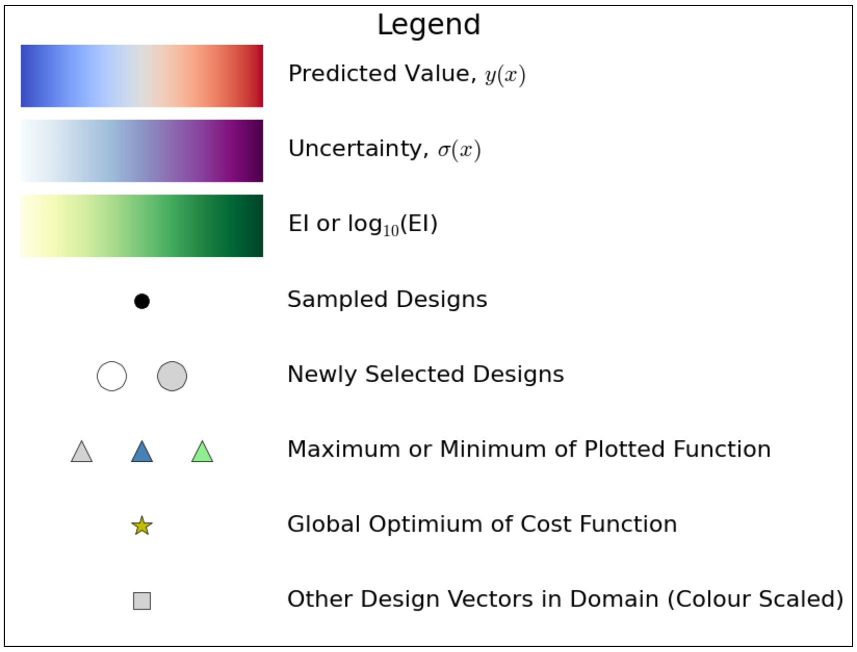
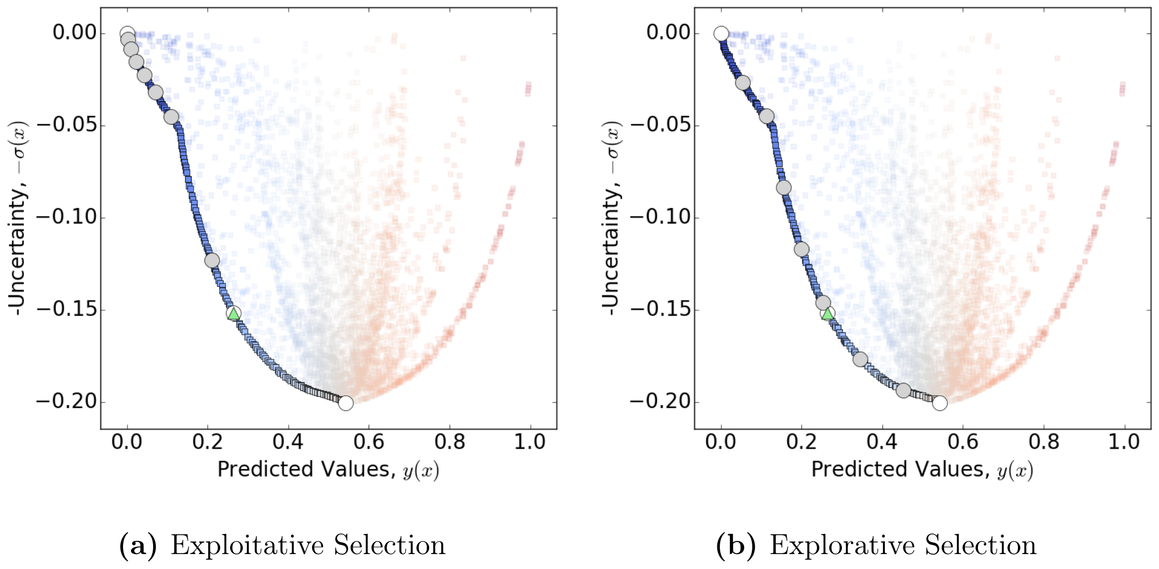

Legend
Predicted Value, $y(x)$
Uncertainty, $\sigma(x)$
EI or $\log_{10}$(EI)
Sampled Designs
Newly Selected Designs
Maximum or Minimum of Plotted Function
Global Optimium of Cost Function
Other Design Vectors in Domain (Colour Scaled)

Figure 4.3.2: Legend for Figures

tuning parameters.

Presume that a Pareto optimal set of designs have been found. We can either select designs close to the minimum predicted function value, close to the maximum standard deviation, or evenly spread out over the Pareto front. Figure 4.3.3 shows two different options we consider reasonable to select the designs. Figure 4.3.3 (a) selects the designs on the Pareto front that are close to the predicted minimum function value. Thus this method for design selection we classify as exploitative. Figure 4.3.3 (b) selects designs evenly spread out over the Pareto front, hence this method is labeled as explorative.



(a) Exploitative Selection **(b)** Explorative Selection

Figure 4.3.3: Criterion space plots of different options for selecting designs on Pareto front. The selected designs are plotted in large white or gray circles The spread of the design selection can bias the search towards (a) exploitation or (b) exploration. The rest of the figure follows the typology of Figure 3.4.3.

In Figure 4.3.4 we show a selection of designs biased towards maximum standard deviation. At first glance one would expect this method to have good explorative characteristics. However, this is not the case and Section 4.4 provides details why this selection is not wise.

4.3.3 Parameterization of Design Selection

In this section we define the parameters we used to tune the design selection to be more explorative or exploitative. The NSGA-II method used to solve the Pareto front, utilizes

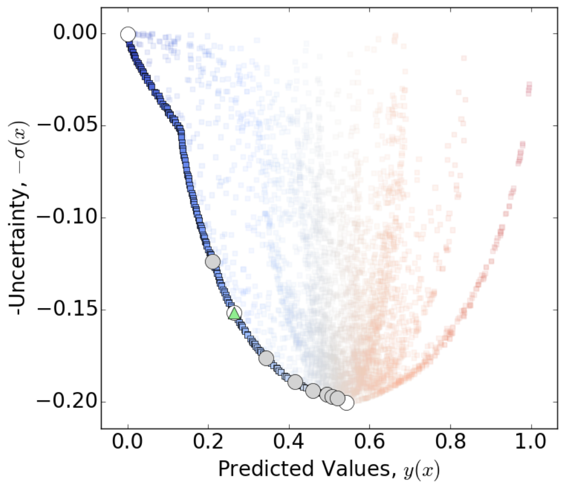


Figure 4.3.4: Unwise Choice for Explorative Design Vector Selection when compared to Figure 4.3.3 (b).

crowding to ensure an even spread of Pareto front designs in the criterion space. We use an ordered list of the Pareto optimal designs to "linearize" the Pareto front, as depicted in Figure 4.3.5. The selection of Pareto optimal designs then reduces to selecting indices.

Once we have linearized the Pareto front, Figure 4.3.6 illustrates the method we used to decide which of the Pareto optimal designs to select. We use a transfer function which maps linearly spaced input to a log-spaced output. Both input and output are scaled between zero and one. By parameterizing the transfer function, the selected designs can be biased towards exploration or exploitation.

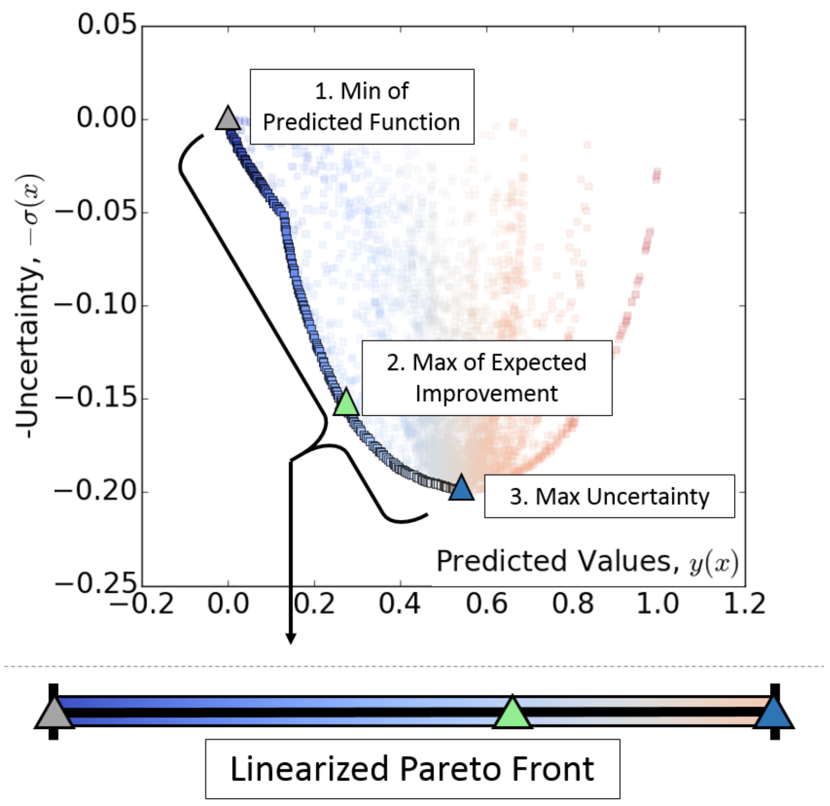


Figure 4.3.5: NSGA-II solves a Pareto Optimal set of designs approximately equally spaced on the Pareto Front. We use this structure, and linearizing the Pareto front in order to select the designs to sample.

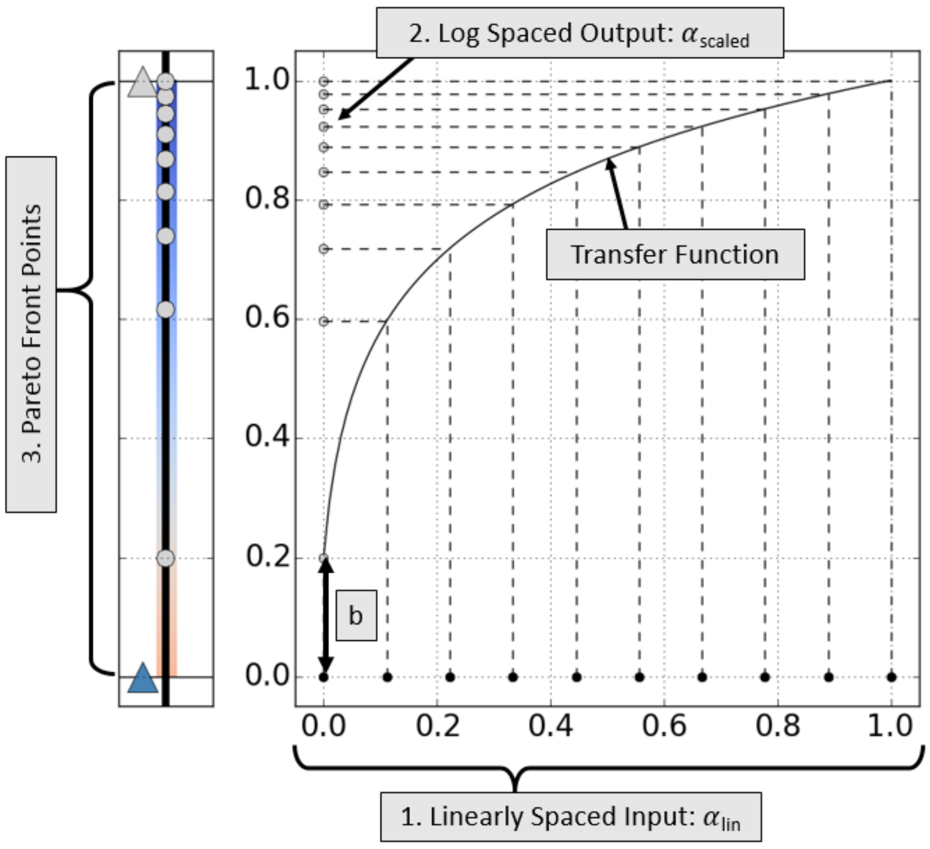

2. Log Spaced Output: α_{scaled}
Transfer Function
3. Pareto Front Points
b
1. Linearly Spaced Input: α_{lin}
0.0
0.2
0.4
0.6
0.8
1.0

Figure 4.3.6: By using a tuning parameter, the selection can be biased towards the exploitation or exploration end of the Pareto front. The transfer function depicted defines our method of biasing the selection.

We define the linear spaced input as $\boldsymbol{\alpha}_{\text{lin}} = [\alpha_0, \alpha_1, \ldots, \alpha_{n_i}]$, where n_i is the number of designs sampled per iteration. α_0 is zero, while α_{n_i} is one. The Pareto optimal designs are equally spaced in the criterion space. We introduce two parameters, the scale parameter a and the distance parameter b. The scale parameter controls the crowding of points towards one side of the Pareto front. For instance, if $a = 0.01$ is selected, the output of the transfer function is linearly spaced, while if a is much larger, such as $a = 60$, the output of the transfer function is clustered to one side of the Pareto Front. The distance parameter, b, controls whether the furthest endpoint of the Pareto front is included or not. As indicated on Figure 4.3.6, b is the distance between the furthest selected design and the endpoint of the Pareto front.

The transfer function is defined as

$$\boldsymbol{\alpha}_{\text{scaled}} = \left(\frac{\log_{10}(a(\boldsymbol{\alpha}_{\text{lin}}) + 1)}{\log_{10}(a + 1)} + c \right) \times \frac{1}{1 + c}, \tag{4.3.1}$$

where c is based on the distance parameter, $c = b/(1-b)$. To switch between exploitation and exploration biased selection, we reverse the values of $\boldsymbol{\alpha}_{\text{scaled}}$, using

$$\text{Reverse}(\boldsymbol{\alpha}_{\text{scaled}}) = -(\boldsymbol{\alpha}_{\text{scaled}} - 1). \tag{4.3.2}$$

To use the $\boldsymbol{\alpha}_{\text{scaled}}$ values to select designs from the Pareto front, the Pareto front indices are given by

$$\text{round}(\boldsymbol{\alpha}_{\text{scaled}} \times \text{Number of Pareto front designs}). \tag{4.3.3}$$

To illustrate the effect these two parameters have, we consider three cases. We generate a method which exploits more than it explores, a method which samples uniformly over the Pareto front, and a method which selects designs close to the maximum standard deviation design. Figure 4.3.7 depicts the three Cases, when ten designs must be selected from the Pareto front. The three designs which are always selected are indicated in white on the figures (the end-points and the maximum EI design). The grey markers are the additional selected designs.

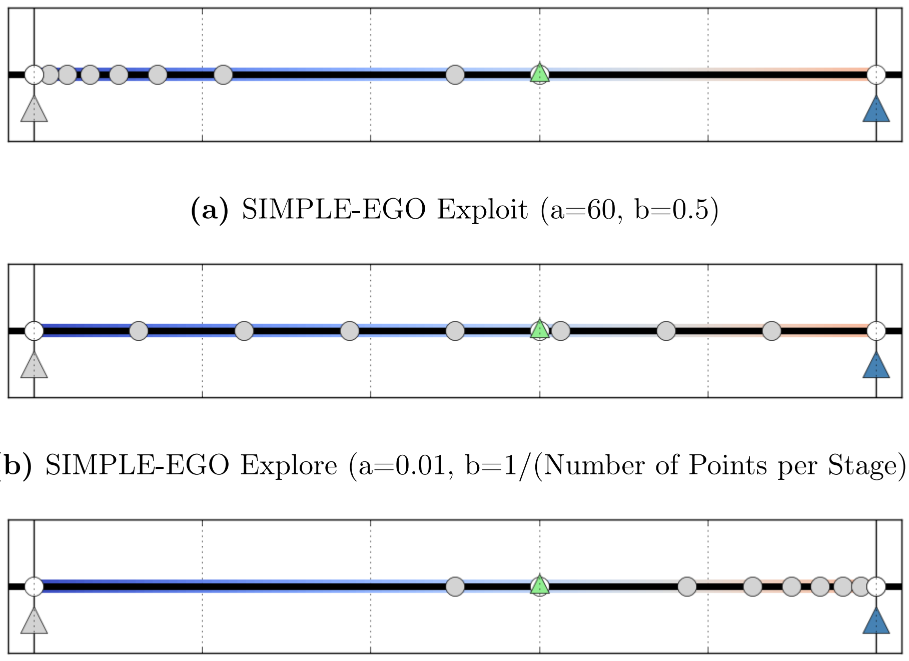
4.3.4 Alternative design selection

The methods considered in this thesis select designs from the Pareto optimal set, using metrics defined in the criterion space. Sampling methods defined in the design space should also be considered in future work, such as the clustering selection method employed by EGO-MO.

4.4 SIMPLE-EGO Visualization

We ran the three SIMPLE-EGO methods depicted in Figure 4.3.7 on the 2D objective functions we described in Chapter 3. We ran these methods for three and ten additional designs per iteration. The results from these runs are summarized graphically and are available in the electronic appendix (See Appendix B).

The behaviour of the SIMPLE-EGO Exploit and the SIMPLE-EGO Explore methods is as desired. These methods result in designs that cover the design domain, and when designs are added in clusters, they are generally added close to the predicted objective function minimum. However, the characteristics of the SIMPLE-EGO Unwise Explore method are undesirable. As an example of this unwanted behaviour, Figure 4.4.1 shows



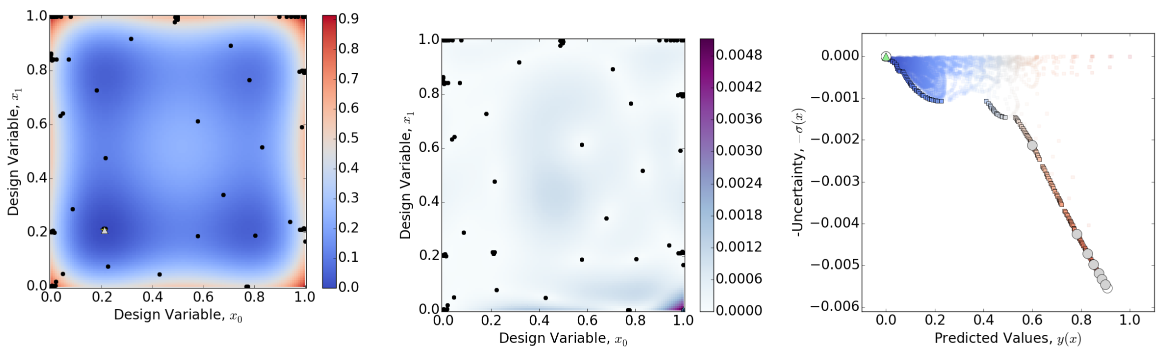
(a) SIMPLE-EGO Exploit (a=60, b=0.5)

(b) SIMPLE-EGO Explore (a=0.01, b=1/(Number of Points per Stage))

(c) SIMPLE-EGO Unwise Explore (a=60, b=0.5, reversed)

Figure 4.3.7: Different Options for selecting ten designs on the Pareto Front. The left side of the figures above represents exploitation, while the right side represents the exploration. The circles represent the designs selected to sample. The more the designs are selected towards the left, the more the algorithm exploits. However, selecting designs towards the right of does not lead to desirable exploitation characteristics (Section 4.4). The endpoints are always included for sampling, as well as the design which maximizes EI.

the design domain plots as well as the criterion space plots for the SIMPLE-EGO Unwise Explore method's 9th iteration. In this case 10 designs are added per iteration. Figure 4.4.2 displays the design space plot of where the selected designs lie in the design domain. The selected designs lie in a tight cluster in the corner of the domain. Tight clustering is acceptable if these designs lie close to the minimum predicted objective function value. When designs are sampled close to the minimum predicted objective function value, this increases the accuracy of the Kriging surface in this important region. The aim of the exploration designs is to locate new local basins. The same information provided by the eight designs added in a tight cluster in Figure 4.4.2 could have been provided by a single design, thus wasting valuable computational resources. Therefore we would not recommend using such a method, and we discard this method of selecting designs for the remainder of this report.



(a) Predicted Objective Function

(b) Uncertainty

(c) Criterion Space Plot

Figure 4.4.1: Design space and criterion space plots at iteration 9 for the Unwise Explore method running on the Styblinsky-Tang function. Notice how designs are selected towards the exploration side of the Pareto front in (c).

Now that we have visually studied the different methods, we would also like to investigate their convergence characteristics. Section 4.5 discusses our methodology and results.

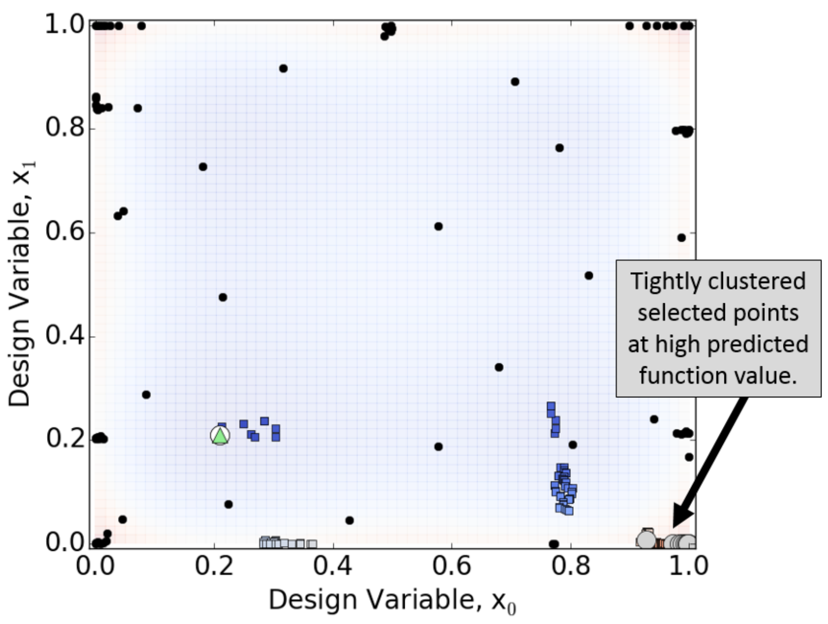


Figure 4.4.2: Design domain plot of Pareto front and selected design vectors. The small black circles represent the previously sampled designs, while the large white and gray circles represent the designs selected to sample in this iteration. In this case the selected designs are tightly clustered around the area of maximum uncertainty, at a high function value. Such behaviour is not favourable.

4.5 Convergence Comparison

4.5.1 Method

In this section we compare their convergence performance of EGO, EGO-MO [16], and the two SIMPLE-EGO methods.

As any algorithm progresses the difference between the true minimum and the minimum sampled value should decrease. This decrease of error as the number of iterations increases, is referred to as the convergence of the function, and it is defined as

$$\text{Convergence Error} = \log_{10}(\Delta y_{\text{min}}) = \log_{10}\left(\frac{y_{\text{min}} - f_{\text{true min}}}{f_{\text{true max}} - f_{\text{true min}}}\right). \tag{4.5.1}$$

Because each LHS DOE is based on a randomized design, the convergence results for a certain method on a certain objective function will be different if a different LHS DOE is used. Thus we average the convergence error over 20 runs for each method, using the same set of initial DOEs for the different methods. The methods were run on the nine different test functions, for the 2D, 3D and 4D instances. Using the same test functions in varying dimensions should give us an indication of how the methods scale. The initial DOE sizes were 10, 32, and 43 for the 2D, 3D, and 4D cost functions respectively.

The classical EGO algorithm proceeded sequentially, selecting one new design per iteration. EGO-MO and SIMPLE-EGO proceeded in parallel and selected three or ten additional designs per iteration.

4.5.2 Individual Results and Discussion

Before we present averaged results, we first wanted to check whether the average is indicative of the convergence of the method. For each one of the methods, on each one of the test functions we constructed an individual convergence plot.

Figure 4.5.1 shows an example of an individual convergence plot. We plot the mean, maximum and minimum error as the method progresses. Furthermore, at each iteration, the standard deviation can also be computed for the 20 runs. In the plot, we shade one standard deviation around the mean. If the difference between the minimum and the maximum, and the standard deviation are small, it would be safe to assume that the mean is a good indication of the convergence characteristics of that method on that test function.

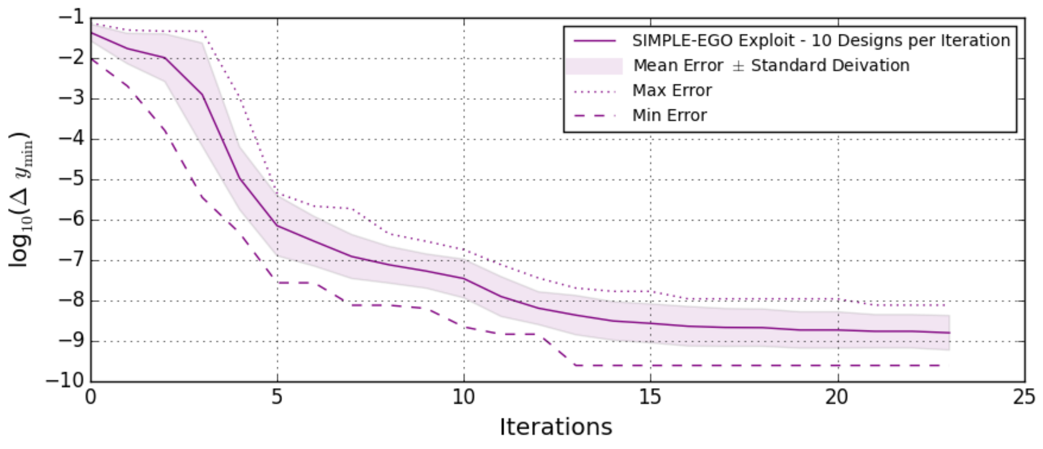


Figure 4.5.1: Convergence Plot of SIMPLE-EGO Exploit on the Styblinsky-Tang Objective Function

In Figure 4.5.2 we present the case with the most variation: SIMPLE-EGO, three designs per iteration, on the 3D Modified Rastrigin test function. As a comparison we also present the results for the same method, on the 2D version of the same test function in Figure 4.5.3), and the results from running SIMPLE-EGO Explore with ten samples per stage on the 3D modified Rastrigin test function in Figure 4.5.4. We encountered large variances specifically for the SIMPLE-EGO method, selecting three designs per iteration on both the 3D and 4D Rastrigin objective functions.

We assume we encounter the large variances on the three samples per iteration SIMPLE-EGO methods, because this method is not guaranteed to sample in the middle of the Pareto front. The higher the number of dimensions, the larger the design domain, and the larger the surface area of the edges. The standard deviation function tends to be highest along the edges and in the corners, until these regions have been been explored sufficiently. The SIMPLE-EGO method's selection of designs, which always includes the two extremes of the Pareto front, would imply that SIMPLE-EGO is adding one design at the current predicted objective function minimum, one design at the current maximum standard deviation, and one design at the maximum EI value. This maximum EI design is likely close to the current predicted objective function minimum. This implies that the SIMPLE-EGO method is probably exploiting a local basin, and exploring the edges.

In contrast to this, when adding ten designs per iteration the SIMPLE-EGO method selects the same three designs as the three designs per iteration method, but also adds 7

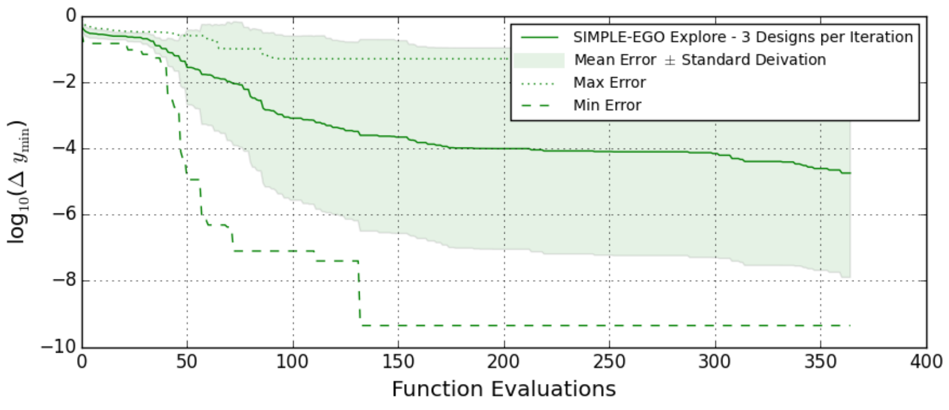


Figure 4.5.2: Convergence Plot for SIMPLE-EGO Exploit, ten samples per iteration, on the modified 3D Rastrigin objective Function

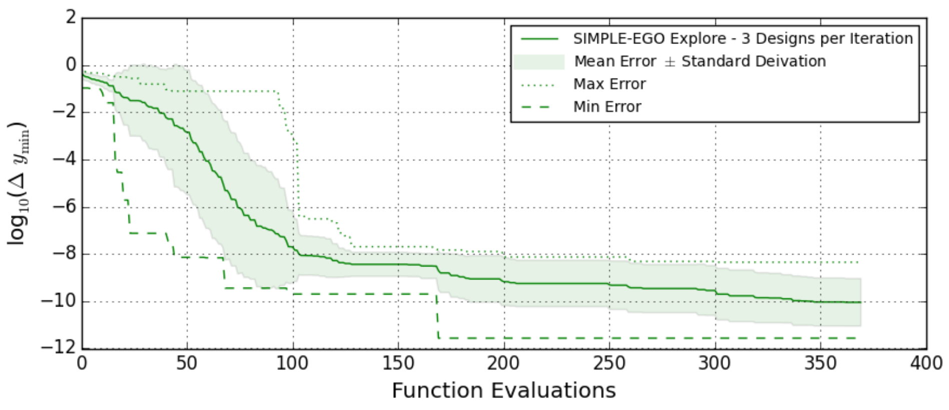


Figure 4.5.3: Convergence Plot for SIMPLE-EGO, three samples per iteration, on the modified 2D Rastrigin objective Function

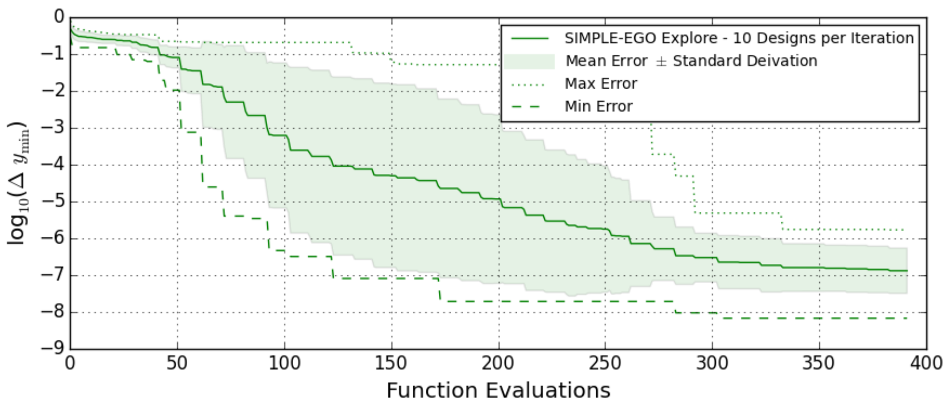


Figure 4.5.4: Convergence Plot for SIMPLE-EGO, ten samples per iteration, on the modified 3D Rastrigin objective Function

other designs from the Pareto front. These designs are likely to be distributed throughout the design domain, rather than biased towards the edges and corners. These designs assist SIMPLE-EGO, selecting 10 designs per iteration, to find the global basin in fewer function evaluations as compared to the three designs per iteration method. This behaviour is deduced by comparison of Figure 4.5.2 to Figure 4.5.4.

The 2D Rastrigin function has a smaller design domain, and relatively less area around the edges. The 2D problem only has 25 local minima, where the 3D and 4D problems have 125 and 625 local minima respectively. For these two reasons, in 2D the convergence problem is not as pronounced as in 3D and 4D. Figure 4.5.3 shows the convergence behaviour of SIMPLE-EGO, adding three designs per iteration, on the 2D modified Rastrigin function.

With the exception of the Rastrigin function, the other individual convergence plots are similar to Figure 4.5.1, which has small variance. The 63 individual convergence graphs are available in the electronic appendix (See Appendix B).

4.5.3 Combined Results

To compare the different methods with each other, we plot the average convergence error in Figures 4.5.5 to 4.5.22. The average plotted convergence error is defined as

$$\text{Average}\left(\log_{10}(\Delta y_{\min})\right) = \text{Average}\left[\log_{10}\left(\frac{y_{\min} - f_{\text{true min}}}{f_{\text{true max}} - f_{\text{true min}}}\right)\right]. \tag{4.5.2}$$

Classical EGO is represented by a solid gray line. The methods selecting three designs per iteration are plotted using a short dashed line, while the methods selecting ten designs per iteration are plotted using a longer dashed line.

For each objective function we present two plots. The first plot is the convergence error versus the iteration count. This plot serves as a measure of how the different methods performed on wall clock time. Wall clock time is the time it takes from when the method is started till when the method is finished. Therefore it is assumed that the computational device used has sufficient parallel capacity that it takes the same amount of time to perform one, three, or ten design evaluations.

The second plot shows convergence error versus the number of function evaluations. We would expect that there could be a negative consequence when running EGO in parallel. When running sequentially, EGO has the knowledge of all the previous designs before it makes the current selection. However, when running in parallel, the method also has the knowledge of all the previously sampled designs, but it does not have the knowledge about the results from the designs selected in the current iteration. Thus we expect that some of the selections made by the parallel methods will be naive. Parallelization should reduce wall clock time, but is likely to increase the total number of function evaluations. The convergence error versus number of function evaluation plots attempt to quantify this impact.

4.5.4 Discussion of Results

From the convergence error versus iteration count results, we can see that an increase in the number of samples per iteration, leads to a reduction in the number of iterations necessary to reach a certain accuracy. For a computational optimization problem this would imply parallelization leads to a reduction of wall clock time.

In contrast to parallelization leading to a reduction in wall clock time, in most cases parallelization leads to an increase in the number of function evaluations required to reach a certain accuracy. The implication of this is that parallelization causes an increase in total CPU time. The exception was the modified Rastrigin function in 3D and 4D. In these cases, EGO, which only samples one design vector per iteration, required more function evaluations to reach convergence than the SIMPLE-EGO methods. This is because Kriging underpredicts the standard deviation and EGO is known to be an exploitative method [25]. Because SIMPLE-EGO is more explorative than EGO, it is able to converge in less function evaluations than EGO on the modified Rastrigin function.

In Section 4.2 we mentioned that EGO-MO has similar behaviour to EGO. We mentioned that both methods are more geared toward exploitation than exploration. We can see this to be true by how fast EGO-MO initially converges on the less multi-modal Styblinsky-Tang and Rosenbrock functions, while EGO-MO has more difficulty converging on the highly multi-modal 3D and 4D modified Rastrigin functions.

Another prominent feature of EGO-MO is that it appears to have an initial exploit phase, where it converges quickly, followed by an explore phase where its convergence slows. This could be due to numerical aspects of EI reducing to zero as the iteration count increases. In Chapter 3 we investigated EI and discussed the challenges associated

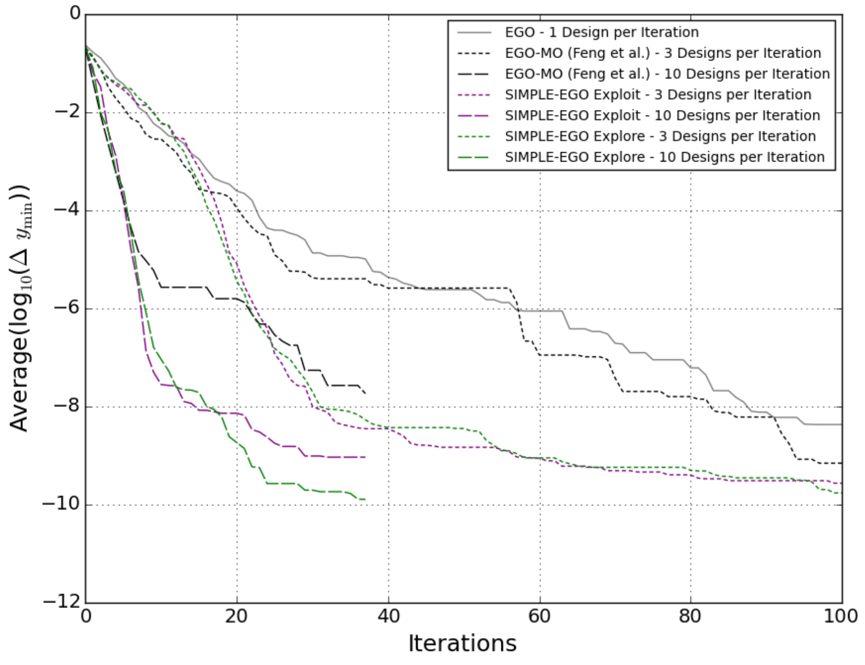


Figure 4.5.5: Convergence Error versus Iteration Count Converging on the 2D Modified Rastrigin Function

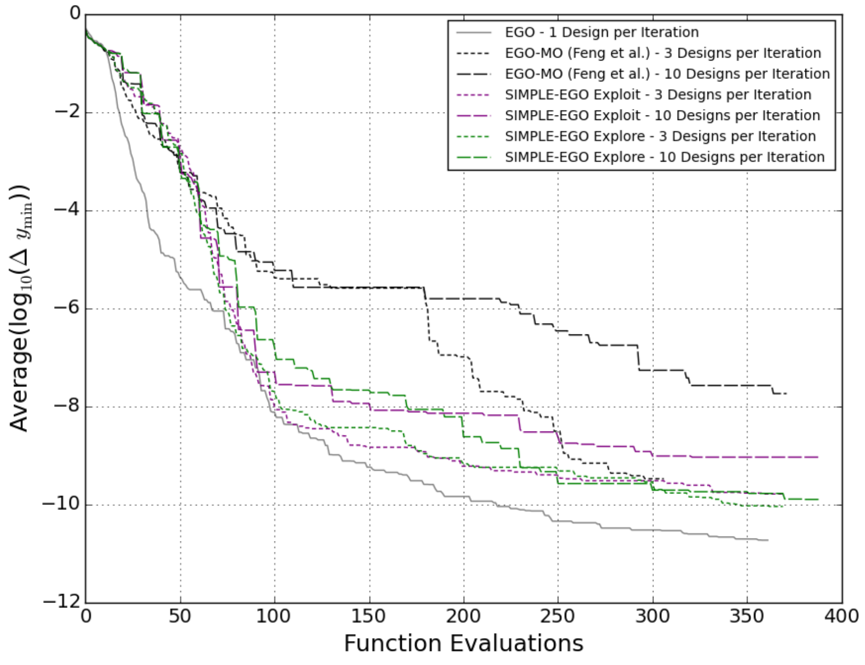


Figure 4.5.6: Convergence Error versus Number of Function Evaluations as Methods Converge on the 2D Modified Rastrigin Function

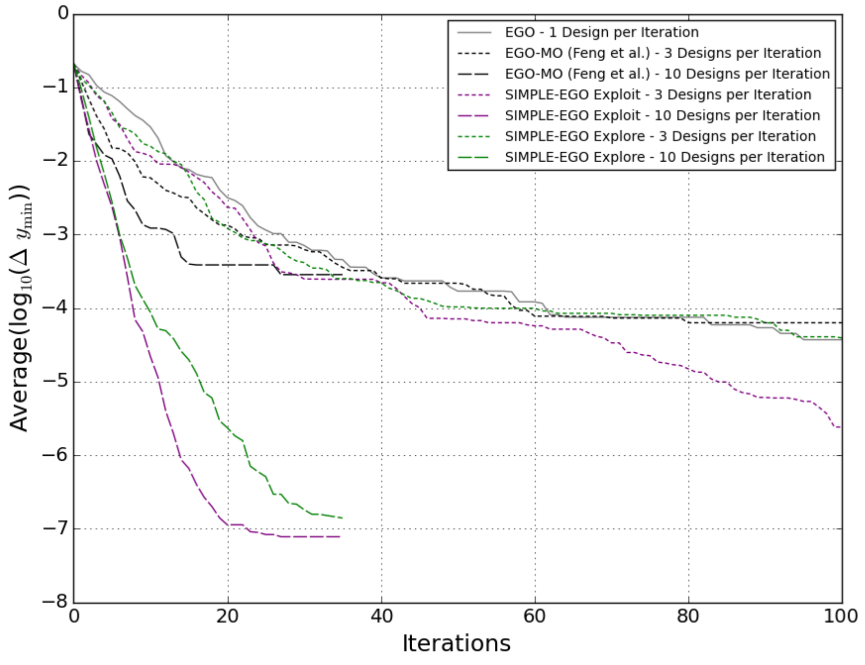


Figure 4.5.7: Convergence Error versus Iteration Count as Methods Converge on the 3D Modified Rastrigin Function

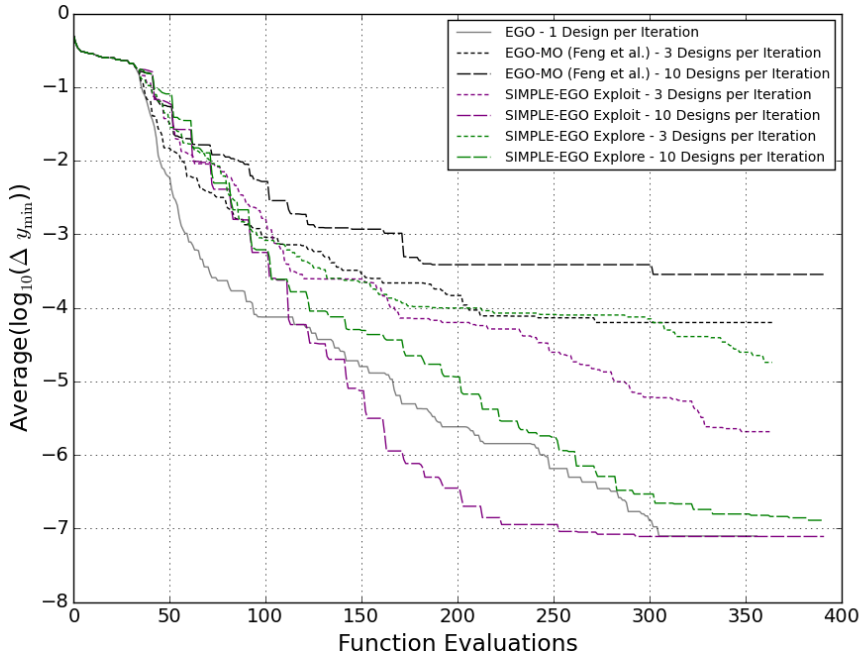


Figure 4.5.8: Convergence Error versus Number of Function Evaluations as Methods Converge on the 3D Modified Rastrigin Function

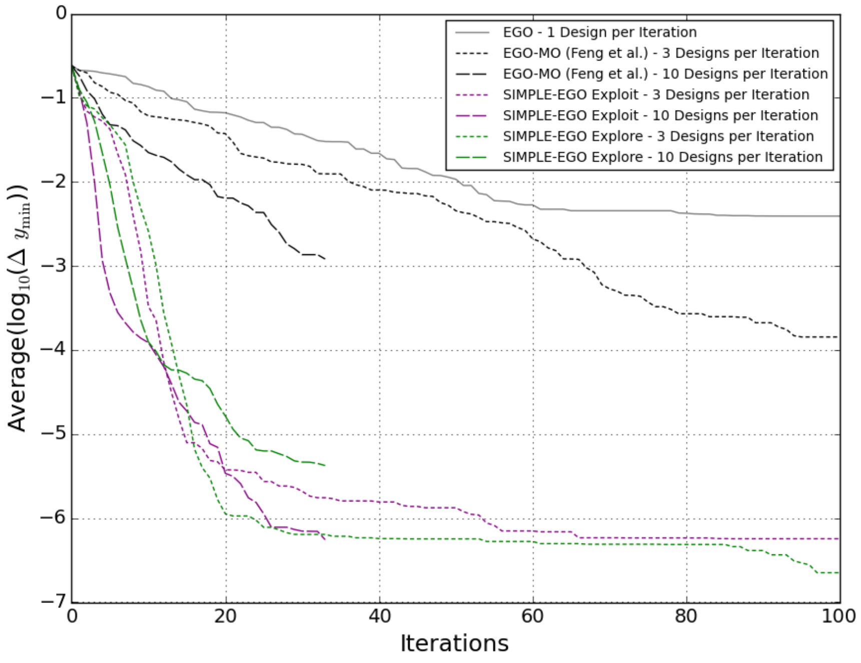


Figure 4.5.9: Convergence Error versus Iteration Count as Methods Converge on the 4D Modified Rastrigin Function

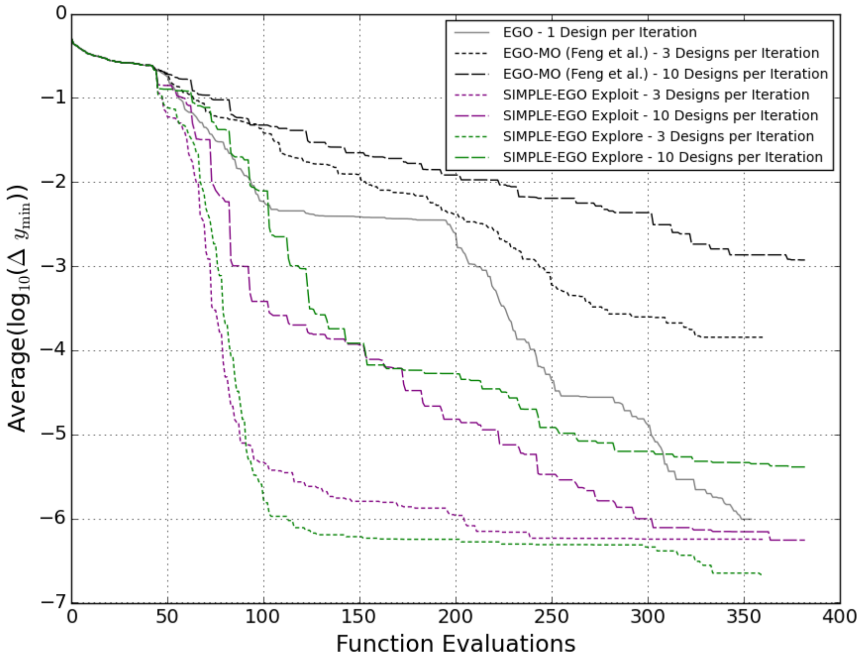


Figure 4.5.10: Convergence Error versus Number of Function Evaluations as Methods Converge on the 4D Modified Rastrigin Function

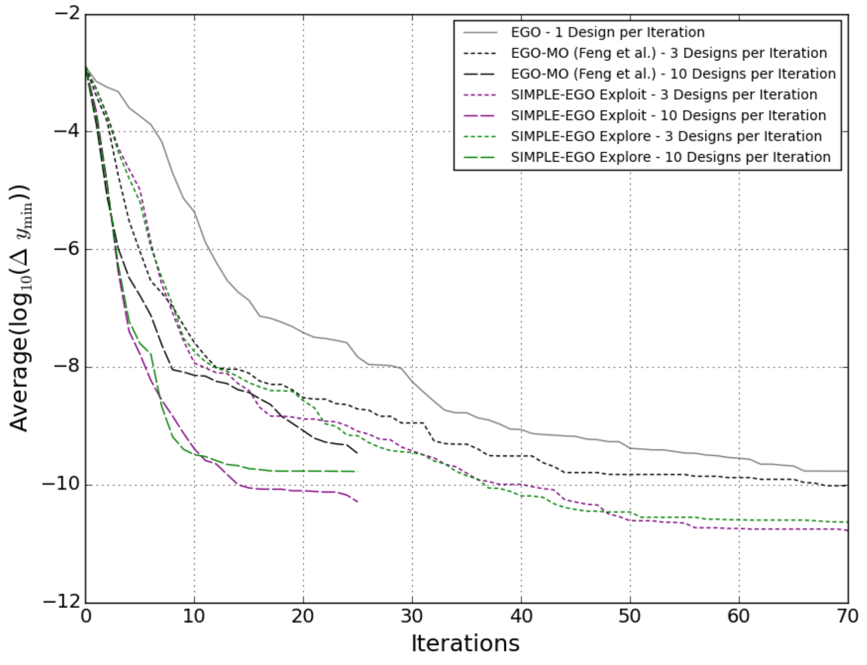


Figure 4.5.11: Convergence Error versus Iteration Count as Methods Converge on the 2D Rosenbrock Function

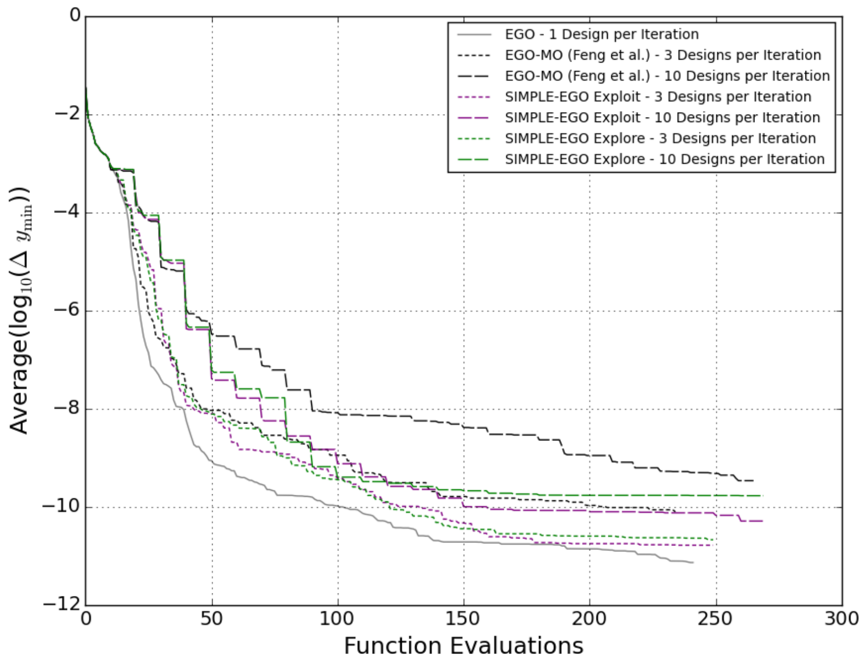


Figure 4.5.12: Convergence Error versus Number of Function Evaluations as Methods Converge on the 2D Rosenbrock Function

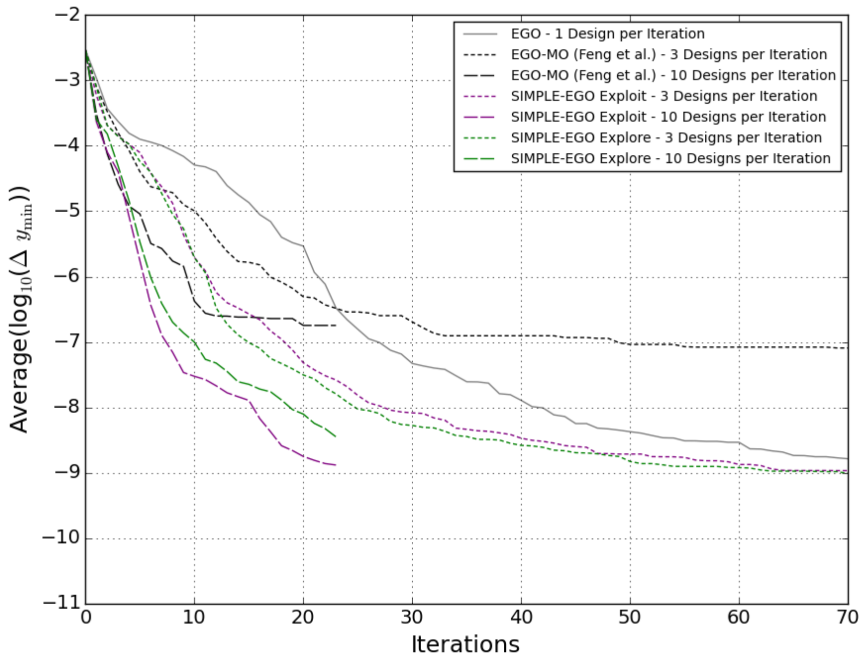


Figure 4.5.13: Convergence Error versus Iteration Count as Methods Converge on the 3D Rosenbrock Function

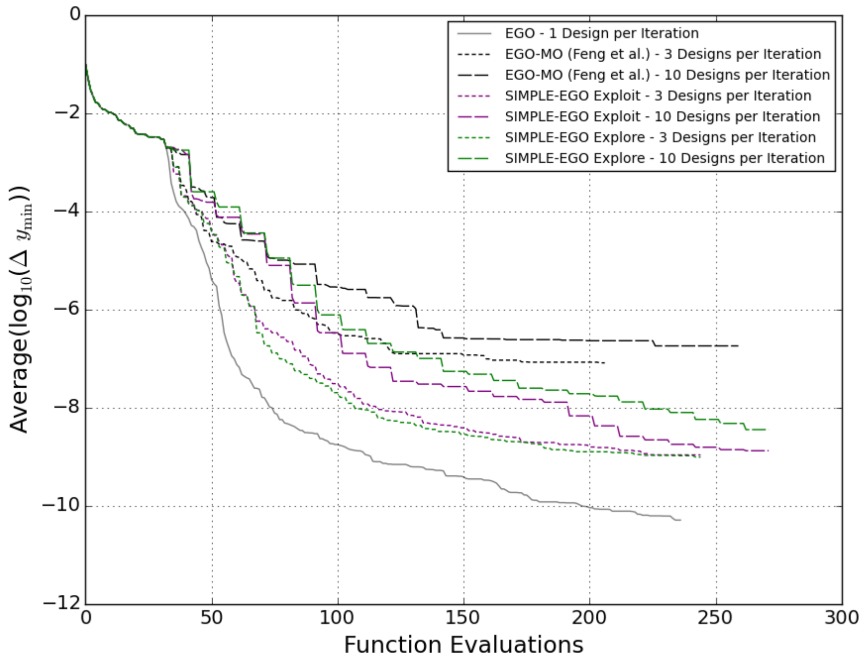


Figure 4.5.14: Convergence Error versus Number of Function Evaluations as Methods Converge on the 3D Rosenbrock Function

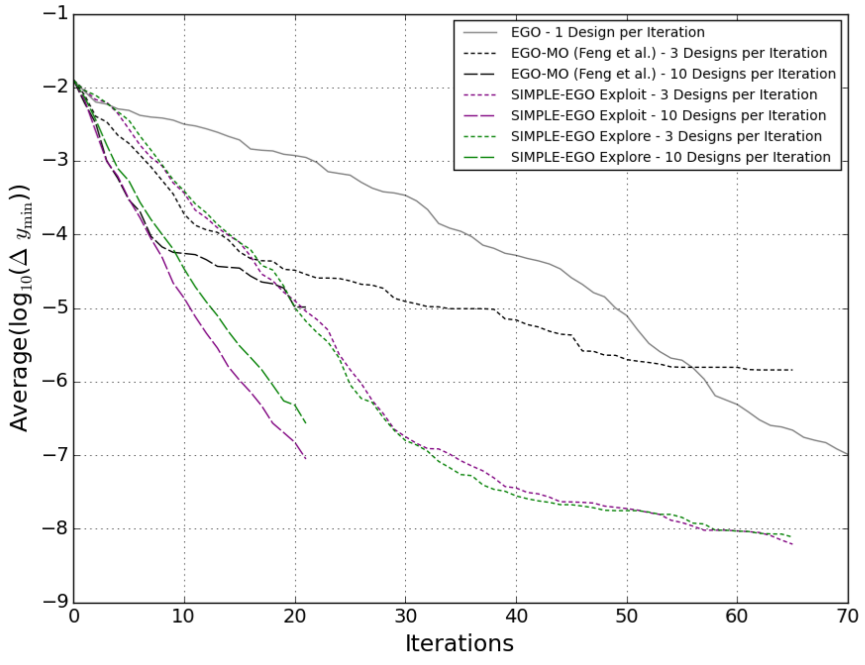


Figure 4.5.15: Convergence Error versus Iteration Count as Methods Converge on the 4D Rosenbrock Function

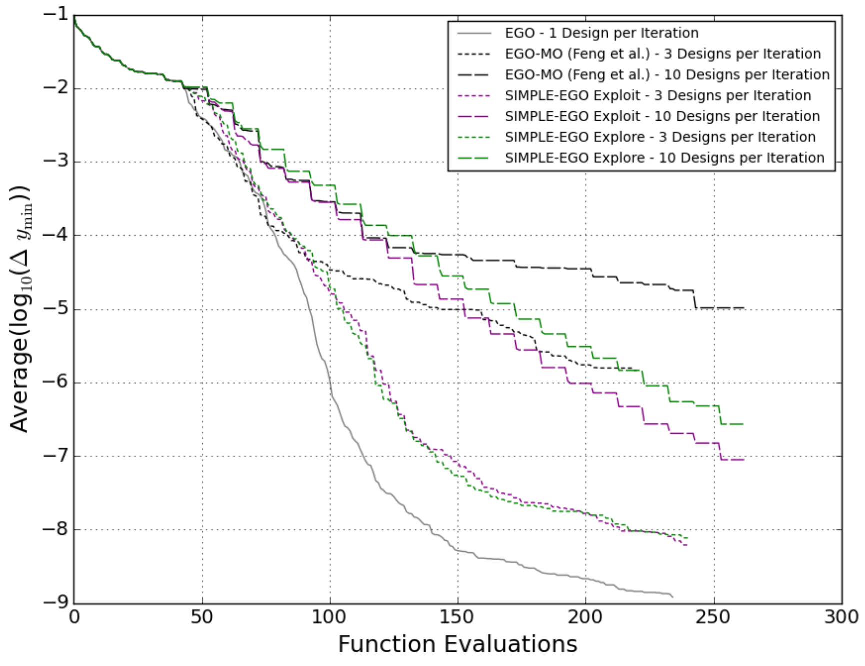


Figure 4.5.16: Convergence Error versus Number of Function Evaluations as Methods Converge on the 4D Rosenbrock Function

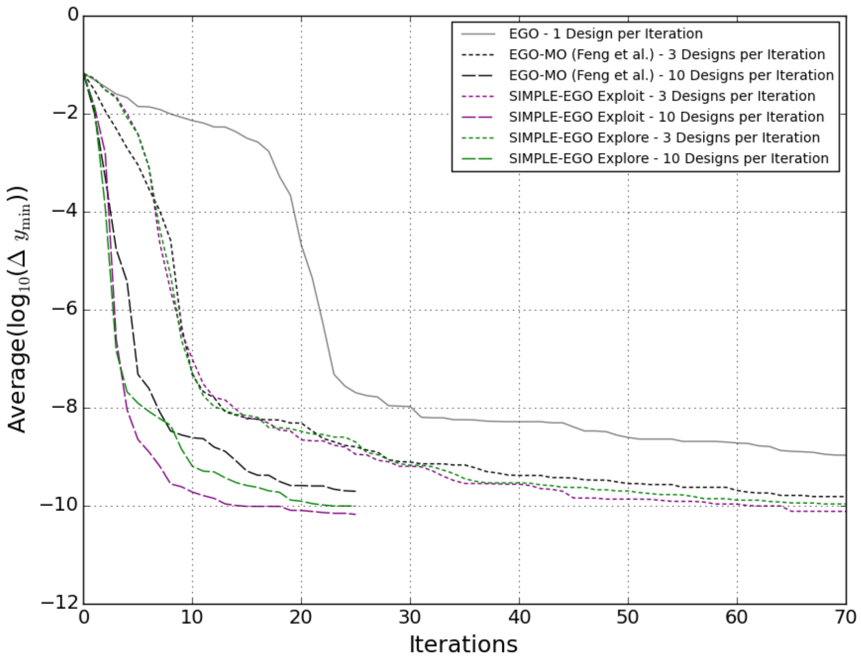


Figure 4.5.17: Convergence Error versus Iteration Count as Methods Converge on the 2D Styblinsky-Tang Function

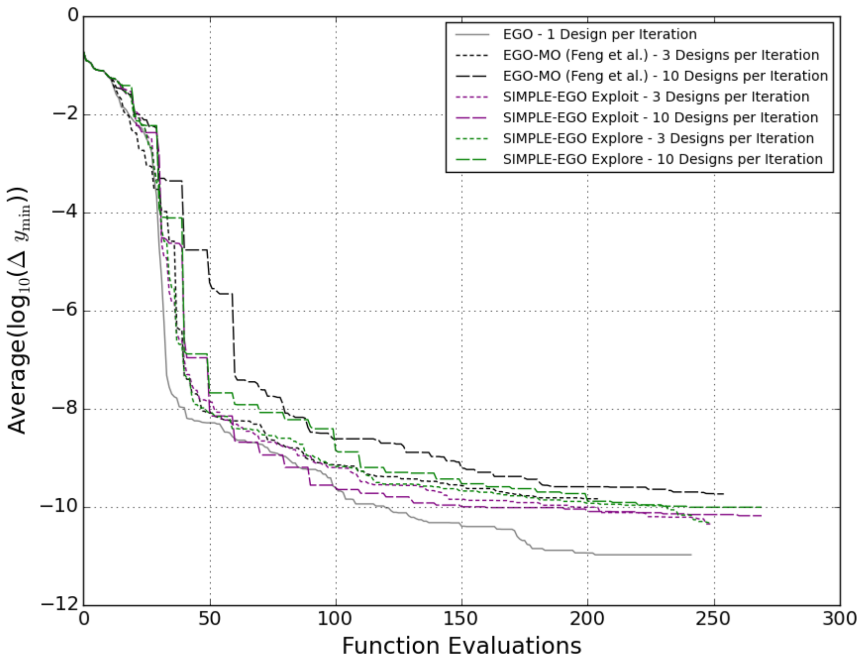


Figure 4.5.18: Convergence Error versus Number of Function Evaluations as Methods Converge on the 2D Styblinsky-Tang Function

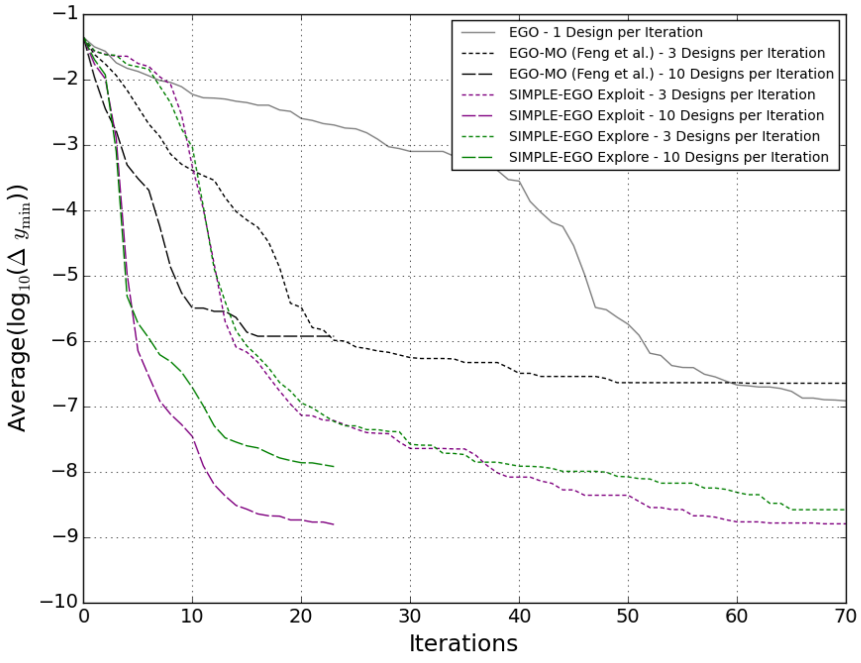


Figure 4.5.19: Convergence Error versus Iteration Count as Methods Converge on the 3D Styblinsky-Tang Function

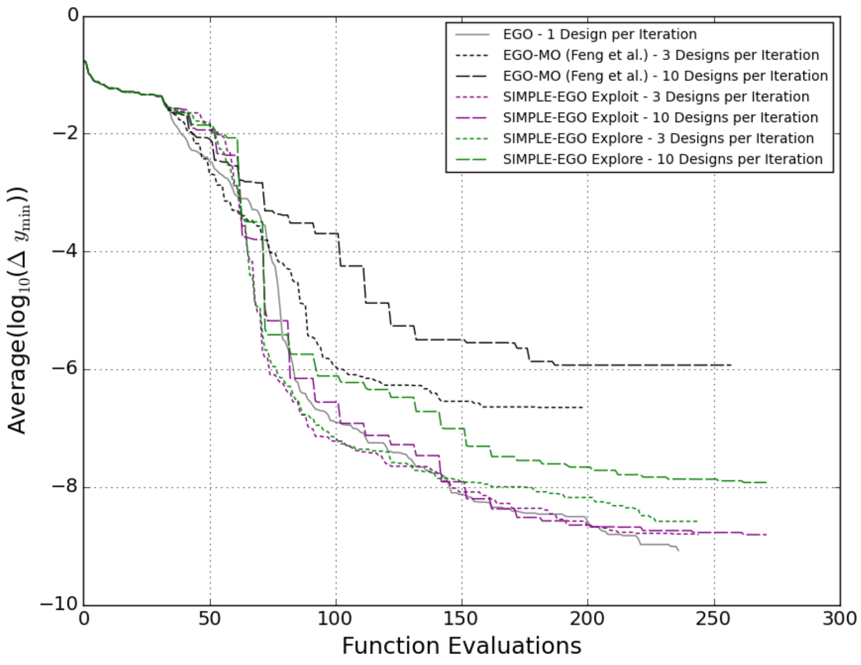


Figure 4.5.20: Convergence Error versus Number of Function Evaluations as Methods Converge on the 3D Styblinsky-Tang Function

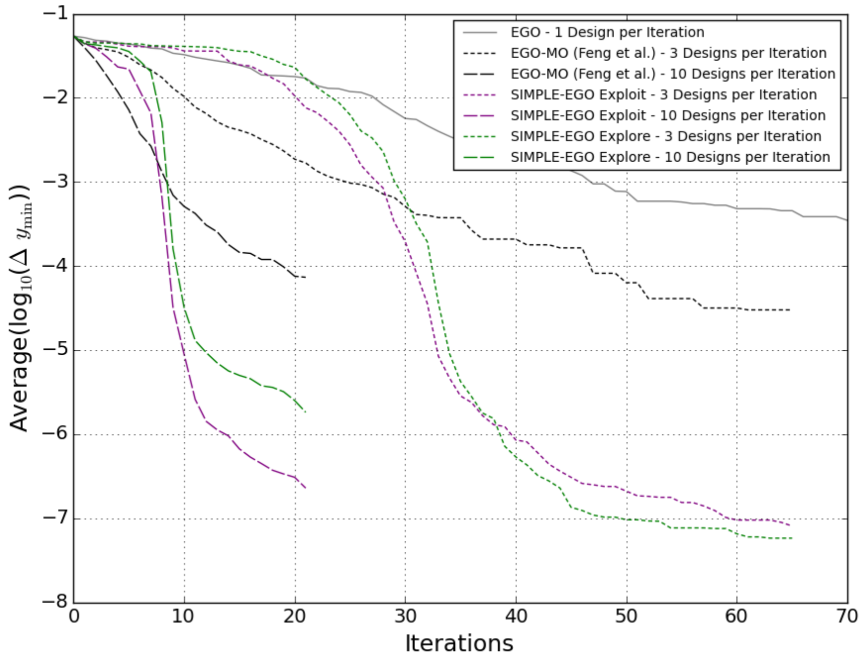


Figure 4.5.21: Convergence Error versus Iteration Count as Methods Converge on the 4D Styblinsky-Tang Function

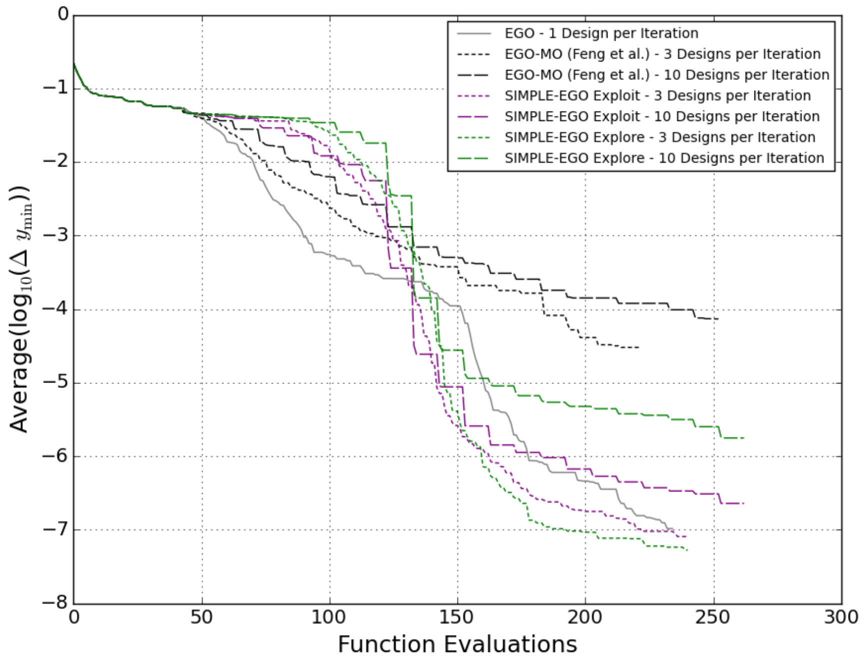


Figure 4.5.22: Convergence Error versus Number of Function Evaluations as Methods Converge on the 4D Styblinsky-Tang Function

with finding the maximum of EI. The two objectives used by EGO-MO tend towards zero as the optimization progresses and it becomes increasingly harder to find the Pareto front. We present a more in depth discussion of EGO-MO in Appendix D.

In reality, the SIMPLE-EGO Explore and SIMPLE-EGO Exploit methods are the same for three samples per iteration, because both of them are selecting the two Pareto front extremes, and the maximum EI design. Thus we would expect the same convergence characteristics. However, for SIMPLE-EGO that adds ten designs per iteration, we expect that the tuning parameter will have an effect. We notice the convergence characteristics for the explorative and the exploitative methods match closely throughout. Although the parameterization we have put forward does not seem to have a large impact on these highly multi-modal test problems, we do believe that design selection is important and future research opportunities include investigating different design vector selection methods.

To conclude, both the parallelization methods EGO-MO and SIMPLE-EGO decrease the number of iterations to reach a certain accuracy. Similar to EGO, EGO-MO is geared towards exploitatation, while the SIMPLE-EGO methods we have defined have more explorative characteristics. Therefore for highly multi-modal problems, SIMPLE-EGO should be preferred. If a user would like to use EGO-MO we would recommend also including the design vector which maximizes the EI function. Then at least one sensible new design is evaluated when the EGO-MO Pareto front becomes difficult to find. We expect such a strategy would perform well on unimodal problems.

Chapter 5

Conclusion

In this thesis we have investigated the characteristics and behaviour of EGO. A further focus of this research is the multi-objective parallelization of EGO.

In each iteration EGO fits a Kriging surface to the response of evaluated designs. Then, using the objective function value and the standard deviation (as measure of uncertainty) predicted by the Kriging surface, EGO selects a subsequent design. In each iteration EGO must decide if it will exploit or explore. The method EGO uses to make this decision is to maximize the EI infill sampling criterion.

Part of our investigation included how to find the maximum of EI. We were surprised at the difficulty of this optimization problem. We found that after a few iterations, EI has small basins which have higher EI values than some of the larger basins. These small basins are difficult to find using most optimization algorithms. In 2D, the basin covered approximately 0.01% of the design domain. However, we know that this basin must be located close to the minimum predicted objective function value. We thus propose a two-step heuristic to find the maximum EI value. We recommend using a stochastic optimization method localized around the predicted objective function minimum, as well as using a stochastic method in the rest of the domain. The results can be compared and the highest EI value selected.

In each iteration EI selects on design to sample. Since Jones et al. [25] suggested the EGO method in 1998, the objective of computational resources have decreased, while their capabilities have increased. We expected EGO would converge to the minimum faster if EGO selects more designs per iteration. We proposed a parallelization strategy called SIMPLE-EGO, which uses the multi-objective characteristics of EGO. In each iteration we find the Pareto optimal set of the objective function pair $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$. We then select designs from the Pareto optimal set to evalute in the subsequent iteration.

We compared our SIMPLE-EGO method to a method from literature, EGO-MO [16]. We found that both SIMPLE-EGO and EGO-MO reduce the number of iterations till convergence is reached. We noticed some detrimental behaviour of EGO-MO, such as difficulties solving the Pareto front.

In our three samples per iteration method, we noticed that the method for design selection we proposed may not yet be optimal, and there is scope for research comparing different methods of design vector selection. Specifically, we anticipate that selecting designs from the Pareto optimal set should use criteria defined in the design domain.

We have also identified another area for future research opportunity. In this study we did not exceed ten new designs per iteration. We suspect that if more designs are added per iteration, exploration clustering would occur, essentially wasting CPU time. A possible solution would be to use a Kriging believer strategy [44, 19] and SIMPLE-EGO together. Furthermore, in this study we limited the test problems to 4D. In future work we would also like to include a demonstration of SIMPLE-EGO solving higher dimensional problems, or real world engineering problems.

To conclude, EGO is an efficient optimization algorithm which aims to use few function evaluations to reach an optimal solution. We would recommend using multi-objective optimization as a parallelization strategy for EGO.

Bibliography

[1] J. S. Arora. *Introduction to Optimum Design*. Academic Press, 2012.

[2] A. Basudhar, C. Dribusch, S Lacaze, and S. Missoum. Constrained efficient global optimization with support vector machines. *Structural and Multidiscplinary Optimization*, 46:201–221, 2012.

[3] B. Betrò. *Operations Research '91: Extended Abstracts of the 16th Symposium on Operations Research held at the University of Trier at September 9–11, 1991*, chapter Bayesian Methods in Global Optimization, pages 16–18. Physica-Verlag HD, 1991.

[4] B. J. Bichon. *Efficient Surrogate Modeling for Reliability Analysis and Design*. PhD thesis, Vanderbilt University, 2010.

[5] F. Biscani, D. Izzo, and C.H. Yam. Parallel global multiobjective optimizer. Retrieved July 1, 2015, from `http://esa.github.io/pygmo/`.

[6] F. Boukouvala and M. G. Ierapetritou. Derivative-free optimization for expensive constrained problems using a novel expected improvement objective function. *AIChE Journal*, 60(7):2462–2474, 2014.

[7] M. Caramia and P. Dell'Olmo. *Multi-objective Management in Freight Logistics: Increasing Capacity, Service Level and Safety with Optimization Algorithms*. Springer, 2008.

[8] R. Carnell. Latin hybercube samples. Retrieved 5 February, 2015, from `https://cran.r-project.org/web/packages/lhs/lhs.pdf`, 2016.

[9] A. Chaudhuri and R. T. Haftka. A stopping criterion for surrogate based optimization using ego. In *10th World Congress on Structural and Multidisciplinary Optimization*.

[10] D. D. Cox and S. John. *Multidisciplinary Design Optimization: State of the Art*, chapter SDO: A statistical method for global optimization, pages 315–329. SIAM, 1997.

[11] N. A. C. Cressie. The origins of kriging. *Mathematical Geology*, 22(3):239–252, 1990.

[12] N. A. C. Cressie. *Statistics for Spatial Data.* John Wiley and Sons, 1991.

[13] K. Deb, A. Pratap, S. Agarwal, and T. Maeyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[14] J. F. Elder IV. Global R^d optimization when probes are expensive: the GROPE algorithm. In *Proceedings of the 1992 IEEE International Conference on Systems, Man, and Cybernetics*, 1992.

[15] Z. Feng, T. Yang, J. Ge, Q. Tang, and Y. Ma. Efficient aerodynamic optimization using a multiobjective optimization based framework to balance the exploration and exploitation. In *11th World Congress on Structural and Multidisciplinary Optimisation*, 2015.

[16] Z. Feng, Q. Zhang, Q. Zhang, Q. Tang, T. Yang, and Y. Ma. A multiobjective optimization based framework to balance the global exploration and local exploitation in expensive optimization. *Journal of Global Optimization*, 61:677–694, 2015.

[17] L. Gautier. rpy2. Retrieved 5 February 2015, from `http://rpy2.readthedocs.io/en/version_2.7.x/`, 2016.

[18] D. Ginsbourger, R. Riche, and L. Carraro. A multi-points criterion for deterministic parallel global optimization based on gaussian processes. *HAL*, 2008.

[19] D. Ginsbourger, R. Riche, and L. Carraro. Kriging is well-suited to parallelize optimization. In Y. Tenne and Goh C., editors, *Computational Intelligence in Expensive Optimization Problems*, volume 2. Springer, 2010.

[20] T. Hengl. *A Practical Guide to Geostatistical Mapping.* University of Amsterdam, 2009.

[21] N. Henkenjohann and J. Kunert. An efficient sequential optimization approach based on the multivariate expected improvement criterion. *Quality Engineering*, 19:267–280, 2007.

[22] T. Ishikawa and M. Matsunami. An optimization method based on radial basis functions. *IEEE Transactions on Magnetics*, 33(2), 1997.

[23] T. Ishikawa, Y. Tsukui, and M. Matsunami. A combined method for the global optimization using radial basis function and deterministic approach. *IEEE Transactions on Magnetics*, 35(3):1730–1733, 1999.

[24] D. R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383, 2001.

[25] D. R. Jones, M. Schonlau, and W. J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[26] E. Jones, T. Oliphant, Peterson. P., et al. SciPy: Open source scientific tools for Python. Online, 2001.

[27] S. Kok and C. Sandrock. Locating and characterizing the stationary points of the extended Rosenbrock function. *Evolutionary computation*, 17(3):437–453, 2009.

[28] H. J. Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.

[29] D. J. Lizotte. *Practical Bayesian Optimization*. PhD thesis, University of Alberta, 2008.

[30] J. D. Martin and T. W. Simpson. Use of kriging models to approximate deterministic computer models. *AIAA Journal*, 43(4):853–863, 2005.

[31] G. Matheron. Kriging or polynomial interpolation procedures. *CIMM Transactions*, 70:240–244, 1967.

[32] M. D. Mckay. Latin hypercube sampling as a tool in uncertainty analysis of computer models. In *Proceeding WSC '92 Proceedings of the 24th conference on Winter simulation*, pages 557–564, 1992.

[33] J. Mockus. Application of bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4(4):347–365, 1994.

[34] J. Mockus, V. Tiesis, and A. Zilinskas. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2, 1978.

[35] J. Muller. User guide for modularized surrogate model toolbox. *Tampere University of Technology Department of Mathematics*, 2012.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[37] V. Picheny, T. Wagner, and D. Ginsbourger. A benchmark of kriging-based infill criteria for noisy optimization. *Structural and Multidisciplinary Optimization*, 48(3):607–626, 2013.

[38] W. Ponweiser, T. Wagner, and M. Vincze. Clustered multiple generalized expected improvement: A novel infill sampling criterion for surrogate models. In *IEEE Congress on Evolutionary Computation*, 2008.

[39] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2014.

[40] C. E. Rasmussen and K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[41] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4):409–435, 1989.

[42] T. J. Santner, B. J. Williams, and W. I. Notz. *The Design and Analysis of Computer Experiments*. Springer Series in Statistics. Springer, 2003.

[43] M. J. Sasena. *Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations*. PhD thesis, University of Michigan, 2002.

[44] M. Schonlau. *Computer Experiments and Global Optimization*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1997.

[45] M. Schonlau, W. J. Welch, and D. R. Jones. Global versus local search in constrained optimization of computer models. Technical Report 83, National Institute of Statistical Sciences, 19 T. W. Alexander Drive PO Box 14006 Research Triangle Park, NC 27709-4006, March 1998.

[46] T. W. Simpson, J. J. Korte, T. M. Mauery, and F. Mistree. Comparison of response surface and kriging models for multidisciplinary design optimization. In *Proceedings of the 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 1998.

[47] A. Sóbester, S. J. Leary, and A. J. Keane. A parallel updating scheme for approximating and optimizing high fidelity computer simulations. *Structural Multidisciplinary Optimization*, 27:371–383, 2004.

[48] A. Sóbester, S. J. Leary, and A. J. Keane. On the design of optimization strategies based on global response surface approximation models. *Journal of Global Optimization*, 33:31–59, 2005.

[49] INFORMS Computing Society. The nature of mathematical programming - mathematical programming glossary. Retrieved 6 May 2016, from http://glossary.computing.society.informs.org/index.php?page=nature.html, 2014.

[50] M. L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Series in Statistics. Springer, 1999.

[51] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved May 6, 2016, from `http://www.sfu.ca/~ssurjano/hart3.html`, 2013.

[52] A. Torn and A. Žilinskas. *Global optimization*. Springer-Verlag New York, Inc., 1989.

[53] F. A. C. Viana and R. T. Haftka. Importing uncertainty estimates from one surrogate to another. In *50th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2009.

[54] F. A. C. Viana, R. T. Haftka, and V. Steffen. Multiple surrogates: how cross-validation errors can help us to obtain the best predictor. *Structural Multidisciplinary Optimization*, 39:439–457, 2009.

[55] F. A. C. Viana, R. T. Haftka, and L.T. Watson. Why not run the efficient global optimization algorithm with multiple surrogates? In *51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2010.

[56] F. A. C. Viana, R. T. Haftka, and L.T. Watson. Efficient global optimization algorithm assisted by multiple surrogate techniques. *Journal of Global Optimization*, 56:669–689, 2013.

[57] A. G. Watson and R. J. Barnes. Infill sampling criteria to locate extremes. *Mathematical Geology*, 27(5):589–608, 1995.

[58] A. Žilinskas. A review of statistical models for global optimization. *Journal of Global Optimization*, 2(2):145–153, 1992.

Appendix A

Derivation of Expected Improvement

Here follows a derivation of Expected Improvement (EI), as presented in Bichon [4]. EI is defined as

$$\mathrm{EI}(\boldsymbol{x}) = \mathrm{E}[\max(y_{\min} - y(\boldsymbol{x}), 0)], \tag{A.0.1}$$

which can be split up into:

$$\max(y_{\min} - y) = y_{\min} - y \qquad \text{for } -\infty < y < y_{\min}, \tag{A.0.2}$$

$$= 0 \qquad \text{for } y_{\min} < y < \infty. \tag{A.0.3}$$

For the remainder of this derivation we omit the dependance on $\boldsymbol{x}$ because we perform the derivation as if we are at a single design space location.

The expected value, E, can be computed by integrating over the interval where the function is nonzero

$$\mathrm{EI} = \int_{\infty}^{y_{\min}} (y_{\min} - y)Y(y)dy \tag{A.0.4}$$

$$= \int_{\infty}^{y_{\min}} y_{\min}Y(y)dy - \int_{\infty}^{y_{\min}} yY(y)dy \tag{A.0.5}$$

$$= y_{\min}\Phi\left(\frac{y_{\min} - \mu}{\sigma}\right) - \int_{\infty}^{y_{\min}} yY(y)dy. \tag{A.0.6}$$

Taking the second term of Equation (A.0.6) we can show

$$\int_{-\infty}^{y_{\min}} yY(y)dy = \int_{-\infty}^{y_{\min}} \frac{y}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2\right] dy. \tag{A.0.7}$$

Use the variable transformation $v = \frac{y-\mu}{\sigma}$ which gives $dy = \sigma dv$. Substituting these gives

$$\int_{-\infty}^{y_{\min}} yY(y)dy = \int_{-\infty}^{\nu_{min}} (v\sigma + \mu)\frac{y}{\sqrt{2\pi}} \exp\left[\frac{-\nu^2}{2}\right] dv \tag{A.0.8}$$

where $v_{\min} = \frac{y_{\min}-v}{\sigma}$. The equation can be broken up into two parts, giving

$$\int_{-\infty}^{y_{\min}} yY(y)dy = \frac{\sigma}{\sqrt{2\pi}} \int_{-\infty}^{v_{\min}} v \exp\left[-\frac{v^2}{2}\right] dv + \mu \int_{-\infty}^{v_{\min}} \frac{1}{\sqrt{2\pi}} \exp\left[-\frac{v^2}{2}\right] dv. \tag{A.0.9}$$

The second term is the integral of the standard normal PDF, which can be expressed using the CDF, giving:

$$\int_{-\infty}^{y_{\min}} yY(y)dy = \frac{\sigma}{\sqrt{2\pi}} \int_{-\infty}^{v_{\min}} v \exp\left[-\frac{v^2}{2}\right] dv + \mu\Phi(v_{\min}). \tag{A.0.10}$$

For the first term in the equation above, we introduce the change of variables $w_c = \frac{v^2}{2}$, which gives $dv = \frac{1}{v}dw_c$. This reduces the integral to:

$$\int_{-\infty}^{v_{\min}} v \exp\left[-\frac{v^2}{2}\right] dv = \int_{\infty}^{w_{\min}} \exp(-w_c)dw_c, \tag{A.0.11}$$

where $w_{\min} = \frac{1}{2}\left(\frac{y_{\min}-\mu}{\sigma}\right)^2$. The lower bound has been changed from $-\infty$ to ∞ because the transformation involves a square. This gives:

$$\int_{-\infty}^{v_{\min}} v \exp\left[-\frac{v^2}{2}\right] dv = -\exp[-w_c]|_{\infty}^{w_{\min}} = -\exp[-w_{\min}]. \tag{A.0.12}$$

We back substitute

$$\int_{-\infty}^{y_{\min}} yY(y)dy = -\frac{\sigma}{\sqrt{2\pi}} \exp[-w_{\min}] + \mu\Phi(v_{\min}) \tag{A.0.13}$$

which can be written in terms of the standard normal PDF as

$$\int_{-\infty}^{y_{\min}} yY(y)dy = -\sigma\phi(v_{\min}) + \mu\Phi(v_{\min}) \tag{A.0.14}$$

$$= -\sigma\phi\left(\frac{y_{\min}-\mu}{\sigma}\right) + \mu\Phi\left(\frac{y_{\min}-\mu}{\sigma}\right). \tag{A.0.15}$$

We finally back substitute

$$\mathrm{EI} = y_{\min}\Phi\left(\frac{y_{\min}-\mu}{\sigma}\right) + \sigma\phi\left(\frac{y_{\min}-\mu}{\sigma}\right) - \mu\Phi\left(\frac{y_{\min}-\mu}{\sigma}\right), \tag{A.0.16}$$

which can be rearranged to give

$$\mathrm{EI} = (y_{\min}-\mu)\Phi\left(\frac{y_{\min}-\mu}{\sigma}\right) + \sigma\phi\left(\frac{y_{\min}-\mu}{\sigma}\right). \tag{A.0.17}$$

Appendix B

Electronic Results

B.1 Introduction

Throughout this document we refer to visual results available in an electronic appendix. In this Section we present details to what is available in the electronic appendix, and how it can be accessed.

The electronic appendix is available online on the University of Pretoria Library catalog.

B.2 Sequential EGO - Visual Results

For EGO we include the visual results for EGO progressing on the following test functions:

1. Modified Rastrigin
2. Rosenbrock
3. Styblinsky-Tang

B.3 SIMPLE-EGO - Visual Results

We ran three variations of SIMPLE-EGO. The following results are presented online:

1. SIMPLE-EGO Exploit

   - (a) Modified Rastrigin
     - i. 3 Designs per iteration
     - ii. 10 Designs per iteration
   - (b) Rosenbrock
     - i. 3 Designs per iteration
     - ii. 10 Designs per iteration
   - (c) Styblinsky-Tang
     - i. 3 Designs per iteration
     - ii. 10 Designs per iteration

2. SIMPLE-EGO Explore
   - (a) Modified Rastrigin
     - i. 3 Designs per iteration
     - ii. 10 Designs per iteration
   - (b) Rosenbrock
     - i. 3 Designs per iteration
     - ii. 10 Designs per iteration
   - (c) Styblinsky-Tang
     - i. 3 Designs per iteration
     - ii. 10 Designs per iteration

3. SIMPLE-EGO Unwise Explore
   - (a) Modified Rastrigin
     - i. 3 Designs per iteration
     - ii. 10 Designs per iteration
   - (b) Rosenbrock
     - i. 3 Designs per iteration
     - ii. 10 Designs per iteration
   - (c) Styblinsky-Tang
     - i. 3 Designs per iteration
     - ii. 10 Designs per iteration

B.4 EGO-MO - Visual Results

We present the following visual results for EGO-MO

1. Modified Rastrigin
   (a) 3 Designs per iteration
   (b) 10 Designs per iteration
2. Rosenbrock
   (a) 3 Designs per iteration
   (b) 10 Designs per iteration
3. Styblinsky-Tang
   (a) 3 Designs per iteration
   (b) 10 Designs per iteration

B.5 Convergence Runs - Individual Convergence

We present two sets of PDF files containing the individual convergence graphs. The one PDF file contains the convergence versus the function evaluations, while the second contains the convergence versus the number of iterations.

Appendix C

Computational Implementation

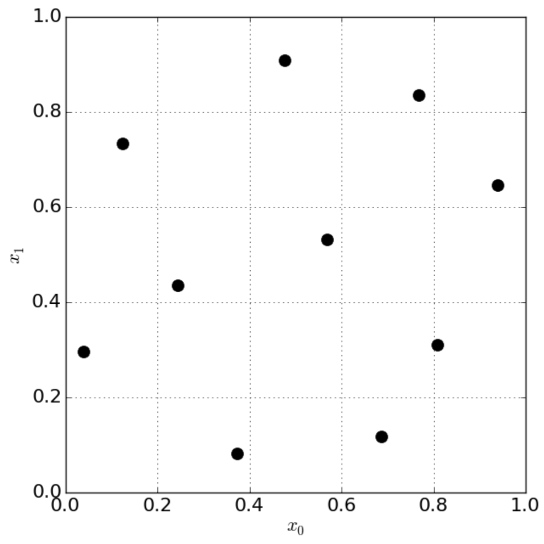
C.1 Introduction

In this appendix we present the details of our computational implementation, so that any reader may independently reproduce these results.

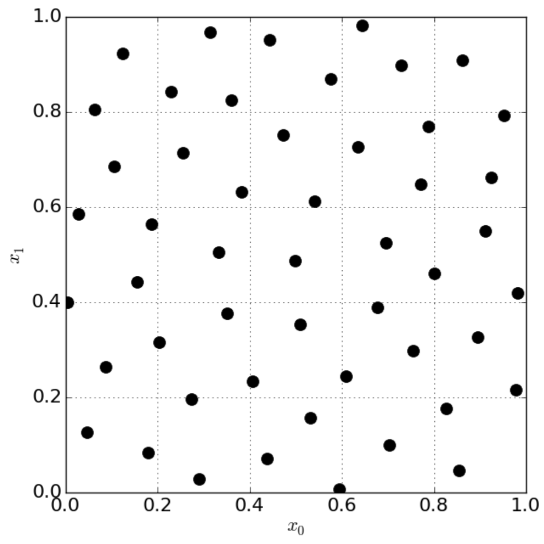
We have programmed these methods using Python. Python is user friendly and many pre-packaged methods are available. We have aimed to, where possible, use well established packages.

C.2 Latin Hypercube Sampling - R

To generate the initial Latin Hypercube Sampling (LHS) design of experiments (DOE), we used an implementation in R [39]. R has a LHS implementation in the `lhs` [8] package. We experimented with the different methods and settings in this package. We decided to use the `optimumLHS` method. We set the `maxSweeps` parameter equal to the number of initial function evaluations. These settings gave us designs as presented in Figure C.2.1. We coupled R to Python using RPy2 [17].



(a) 10 Samples



(b) 50 Samples

Figure C.2.1: Latin Hypercube Design of Experiments

C.3 Gaussian Process - Sklearn

Sklearn, (also referred to as Scikit-Learn) [36] is the method we used to fit the Gaussian Process as presented in Chapter 2. Initially we used our own implementation in Python, which follows exactly the methodology described in Chapter 2. However, for better computational efficiency, we decided to switch to the Sklearn implementation.

We used the following settings:

- Regression: Constant
- Correlation Function: Squared Exponential
- Nugget: 1e-12
- Theta$_{\text{LB}}$: 0.1
- Theta$_{\text{UB}}$: 100
- Theta Starting Value: 10
- Normalize: False

This method uses MLE to fit the Gaussian Process. Any theta between 0.1 and 100 can be selected. We found these settings to best represent our own implemented method.

C.4 Differential Evolution - Scipy

Because EI is highly multi-modal we decided to use a Stochastic Optimizer to find the global maximum of the function. We used the Differential Evolution (DE) implementation in `scipy.optimize` [26]. We used the following settings:

- popsize: 50 (which is interpreted by the DE as $50n$ individuals, where n is the number of dimensions)
- Tolerance: 0.01
- Polish: True (Default)

C.5 Multi-Objective Optimizer - PyGMO

As there is no built in multi-objective optimization algorithm in Python we used a package called PyGMO [5]. We found our Pareto front using a Non-dominated Sorting Genetic Algorithm (NSGA-II) [13]. Our population size was 200 and we ran it for 150 generations, using the default settings. We found that these settings could repeatably find the Pareto front.

C.6 k-means Clustering - Sklearn

EGO-MO requires the use of a clustering algorithm. We used the k-means clustering available in the Sklearn toolbox (also referred to as Scikit-Learn) [36]. We select the number of clusters we require. We use the default settings.

Appendix D

EGO-MO Discussions

D.1 Introduction

Feng et al. [16] proposed a method in 2015, named EGO-Multi-Objective (EGO-MO). EGO-MO uses the exploitation exploration characteristics of the two terms of the Expected Improvement (EI) function. They find the Pareto optimal set by treating the two EI terms as a multi-objective optimization problem. In this Appendix we present an overview of the method. We also present a visual study, discussing the behaviour of EGO-MO.

D.2 Overview of EGO-MO

Figure D.2.1 gives a summary of the EGO-MO algorithm. EGO-MO uses a Multi-Objective Evolutionary Algorithm to find a Pareto front for the following optimization problem:

$$\text{Minimize} f_{\text{EI1}} \text{and} f_{\text{EI2}}: \quad f_{\text{EI1}}(\boldsymbol{x}) = -(y_{\min} - y(\boldsymbol{x}))\Phi\left(\frac{y_{\min} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right), \tag{D.2.1}$$

$$f_{\text{EI2}}(\boldsymbol{x}) = -\sigma(\boldsymbol{x})\phi\left(\frac{y_{\min} - y(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right). \tag{D.2.2}$$

Thereafter, certain designs on the Pareto front are selected. They discard any design vector on the Pareto front which are closer than an Euclidean distance to previously sampled design vectors. Thereafter, they select the Pareto optimal designs to sample.

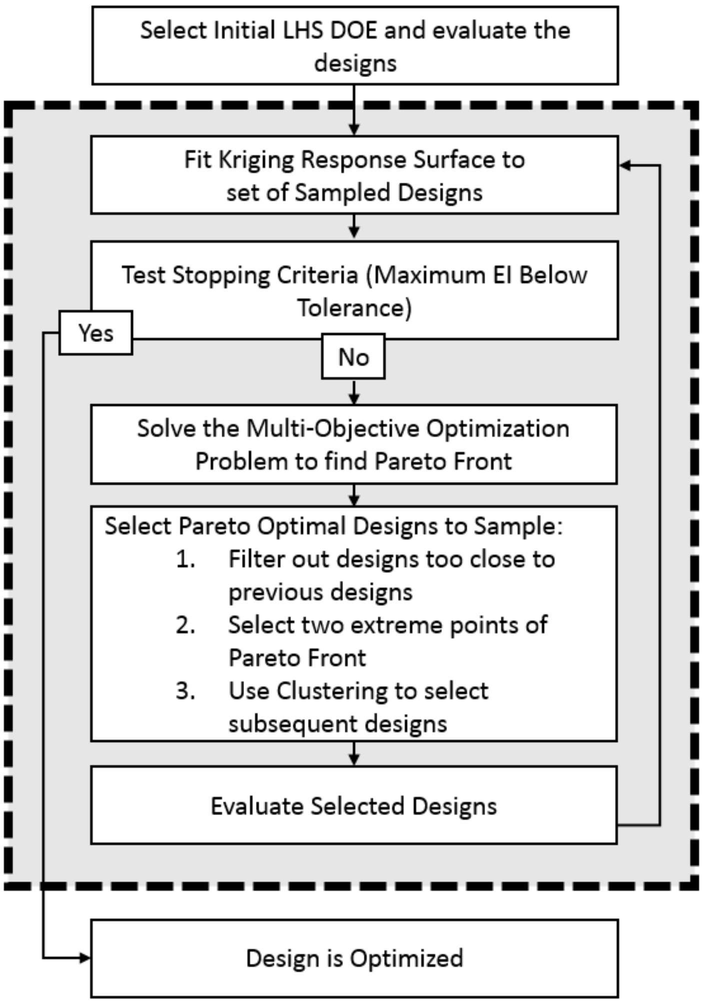

Select Initial LHS DOE and evaluate the designs
Fit Kriging Response Surface to set of Sampled Designs
Test Stopping Criteria (Maximum EI Below Tolerance)
Yes
No
Solve the Multi-Objective Optimization Problem to find Pareto Front
Select Pareto Optimal Designs to Sample:
1. Filter out designs too close to previous designs
2. Select two extreme points of Pareto Front
3. Use Clustering to select subsequent designs
Evaluate Selected Designs
Design is Optimized

Figure D.2.1: EGO-MO Algorithm

They select the two Pareto front edge points. To select the remaining design vectors they used c-means clustering. Once the Pareto optimal design vectors have been clustered they select a design from each cluster which maximizes the Euclidean distance from the other design vectors in the domain. We followed this same procedure, however instead of c-means clustering, we use k-means clustering as implemented by the Sklearn toolbox [36].

D.3 Visual Investigation

D.3.1 Define plots

To investigate and understand the behaviour of EGO-MO we followed a visual study as discussed in Chapter 3. In Chapter 3 we plotted a Pareto front using $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$. For EGO-MO, the Pareto front is determined by using f_{EI1} and f_{EI2}. We name this Pareto front the EI-Pareto front, while we name the Pareto front using the predicted function value $f(\boldsymbol{x})$ and the uncertainty $\sigma(\boldsymbol{x})$ the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front.

The EI criterion space plot is shown in Figure D.3.1. The colors selected in the legend for EI are green and yellow, and thus for effortless recognition, we plot the EI-Pareto front in green. The selected designs are indicated as white and gray circles, while the maximum EI design is indicated with a green triangle. We solved the EI-Pareto front, and the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front. Figure D.3.1 (a) shows the EI-Pareto front, plotted in the EI criterion space, while Figure D.3.1 (b) indicates the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front designs. We consider it important to understand the mapping which exists between the two criterion space plots. When we map the designs between the different criterion spaces, and the design space, we maintain the same coloring and symbols.

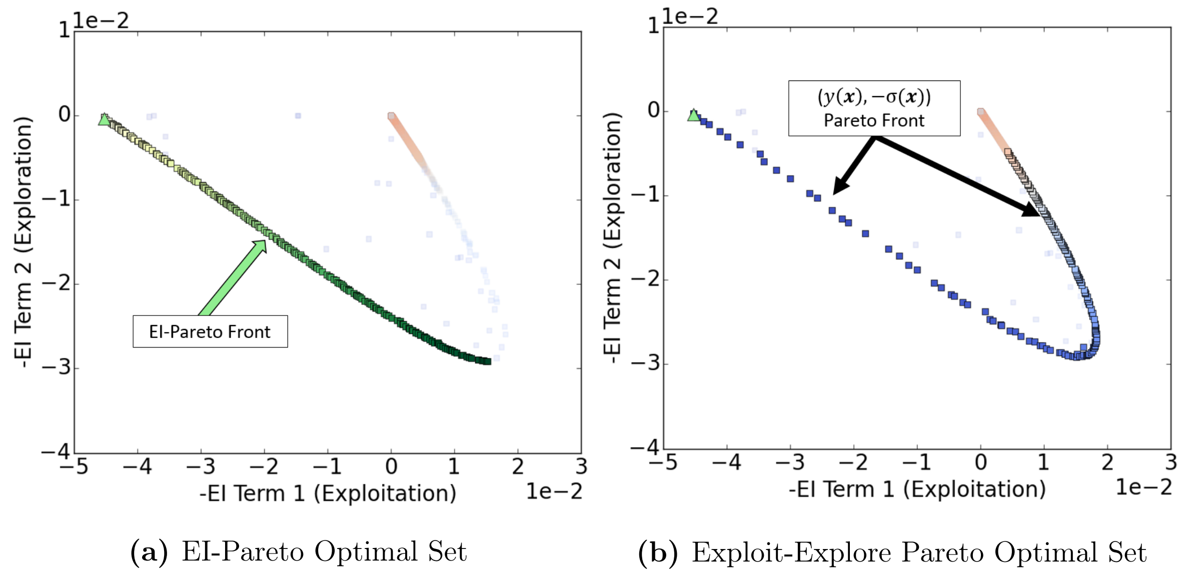
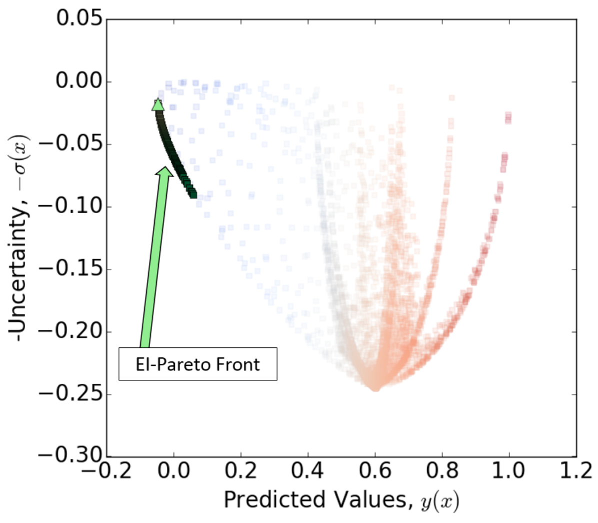
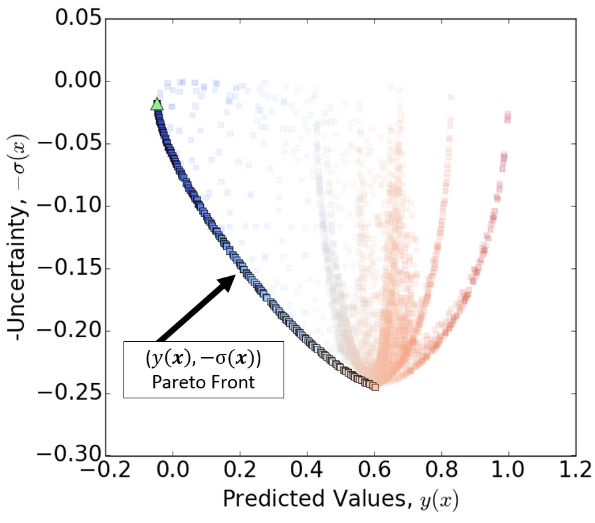


Figure D.3.1: EI Criterion Space

The $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ criterion space plot is shown in Figure D.3.2. In this plot we can see that for the plotted case, the EI-Pareto front designs lie on the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front. But we can also see that the EI-Pareto front covers a smaller section of the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front range.



(a) EI-Pareto Optimal Set



(b) $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto Optimal Set

Figure D.3.2: Exploit Explore Criterion Space

It is also possible to map both the EI-Pareto front designs, as well as the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front designs back to the design domain. We show an example of such a plot in Figure D.3.3. The design domain range for the EI-Pareto front is much smaller than the design domain range for the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front.

We ran EGO-MO on three different test functions listed in Chapter 3, using three and ten designs per iteration. These figures are listed in the electronic appendix (See Appendix B). In Section D.3.2 and D.3.3 we discuss some of the behaviour we noticed.

D.3.2 EI-Pareto Front Designs are more tightly clustered

In Figure D.3.3 we can see that the designs on the EI-Pareto front are tightly clustered. We found that this is the case for many of the iterations.

EGO intelligently decides in each iteration whether it will exploit or explore. This characteristic of a method intelligently deciding when to exploit and when to explore is in general a positive characteristic. We found that EGO-MO also follows this behaviour but selects designs quite tightly clustered.

This behaviour is favourable in certain circumstances, and less favourable in others. It

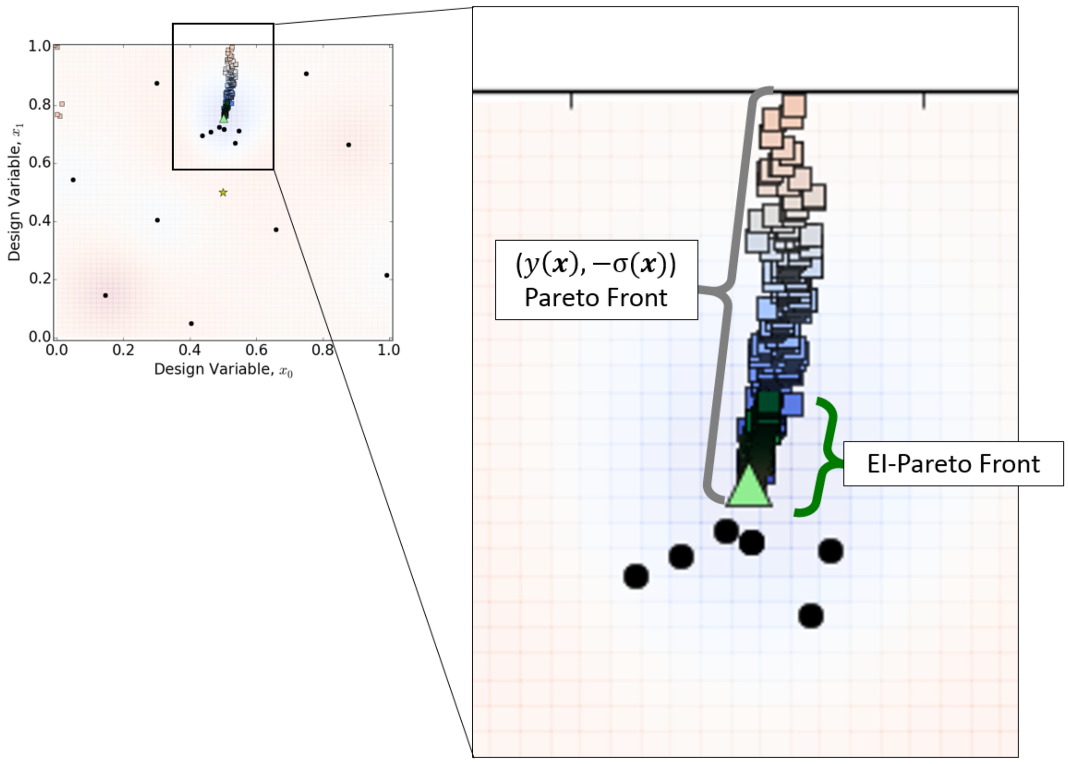


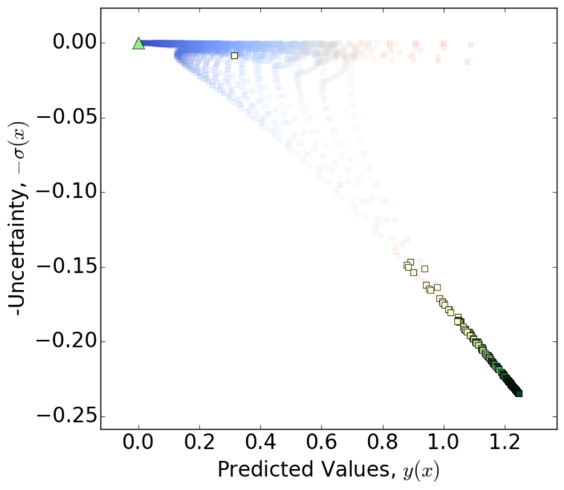
Figure D.3.3: Design Space Plot

is especially favourable on unimodal objective functions when only three designs are selected per iteration. EGO-MO, like EGO, is a exploitative method. When EGO-MO selects three designs per iteration and selects three designs close to the current predicted minimium objective function value, the response surface accuracy is improved around the minimum, and EGO-MO will converge quite fast.

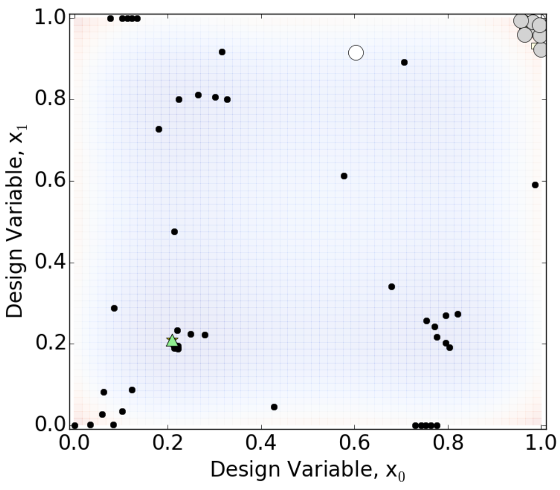
However, when ten designs are added around the current predicted minimium objective function value and the surface is not yet very accurate, ten designs are added in a cluster, where one design would have provided the same amount of information.

The worst case is when all the EI-Pareto front designs are clustered where the objective function value is high. In such a case, EGO-MO will select ten designs in an area where the predicted objective function value is high. This is the same unwanted behaviour we encountered on the SIMPLE-EGO Unwise Explore method (discussed in Section 4.4).

In Figure D.3.4 we present an example of EGO-MO selecting ten designs on the Styblinsky-Tang function. In Figure D.3.4 (a) we present the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ criterion space plot. In this figure we can see the EI-Pareto front designs are all located towards the exploration side of the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front. In Figure D.3.4 (b) we can see the selected designs are clustered in the corner of the design domain, where the predicted function values are between 0.8 and 1.2. This behaviour is not favourable. It could be because of this clustering issue that Feng et al. [16] reported linear speedup for EGO-MO selecting 3 designs per iteration, but worse speedup when EGO-MO selected more designs per iteration.



(a) Exploit-Explore Criterion Space Plot, Indicating the EI-Pareto Front Designs



(b) Design Space Plot for EGO-MO Selected Designs

Figure D.3.4: EGO-MO selecting 10 designs per iteration on the Styblinsky-Tang test function

D.3.3 EI-Pareto Front Becomes difficult to solve

In Section 3.3 we discussed the difficulty in finding the maximum of the EI function. As an optimization run progresses, both the value of the best predicted function value $y_{\boldsymbol{x}}$, and the design domain uncertainty $\sigma(\boldsymbol{x})$ decrease. This reduces EI to zero in large sections of the design domain.

Furthermore, designs become clustered around the minimum predicted objective function value, driving the standard deviation in that region even lower. These characteristics make the maximum of EI very difficult to find after EGO has been run for a few iterations, since it occurs often that the maximum EI value is located in a very small basin.

EGO-MO encounters these same problems, and the Pareto front becomes difficult to solve as the optimization progresses. After 20 iteration we are no longer able to solve the EI-Pareto front. In Figure D.3.5 we plot the EI-Pareto front designs (in green) in the $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ criterion space. As a result the EI-Pareto front designs are scattered. The $(y(\boldsymbol{x}), -\sigma(\boldsymbol{x}))$ Pareto front can still be solved, indicated using the red-blue scaled squares.

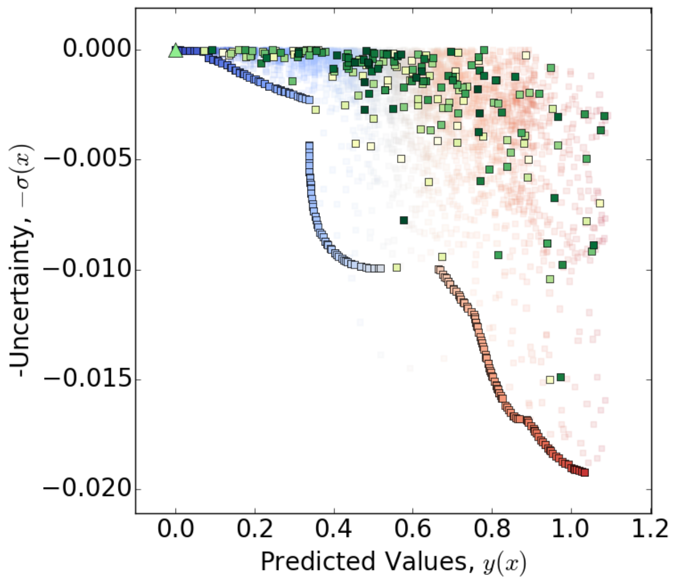


Figure D.3.5: EI-Pareto Front Points not Practically Solvable

D.4 Conclusion

EGO-MO is a multi-objective parallelization algorithm for EGO. EGO-MO mimics the behaviour of EGO, by intellegently selecting whether to exploit or explore, based on the information it has in an iteration. However, this intelligent decision making has some drawbacks. The designs EGO-MO selects in each iteration are often tightly clustered, which is not a good characteristic when many designs are added per iteration. Secondly, as EGO-MO progresses, the EI-Pareto front becomes very difficult to find. This characteristic makes it difficult for EGO-MO to continue converging.