A rubric for algorithm selection in optimization of black-box functions

by

Stefan Sremac

B.Ed., Canadian University College, 2008

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE COLLEGE OF GRADUATE STUDIES

(Mathematics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

June 2015

Abstract

When optimizing black-box functions, little information is available to assist the user in selecting an optimization approach. It is assumed that prior to optimization, the input dimension d of the objective function, the average running time t_f of the objective function and the total time T allotted to solve the problem, are known. The intent of this research is to explore the relationship between the variables d, t_f, and T and the performance of five optimization algorithms: Genetic Algorithm, Nelder-Mead, NOMAD, Efficient Global Optimization, and Knowledge Gradient for Continuous Parameters. The performance of the algorithms is measured over a set of functions with varying dimensions, function call budgets, and starting points. Then a rubric is developed to assist the optimizer in selecting the most appropriate algorithm for a given optimization scenario.

Based on the information available prior to optimization, the rubric estimates the number of function calls available to each algorithm and the amount of improvement each algorithm can make on the objective function under the function call constraint. The rubric reveals that Bayesian Global Optimization algorithms require substantially more time than the competing algorithms and are therefore limited to fewer function call budgets. However, if the objective function requires a large running time, this difference becomes negligible. With respect to improvement, the rubric suggests that Derivative Free Optimization algorithms are preferred at lower dimensions and higher budgets, while Bayesian Global Optimization algorithms are expected to perform better at higher dimensions and lower budgets. A test of the claims of the rubric reveals that the estimate of function call budget is accurate and reliable, but the improvement is not estimated accurately. The test data demonstrates large variability for the measure of improvement. It appears that the variables d, t_f, and T are insufficient for describing the expected performance of the assessed algorithms, since variables such as function type and starting point are unaccounted for.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to acknowledge the support of my loving, faithful, and patient wife, Jodi, who has willingly chosen to endure the life of a student's wife. You deserve no small credit for my completion of this program.

My journey toward this milestone would not have been possible without the support and faith of my family: Mom and Dad, Lidja and Nick, Ana and Josh. Your unbridled support of my educational pursuits has made this program significantly more enjoyable.

The time I have spent in Kelowna has been overwhelmingly positive due to some excellent friends and a loving church family.

I want to express my gratitude to the amazing and caring faculty and staff of the Mathematics and Statistics department at UBC's Okanagan campus.

I am thankful for the fellow graduate students and post-doctoral fellows with whom I shared my time at this university. I have enjoyed learning with and from you.

I appreciate Dr. Yves Lucet and Dr. Abbas Milani for taking the time to be a part of my thesis committee.

For the completion of this thesis, I am greatly indebted to the supervision of Dr. Warren Hare and Dr. Jason Loeppky, whose guidance and input has been extremely valuable. I appreciate the time you have invested in me and the respect with which I have been treated.

The research presented in this thesis would not have been possible without the generous funding received from the University of British Columbia, in the form of the Graduate Entrance Scholarship and the University Graduate Fellowship, and the funding received from the Natural Sciences and Engineering Research Council of Canada (NSERC) in the form of the Discovery Grants of Dr. Hare and Dr. Loeppky.

Finally, I am thankful to God for the fulfilling life that I have been given, for the joys I experience each day, and for His unfailing love to me.

Dedication

To my parents, Sima and Brankica, and to my wife, Jodi. You have sacrificed the pursuit of your own interests for the sake of mine.

Chapter 1

Introduction

The last century has seen an incredible increase in computing power. Consider that in the early years of the 20th century, the challenge for inventors in computing was to create a compact machine to perform addition, subtraction, multiplication, and division with ease and timeliness, [CKAEY14]. Certainly, it has been a long time since such a machine was at the forefront of computational innovation. In addition to the increase in computing power, innovations of the last decades have made this power commonly available through personal computers and smart phones. One of the results of such an environment is that computers have been used to create simulations of processes that are quite complex in real life. In the first chapter of [SWN03] several examples of computer simulators are presented. Among them are computer simulators that model the evolution of fires in closed spaces, the design of prosthesis devices, and the design of helicopter blades. In many instances the computer simulation of an event is simpler, faster, and cheaper to perform than it is to observe the actual event. The intensive computations required by a simulator, which were unfathomable only decades ago, are at present the best choice in certain applications.

One application of computer simulations is for performing experiments which would be challenging or impossible to perform in reality. Consider testing the spread-rate of an infectious disease in a human population, [BBC+02]. For obvious reasons this test would be unethical to perform without the use of a computer simulation. Such simulations will be referred to as *computer experiments*. While we may express computer experiments as functions, the code that comprises them can not usually be expressed analytically and thus very little is known about the nature of the function. For this reason, computer experiments are referred to as *black-box* functions.

In some computer experiment applications finding the optimal output value and the corresponding input values is the main objective. Consider a computer experiment that simulates a car crash, such as the one described in [JSW98]. The car design that performs the best in the computer experiment is the one that should be tested in a real experiment and produced if the results remain positive. The following subsections present the motivation

behind this work and a formal definition of the problem.

1.1 Motivation for Algorithm Comparison

Black-box functions present two challenges to optimizers. The first is that common optimization methods using *gradients*, *convexity*, *linearity* and other properties of the objective function do not apply to a black-box function since it lacks an analytic expression. The second challenge is regarding the time required to evaluate the black-box function. Some computer simulation functions may require significant computational time, such as Ford Motor Company's crash simulation which can take up to 106 hours to run, [WS07]. There are, however, black-box functions which are far less expensive in terms of time. In instances where the black-box is very expensive an algorithm that is near the optimal value in as few function evaluations as possible is desirable, while computationally cheap functions do not impose such a requirement. This demonstrates that the computational time of a function should be taken into account when choosing an optimization algorithm in the black-box context.

The motivation behind this thesis is in part due to claims made by [JSW98] and [SFP11] regarding *Bayesian Global Optimization (BGO)*. In [JSW98] it is said that the method "often requires the fewest function evaluations of all competing methods" and in [SFP11] the authors look for an algorithm that can "give satisfactory results with as few function evaluations as possible." The downside however is that the algorithm will "spend extra time deciding where [it] would like to evaluate the function next," [SFP11]. In addition, both of the works intend to minimize an *expensive* black-box function, so that the time taken by the algorithm itself, even if very long, will be small relative to obtaining the function values. Naturally, several questions arise.

- Are BGO algorithms really the best choice when the black-box function is very expensive?
- How expensive does a black-box function have to be in order for BGO algorithms to be the preferred choice?
- If the black-box function is not very expensive, is another family of algorithms a better choice?
- Does the input dimension of the function have an effect on choice of algorithm?

– Can a rubric be developed to select the best performing algorithm based on function evaluation time and input dimension?

To answer the above questions, the performance of several competing algorithms, including BGO algorithms, will need to be compared. This thesis focuses on algorithms from three different families. The first family of algorithms used for optimizing black-box functions are *heuristic methods*. By nature this family of optimizers does not have a strong theoretical framework and it may not even be guaranteed to converge. However, heuristic methods are, in general, simple to understand and implement, and are practically motivated. The second family of algorithms is the *Derivative Free Optimization* (DFO) family. These algorithms tend to use function values to explore the geometry of the objective function and move towards the optimal value. The third family of algorithms used for black-box optimization is the BGO family. These algorithms use statistical tools to create a model of the true function which is cheaper to compute. New points to explore, are then chosen based on this model function. Chapter 3 provides a deeper analysis of each of the algorithms as well as the implementations used.

1.2 Definition of the Problem

To formally define the optimization problem let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuous black-box function with input variable $x \in X \subset \mathbb{R}^d$, where X is the input domain. In this thesis, the domain is the unit hypercube, $X \equiv [0,1]^d$. The black-box function is assumed to be *deterministic*, that is, for any $\bar{x} \in X$ there exists a $\bar{y} \in \mathbb{R}$ such that $f(\bar{x}) = \bar{y}$ each time $\bar{x}$ is evaluated. Deterministic functions essentially have no measurement error. It is assumed that the user aims to minimize the black-box function. Mathematically, the problem is written as

$$\begin{aligned} \text{minimize} \quad & f(x), \\ \text{subject to} \quad & x \in X. \end{aligned} \tag{1.1}$$

Let X^* denote the argmin of f, that is,

$$X^* = \arg\min_{x \in X} f(x) := \{x \in X : f(x) \leq f(y) \text{ for all } y \in X\}.$$

The optimization domain X is a *compact* set in $\mathbb{R}^d$ and f is continuous function mapping X to $\mathbb{R}$ thus the generalized Extreme Value Theorem (refer to Chapter 5 of [DD02]) guarantees that X^* is non-empty. Let f^*

denote the minimum value of f over the domain X, that is,

$$f^* = \min_{x \in X} f(x) := \{f(x^*) : x^* \in X^*\}.$$

1.3 Thesis Outline

This thesis is laid out as follows. In Chapter 2 some background information is provided regarding Gaussian Process models which are used by two of the algorithms considered in the experiment. Chapter 3 is a presentation of the algorithms used in this thesis. In Chapter 4 the development of the rubric is documented. Section 4.1 describes the framework of the rubric and Section 4.2 details the estimation of some functions crucial to the rubric. Chapter 5 contains a demonstration of the rubric, an analysis of implications arising from its results, and a test of the reliability of the rubric. Finally in Chapter 6 some conclusions are drawn and future research directions discussed.

Chapter 2

Gaussian Process Model

Two of the algorithms used in this thesis adopt a model based approach, where the objective function f is replaced by an estimate of the function (often referred to as a *surrogate* or *model*.) One method for creating a model of the true function is to use a Gaussian process. What follows is a brief discourse on model building with the Gaussian process. To be consistent with notation commonly used with Gaussian processes, y will denote the objective function instead of f.

Assume that the objective function y has been evaluated N times at the points $x^1, x^2, \ldots, x^N$. This initial sample of N points should be sufficiently large and provide a good coverage of the domain so that a good model can be developed. In [LSW09] it is argued that $10d$ is a sufficient size for an initial sample. Now the task is to predict the function y based on this sample of N points. The next step in building a model of y requires some knowledge of a *stochastic process*.

Informally, a stochastic process $Y = \{Y_t : t \in T\}$ is a family of independent random variables specified in some domain T. When T is a singleton a stochastic process is just a random variable and when T is a set of finitely many elements the stochastic process is a vector of random variables. However, when T is a set of infinitely many elements, the stochastic process is a random function. A stochastic process can therefore be viewed as an extension of the concept of random variables to functions. A *Gaussian* process, then, is a stochastic process where each random variable has a Gaussian distribution. For a detailed presentation of GPs refer to Chapter 2 in [SWN03]. The Gaussian assumption on the stochastic process is made for the ease of working with Gaussian distributions.

Adopting a Bayesian approach, prior knowledge of y is specified by the GP, $Y = \{Y_x : x \in X\}$. Then the random variables $Y_{x^1}, Y_{x^2}, \ldots, Y_{x^N}$ associated with the N sampled points have a multi-variate normal distribution with mean vector μ_{Y_N} and variance-covariance matrix $\mathbf{\Sigma}_{Y_N Y_N}$. Let $y_N = (y(x^1), y(x^2), \ldots, y(x^N))^T$ be the vector of N observed function values. Then the posterior distribution of any $Y_x \in Y$ conditioned on the

observed values y_N is normal and specified by

$$\mu_{Y_x|y_N} := E[Y_x|y_N] = \mu_{Y_x} + \mathbf{\Sigma}_{Y_xY_N}\mathbf{\Sigma}^{-1}_{Y_NY_N}(y_N - \mu_{Y_N}) \tag{2.1}$$

and

$$\sigma^2_{Y_xY_x|y_N} := \text{cov}[Y_x, Y_x|y_N] = \mathbf{\Sigma}_{Y_xY_x} - \mathbf{\Sigma}_{Y_xY_N}\mathbf{\Sigma}^{-1}_{Y_NY_N}\mathbf{\Sigma}^T_{Y_xY_N}, \tag{2.2}$$

where $\mathbf{\Sigma}_{Y_xY_N}$ is an N-length vector of covariances between Y_x and each element of Y_N; $\mathbf{\Sigma}_{Y_xY_x}$ is the variance, $\sigma^2_{Y_x}$, of the random variable Y_x; μ_{Y_x} is the mean of the random variable Y_x. The posterior distribution is really an estimate of the function y. The equations (2.1) and (2.2), as shown in [CMMY91], then become

$$\hat{y}(x) = \mu_{Y_x} + \mathbf{\Sigma}_{Y_xY_N}\mathbf{\Sigma}^{-1}_{Y_NY_N}(y_N - \mu_{Y_N}), \tag{2.3}$$

which is the approximation to y at x and

$$\sigma^2_x = \mathbf{\Sigma}_{Y_xY_x} - \mathbf{\Sigma}_{Y_xY_N}\mathbf{\Sigma}^{-1}_{Y_NY_N}\mathbf{\Sigma}^T_{Y_xY_N}, \tag{2.4}$$

which is the uncertainty of the estimation in (2.3).

Models created using the GP provide an estimate of the true function but also provide a measure of the uncertainty that comes along with the estimate at each point. The uncertainty is manifested in the variance, (2.4), of $\hat{y}$ and is heavily relied on by two of the algorithms presented in Chapter 3. Also note that the costly operation of inverting the matrix in (2.3) does not depend on the unobserved point x and therefore when $\mathbf{\Sigma}^{-1}_{Y_NY_N}$ has been computed once, $\hat{y}$ can be computed quickly for multiple points in the domain. Equations (2.1) and (2.2) are written for multiple points in [CMMY91]. The requirements necessary for $\mathbf{\Sigma}^{-1}_{Y_NY_N}$ to be invertible will be discussed shortly.

Equations (2.3) and (2.4) require knowledge of the mean vectors, μ_{Y_x} and μ_{Y_N}, the covariance vector $\mathbf{\Sigma}_{Y_xY_N}$, the covariance matrix $\mathbf{\Sigma}_{Y_NY_N}$ and the variance $\sigma^2_{Y_x}$. All of these values will have to be estimated and to make the task simpler, in the prior specification of y (that is the GP) it will be assumed that each $Y_x \in Y$ will have the same mean, μ, and variance, σ^2. Equations (2.3) and (2.4) then simplify to

$$\hat{y}(x) = \mu + \mathbf{r}_{Y_xY_N}\mathbf{R}^{-1}_{Y_NY_N}\left(y_N - \mu\mathbf{1}\right), \tag{2.5}$$

and

$$\sigma^2_x = \sigma^2\left(1 - \mathbf{r}_{Y_xY_N}\mathbf{R}^{-1}_{Y_NY_N}\mathbf{r}^T_{Y_xY_N}\right), \tag{2.6}$$

where $\mathbf{1}$ is an N-length vector of ones; $\mathbf{r}_{Y_x Y_N}$ is an N-length vector of correlations; $\mathbf{R}_{Y_N Y_N}$ is an $N \times N$ matrix of correlations. The variance was factored out of the covariance matrices to obtain the correlation matrices. Now only one μ and one σ^2 need to be estimated which is a simpler task than estimating $2N + 2$ parameters as is the case in equations (2.3) and (2.4).

Two more terms from equations (2.5) and (2.6) remain to be discussed, the correlation vector and the correlation matrix. The initial assumption on the objective function y is that it is continuous over X. This implies that if $x^1, x^2 \in X$ are close to one another their function values should be similar. In terms of correlation, two input points close to each other should be highly correlated. Points far away from each other will exhibit a low correlation. This leads to a correlation function of the form

$$\text{corr}(Y_{x^1}, Y_{x^2}) = \prod_{i=1}^{d} R(x_i^1 - x_i^2), \tag{2.7}$$

where R is a correlation function in one dimension and depends on the distance between x_i^1 and x_i^2. Recall that each $x \in \mathbb{R}^d$, thus the product is over the dimensions. Equation (2.7) is referred to as a product correlation and allows for control and analysis of the effects of each dimension. A possible choice for R is the squared-exponential correlation

$$R(x_i^1 - x_i^2) = e^{-\theta_i (x_i^1 - x_i^2)^2}, \tag{2.8}$$

where $\theta_i \geq 0$ is an unknown parameter. Variants of this exponential correlation are used in [SFP11] and [JSW98]. In the above equation $R(x_i^1 - x_i^2)$ will be close to 1 when $x_i^1 - x_i^2$ is close to 0 which is in agreement with the continuity assumption on y. Taking equation (2.8) into the product correlation of (2.7) gives

$$\text{corr}(Y_{x^1}, Y_{x^2}) = \prod_{i=1}^{d} e^{-\theta_i (x_i^1 - x_i^2)^2}. \tag{2.9}$$

Each θ_i is a parameter specific to dimension i. Small values of θ_i can indicate that input variable i has a negligible contribution to the function y. Constructing the correlation matrix from (2.5) and (2.6) using the correlation function from (2.9) ensures that the matrix is positive definite and hence invertible, [SWN03]. The squared-exponential correlation from (2.8) is only one type of correlation function and Chapter 2 in [SWN03] presents some other correlation functions.

Equations (2.5) and (2.6) have $d + 2$ parameters which need to be estimated. The most commonly used method of estimating the parameters is by the well known maximum likelihood estimation which will not be described in this thesis. For information on this step of the process refer to Chapter 3 of [SWN03]. Once the parameters are estimated, (2.5) and (2.6) are fully known and the model function $\hat{y}$ is fully known.

Chapter 3

Algorithms

Prior to presenting each of the algorithms, the notation used, as well as, notions of convergence will be discussed. Let $k \in \mathbb{N}$ denote the iteration of the optimization algorithm. Let $S^k = \{x^0, x^1, \ldots, x^{|S^k|}\}$ be the ordered set of all points, $x \in X$, that have been sampled from iteration 0 up to and including iteration k. The points in S^k are ordered chronologically so that x^0 is the first point sampled and $x^{|S^k|}$ is the most recently sampled point. Then following the completion of iteration k, the best function value obtained is

$$f^k_{\text{best}} := \min_{x \in S^k} f(x).$$

Similarly, the best point obtained by the end of the kth iteration is

$$x^k_{\text{best}} := \{x \in S^k : f(x) = f^k_{\text{best}}\}.$$

If multiple points satisfy the requirements of x^k_{best}, the one with the lower index in S^k is taken, thus keeping x^k_{best} a singleton. An algorithm is said to converge to the true minimum value if

$$\lim_{k \to \infty} f^k_{\text{best}} = f^*. \tag{3.1}$$

Let $\text{dist}(x^k_{\text{best}}, X^*)$ be the distance between x^k_{best} and X^*, defined as

$$\text{dist}(x^k_{\text{best}}, X^*) := \inf_{x \in X^*} \left\| x^k_{\text{best}} - x \right\|_2.$$

Then, an algorithm is said to converge to a true minimizer if

$$\lim_{k \to \infty} \text{dist}(x^k_{\text{best}}, X^*) = 0. \tag{3.2}$$

In the following sections each of the algorithms is presented in greater depth along with some convergence results. All implementations are in the programming language `R`.

3.1 Genetic Algorithm

The genetic algorithm is a heuristic approach that was formally introduced in [Hol75] and has since become a well-known option to optimizers. The basic idea behind this algorithm is to mimic biological evolution and using "survival-of-the-fittest" obtain the optimal value of the function.

At iteration k, the algorithm considers a set of points, say

$$P^k = \{x^0, x^1, \dots, x^{|P^k|}\} \subset \mathbb{R}^d.$$

The initial sample of points, P^0, can be chosen randomly, but each subsequent set is derived from the previous. Since the ideology behind the algorithm is that of biological evolution, each point x^i, $i = 1, 2, \dots, |P^k|$, is viewed as a member of a population P^k and the components of the vector x^i are its 'genes'. The constraint set X is the minimal requirement for survival in this population and any individual $x^i \notin X$ will fail to survive or pass its genes to the next generation. In the next step, we use the fitness function, g, to determine the fitness of each individual in P^0. For the sake of the application in this thesis we let $g = -f$ so that smaller objective values correspond to higher fitnesses. And of course we have $f(x^*) = -g(x^*)$ for any $x^* \in X^*$.

Based on the fitness values of the members of P^k, three steps will be taken to determine the next population, P^{k+1}. In the first step, *Selection*, the fittest individuals from P^k will be selected to survive into the next generation P^{k+1}. The second step, *Cross-over*, selects sufficiently many parent points from P^k and breeds them to create offspring. To stay true to the biological motivation for this algorithm, in this step the parent points are viewed as chromosomes and the components of each parent chromosome are its genes. As a result of the chromosomal motivation the individuals in P^k are often encoded, with binary encoding being a common choice. Cross-over will create an offspring by using some genes from each parent and this process generally relies on probabilities to determine which genes are taken from which parent. If the chromosomes are not encoded, a convex combination of the parents, where the parameter of convexity is chosen randomly, is a common choice. The final step, *Mutation*, corresponds to the mutation of genes which takes place in biological reproduction. Here the offspring are perturbed by changing the value of some of their genes. The probability of a gene mutating is governed by a mutation parameter which is generally small so as to not deviate from the biological model. Once all the steps have been completed and P^{k+1} has been created, the process is

repeated to obtain future generations. A thorough description of the genetic algorithm can be found in [Hol75].

In order to determine a stopping point for the genetic algorithm, it will help to first present the theoretical convergence results. The introduction of random choices in the *Cross-over* and *Mutation* steps at each iteration places the genetic algorithm in the category of stochastic search methods. As such, the convergence shown in (3.1) and (3.2) is replaced by equations involving probabilities. In [EAVH91], the authors show that under some assumptions the genetic algorithm will converge to the true minimum with probability 1 as the number of iterations approaches infinity. That is,

$$Prob\left[\lim_{k\to\infty} f^k_{\text{best}} = f^*\right] = 1 \quad \text{and} \quad \lim_{k\to\infty}\left(Prob\left[P^k \cap X^* \neq \emptyset\right]\right) = 1.$$

With this in mind, there is no theoretically preferred event at which to stop the algorithm. One common way the algorithm may be stopped is when a function value budget, say $\texttt{N}_{\texttt{feval}}$, has been reached. Alternatively the algorithm may be stopped if the number of generations where x^k_{best} has failed to improve, exceeds the parameter $\texttt{N}_{\texttt{fix}}$. The first criteria stops the algorithm if it has taken too long while the second terminates it if too much time has passed without improvement to the optimal value.

The implementation used in this thesis can be found in the `R` package `GA` which is documented in [Scr13]. The population size of $\max\{10, \sqrt{\texttt{N}_{\texttt{feval}}}\}$ is used at each iteration and the top 5% from each population are assured survival into the next generation. The mutation parameter is 0.1 and the algorithm is stopped only when the function call budget has been exhausted.

3.2 Nelder-Mead

The Nelder-Mead (NM) method was introduced in [NM65] and is a minimization method utilizing only objective function values. The approach moves a special type of polytope called a *simplex* around the domain space in search of the best function value.

A simplex in $\mathbb{R}^d$ is a d-dimensional polytope with $d+1$ vertices. In $\mathbb{R}^2$, for example, a simplex is formed by the edges and interior of a triangle. At iteration k the NM method considers $d+1$ points $\{x^0, x^1, \ldots, x^d\}$ which are the vertices of a simplex. The points are then ordered so that $f(x^0) \leq f(x^1) \leq \cdots \leq f(x^d)$ and a search is performed to replace the worst point, x^d. Up to four points may be evaluated along the line passing through x^d and $\sum_{i=0}^{d-1} x^i/d$. If a better function value is found the corresponding point

replaces x^d, otherwise the simplex shrinks towards the best point x^0. The approach guarantees that the next point(s) chosen along with the ones that remain are the vertices of a simplex in $\mathbb{R}^d$ and the process is repeated in subsequent iterations.

For objective functions that are bounded below, the NM approach guarantees convergence to some function value $\bar{f}$, [LRWE98]. That is, if $X^* \neq \emptyset$, then

$$\lim_{k\to\infty} f^k_{\text{best}} = \bar{f}.$$

While the assurance of convergence is desirable, the accumulation point is cause for some concern. It is important to note that the statement of convergence develops no relationship between $\bar{f}$ and f^*. Most notably, $\bar{f}$ is not necessarily the minimum value. An example showing the failure of the method to converge to a stationary point on a reasonable function was presented in [McK98]. A second concern is that the NM method is a local search method in that it makes decisions based on the local geometry of the objective function. This may cause convergence to a local minimum while ignoring the global minimum.

The simplicity of this method makes it readily implementable and the fact that gradients are not needed is appealing to many users. In practice, the algorithm performance can range from very good to quite poor, [Wri96]. The implementation used in this thesis is that of [VB11] which is an `R` implementation of the algorithm presented in [Kel99].

3.3 Pattern Search

The pattern search method was first introduced as a *direct search* algorithm by [HJ61]. The name 'direct search' was used to indicate that only objective function values were to be used in the optimization process. Subsequent algorithms such as the one presented in [DT91] expanded on the original method while staying true to the core ideas. As such, the term 'pattern search' does not refer to a specific algorithm, instead it encompasses a range of algorithms which share several intrinsic properties.

At the kth iteration a pattern search algorithm has a best point x^k_{best} and the corresponding best function value f^k_{best}. The algorithm then explores points surrounding x^k_{best} in a *pattern* of directions and a fixed step size. If a better function value is found, the corresponding point replaces x^k_{best} and the process is repeated in the next iteration. For the case where no better function value is found, the step size is decreased and the same exploratory step performed.

Like the NM algorithm, pattern search is a local optimization method and this fact should be considered when global optimization is desired. A simple way to combat local optimization is to start the algorithm at multiple locations in the domain. The algorithm then explores the area local to each starting point and the combined results provide information about the global optimum. The pattern search approach is simple, intuitive and more importantly it is flexible and has strong convergence results. In [Tor97], it is shown that if $f \in \mathcal{C}^1$, the level sets of f are compact, and some conditions are placed on the directions being searched at each iteration, then

$$\lim_{k\to\infty} \|\nabla f(x^k_{\text{best}})\| = 0.$$

This of course is not a guaranteed convergence to the minimum of the function, but there is at least a guarantee of convergence to a critical point. The power of this result is that it holds for *any* pattern search algorithm satisfying the aforementioned conditions.

With regards to the flexibility of pattern search, as mentioned earlier, [AD06] introduced a *search* step prior to the directional exploration which allows for any type of finite global search. This step has one main benefit in that it allows the algorithm to 'jump' out of local modes if a better function value is found elsewhere without losing the convergence results from [Tor97]. The pattern search algorithm used in this thesis is Nonsmooth Optimization by Mesh Adaptive Direct Search (NOMAD), [ABLD08], which is a modification of the Mesh Adaptive Direct Search (MADS) algorithm, [AD06].

In this algorithm a Variable Neighbourhood Search (VNS) subroutine, [HM03], is used to aid NOMAD in global optimization. Once the step size used in pattern search has become sufficiently small, the VNS algorithm is used to perturb x^k_{best} and obtain x'. Then a neighbourhood of x' is explored in a pattern of directions and increasing step sizes in search of x'' such that $f(x'') \leq f(x^k_{\text{best}})$. If no such point is found then pattern search continues around x^k_{best} otherwise $x^k_{\text{best}} = x''$. Coupling the local nature of pattern search with the global nature of VNS creates an algorithm more effective for global optimization.

The `R` implementation of NOMAD is sNOMADr and can be found in the package `crs`, [RN14]. The algorithm will be referred to as sNOMADr throughout this paper.

3.4 Efficient Global Optimization

The Efficient Global Optimization (EGO) algorithm was introduced in [JSW98] as an approach to minimizing expensive-to-compute black-box functions. The approach improves on several previous attempts at optimization using GP models, and claims to use a relatively small amount of function calls.

Suppose only N function calls can be afforded throughout the entire optimization process. Then the EGO algorithm begins by evaluating an initial sample of points $x^1, x^2, \ldots, x^m$ where $m < N$. Only a portion of the entire sampling budget should be used in this step. The suggestion of $10d$ points, mentioned in Chapter 2, is often used. Based on these m points, a GP model, $\hat{f}$, of the function is created. The authors then introduce an improvement criterion to select the next point to evaluate. The improvement criterion looks for an unsampled point $x \in X$ where its objective function value could be smaller than the current lowest function value. The improvement criterion is

$$I(x) = \max_{x \in X}(f_{best} - Y(x), 0), \tag{3.3}$$

where f_{best} is the smallest function value obtained so far, and $Y(x)$ is the random variable with mean and variance specified by (2.5) and (2.6) (that is the GP). If $Y(x)$ is smaller than the best function value obtained so far, then the improvement is the amount by which $Y(x)$ is better than the best function value. If $Y(x)$ does not improve on f_{best}, then the improvement at x will be 0. The improvement function is always non-negative and its maximum is at the point where the greatest improvement in function value will occur. However, the criterion in (3.3) contains a random variable and cannot be computed. Naturally this leads to the *expected improvement (EI)* criterion,

$$E[I(x)] = E\left[\max_{x \in X}(f_{best} - Y(x), 0)\right], \tag{3.4}$$

which is just the expectation of the improvement criterion in (3.3). The next point to be sampled is the maximizer of (3.4). The authors show that the expected improvement criterion can be expressed in closed form and optimized using a branch and bound algorithm. Once the maximizer has been found, its function value is obtained, a new GP function model is created using the $m + 1$ points and the EI criterion is computed again. These steps are repeated until the budget, N, is exhausted.

The EI criterion is the key element of the EGO algorithm. While earlier work in optimization with GP function models relied on heuristics, [JSW98], this work introduced a criterion based on theory. The criterion tends to explore regions of X where the GP function model exhibits low values, but also regions where the model exhibits high uncertainty. This balance between local and global search allows the algorithm to avoid stagnating at locally optimal points.

The EGO approach is not guaranteed to converge in general as shown in [Bul11]. The problem in convergence comes mainly from the constantly changing GP model by which the EGO approach makes most of its decisions. If the GP is not a good model of the true function, then convergence to the true minimum may not be likely.

EGO was designed for black-box optimization and that is why it is appropriate for this thesis. The implementation used is found in the `R` package `DiceOptim` by [GPR$^+$13].

3.5 Knowledge Gradient for Continuous Parameters

The Knowledge Gradient for Continuous Parameters (KGCP) was introduced in [SFP11] and is an extension of the previous work, [FPD09], which was considered in the discrete context. It is also an extension of the EGO algorithm in that it was designed for non-deterministic data. The KGCP is designed for maximization problems, so the objective function f which needs to be minimized will be replaced by $g = -f$.

The KGCP assumes that the data are noisy and that at any point $x \in X$,

$$\hat{y}(x) \sim N(g(x), \lambda(x)), \tag{3.5}$$

where $\hat{y}(x)$ is the noisy observation at x, $g(x)$ is the true objective value at x, and $\lambda(x)$ is the noise variance at x. Like the EGO algorithm the KGCP takes an initial sample of m points, $x^1, x^2, \ldots, x^m$. Let

$$\hat{Y} = \{\hat{y}(x^1), \hat{y}(x^2), \ldots, \hat{y}(x^m)\}$$

be the set of observed values from the initial sample. Then the GP model at some $x \in X$ is the expected value of the true function at $x \in X$ based on the m observations,

$$g^m(x) = E[g(x)|\hat{Y}]. \tag{3.6}$$

The superscript m is used to index the progression of models. The algorithm then considers sampling another point $x \in X$ and predicts g^{m+1} (the next model) without having the observation $\hat{y}(x)$. That is

$$\hat{g}^{m+1}(x) = E[g(x)|\hat{Y}, x]. \tag{3.7}$$

Use of the 'hat' notation in $\hat{g}^{m+1}$ is to indicate that (3.7) does not express the actual model of the function based on $m+1$ sampled points, since the $(m+1)$th point, x, has not yet been observed. The equation is instead a way to 'look ahead' at what the model might be like if it was built using the additional point x. Then the KGCP criterion at some $x \in X$ is

$$KGCP(x) = \left(E\left[\max_{i=1,\dots,m+1} \hat{g}^{m+1}\left(x^i\right) \middle| \hat{Y} \right] - \max_{i=1,\dots,m+1} g^m\left(x^i\right) \right)_{x^{m+1}=x}. \tag{3.8}$$

This criterion measures the improvement in function value if x was used to create the next model. The maximizer of this criterion is then the point x which is expected to create a model with the most improvement in function value over the current model. The $KGCP$ function is non-negative and multi-modal and similar to the EI criterion from EGO. A closed form expression of (3.8) as well as detailed connections of the above equations and statements can be found in [SFP11] and [FPD09].

The $KGCP$ criterion is differentiable and the derivative is available in closed form. This allows for use of gradient-based optimization techniques such as steepest ascent or BFGS. The authors suggest steepest ascent and use it in their implementation. It should be noted that the non-concavity of the $KGCP$ function means that any maximizer of (3.8) obtained by gradient-based methods may only be a local maximizer. As such a multi-start approach is used to increase the probability of reaching the global maximizer, although in practice this is not crucial, [SFP11].

The KGCP method continues sampling one point at a time until $N-1$ points have been sampled. By this time the model g^{N-1} should be very accurate and the last point is taken as the maximizer of this model. In [SFP11] if the KGCP is used for maximization, under some assumptions, it is shown that

$$\lim_{m\to\infty} Var\left[g^m\right] = 0. \tag{3.9}$$

This result implies that the KGCP sampling technique leads to GP models which approach the true function g. Eventually, the model g^m is indistinguishable from g. This in turn leads to the maximizer of g^m being indistinguishable from the maximizer of g. However, one of the assumptions for

this convergence result is that the parameters of the GP are known and fixed throughout the optimization process. In practice this is unlikely to be true since the parameters of the GP are unknown and estimated at each iteration of the algorithm. Thus the GP model is constantly changing and the concerns regarding EGO, expressed in [Bul11], are valid for the KGCP algorithm as well.

The KGCP approach was implemented in `R` by the author, using GP models created with the package `DiceOptim` by [GPR+13]. The steepest ascent algorithm used was also implemented in `R` and the BFGS optimizer is from the standard `optim` package in `R`. The implementation of KGCP used in this thesis can be acquired by contacting the thesis author.

Chapter 4

Developing a Rubric

In Section 1.1 the motivating questions behind this thesis were presented. This chapter is concerned with the development of a rubric to help answer those questions. Given the input dimension, the average time required to evaluate the objective function, and a time budget, the rubric should determine which of the algorithms in question will provide the best result. The following sections present the theoretical framework and development of the rubric.

4.1 Theoretical framework

In black-box optimization it can be assumed that prior to optimizing, the user will have some basic information regarding the problem. In this thesis it is assumed that the user will have an allotted time to perform the optimization. This will be referred to as the *time budget* and denoted by T. It is also assumed that the user knows the average time required for the objective function to evaluate a single point, denoted by t_f. Lastly, it is assumed that the user will know the input dimension, d, of the objective function. Now, for an optimization problem, the time, T, required to obtain a solution can be expressed as

$$T = T_f + T_{\mathcal{A}}, \tag{4.1}$$

where T_f is the time spent evaluating the objective function and $T_{\mathcal{A}}$ is the time used by algorithm $\mathcal{A}$ to decide which point(s) to evaluate next. The first term, T_f, is the time required for a single function evaluation multiplied by the number of function calls:

$$T_f = nt_f,$$

where n is the number of times the function is evaluated. The value n will be referred to as the *function call budget*. The second term, $T_{\mathcal{A}}$, of equation (4.1) is generally unknown, however, some relationships can be developed. First, the time used by the algorithm will be affected by the

function call budget, n. If more function evaluations are available, then more iterations will also follow. This, in turn, will require the algorithm to make more decisions, hence increasing $T_{\mathcal{A}}$. This relationship will be different for each algorithm due to a number of factors. One factor is the number of function calls an algorithm may perform at each iteration. For example, the implementation of the GA used in this thesis will evaluate $\max(10, \sqrt{n})$ points in each iteration. For $n \geq 100$ this leads to $\sqrt{n}$ iterations. The EGO algorithm, on the other hand, will initially obtain $2d + 2$ points and then evaluate the function once per iteration, leaving one function call for the end. This leads to $n - (2d + 2) - 1$ iterations. The second relationship that can be developed is that between $T_{\mathcal{A}}$ and the input dimension d. As already noted, the number of iterations performed by the EGO algorithm is affected by the input dimension. The same result is true for the KGCP algorithm. In the NM algorithm the number of vertices in the simplex is a function of the input dimension. A higher dimension will lead to a larger simplex which will increase the decision-making computations at each iteration, leading to an increased time $T_{\mathcal{A}}$. Similar arguments can be made for each of the algorithms used in this thesis, concluding that the time used by each algorithm is in part a function of n and d. The variables n and d may not be the only factors contributing to $T_{\mathcal{A}}$, however, in this thesis it is assumed that these are the only variables known prior to optimizing the objective function. While the functions $T_{\mathcal{A}}$ are not known, they will be estimated in Section 4.2.

The discussion from the previous paragraph leads to

$$T \approx nt_f + t_{\mathcal{A}}(d, n), \tag{4.2}$$

which is a reformulation of (4.1), where $t_{\mathcal{A}} : \mathbb{R}^2 \to \mathbb{R}$ is the time used by the algorithm and depends on the input dimension and the function call budget. In this equation T is a function of d, t_f, and n. Assuming that the function $t_{\mathcal{A}}$ is known for each algorithm $\mathcal{A}$, the user can obtain n. Since $t_{\mathcal{A}}$ is different for each algorithm, n will also be different for each algorithm. For example, an optimization problem with a given T, d and t_f, might lead to the GA evaluating the function 200 times while allowing the EGO algorithm to sample just 20 points.

This provides the user with at least a starting clue as to which algorithm may perform better. However, the number of function calls allotted to an algorithm may not be an accurate measure of performance, since it does not take into account, the nature of the algorithm. This brings up the need for a performance criterion. In this thesis the *function value improvement* approach is adopted, where the criterion measures how much improvement

an algorithm makes on the function value over the course of the optimization process.

Let $x^0 \in X$ be the first point evaluated by algorithm $\mathcal{A}$ and $x^n_{\text{best}} \in X$ be the best point evaluated by the end of the optimization process. Then the improvement in function value by algorithm $\mathcal{A}$ after n function evaluations is

$$\mathcal{I}^n_{\mathcal{A}} = -\log_{10}\left(\frac{f(x^n_{\text{best}}) - f^*}{f(x^0) - f^*}\right), \tag{4.3}$$

where f^* is the minimum of the function. In a good performance, $f(x^n_{\text{best}}) - f^*$ should be much smaller than $f(x^0) - f^*$, which will lead to a large improvement value $\mathcal{I}^n_{\mathcal{A}}$. Using the base 10 logarithm means that $\mathcal{I}^n_{\mathcal{A}}$ can be loosely interpreted as the digits of accuracy gained by algorithm $\mathcal{A}$. The improvement function is not known and most likely does not exist in the general case, since it can be affected by a large number of different factors. First of all, "steep" objective functions may provide greater improvement values than "flat" objectives. Another factor regards the starting point, x^0. Suppose that $f(x) = e^x$ and $x^0 = 10$. Some algorithms may make large improvements on the function value starting at such a "steep" location on the graph of the objective. If instead the same algorithm was used with starting point $x^0 = -10$, the algorithm may not demonstrate a drastic improvement in function value, since it started on a flat spot of the function. To account for these issues, the improvement is normalized as

$$I^n_{\mathcal{A}} = -\log_{10}\left(\frac{f(x^n_{\text{best}}) - f^*}{f(x^n_{random}) - f^*}\right), \tag{4.4}$$

where $f(x^n_{random})$ is the best function value obtained after n randomly sampled points from X. The assumption in this formulation, is that if algorithm $\mathcal{A}$ obtains abnormally large improvement on a given function, the random sampling technique will also obtain abnormally large improvement. The ratio, therefore, is to stabilize the measure of improvement which can now be loosely interpreted as how many more or less digits of accuracy algorithm $\mathcal{A}$ can obtain compared to a random sample of n points.

The question, now, is whether $I^n_{\mathcal{A}}$ is related to the information available at the beginning of the optimization process, namely d and n? In this section the existence and strength of relationships is developed in the theoretical context. In Section 4.2, the relationships are studied in an empirical setting. First, the variable n is considered.

Let f^n_{best} be the lowest function value obtained after n function calls. With knowledge of f^n_{best}, a good optimization algorithm will not choose

f_{best}^{n+1} if it is not an improvement on the current best value. Such a policy ensures that for any $m \geq 0$

$$f_{best}^{n+m} \leq f_{best}^{n}.$$

Manipulation of this inequality leads to

$$\begin{aligned} & f_{best}^{n+m} - f* \leq f_{best}^{n} - f* \\ \implies \quad & \frac{f_{best}^{n+m} - f*}{f_{best}^{n} - f*} \leq 1. \end{aligned}$$

The above manipulations are only permitted if $f_{best}^{n} \neq f*$. Corresponding assumptions for the random sampling technique lead to

$$\frac{f_{random}^{n+m} - f*}{f_{random}^{n} - f*} \leq 1.$$

Now if it is assumed that

$$\frac{f_{best}^{n+m} - f*}{f_{best}^{n} - f*} \leq \frac{f_{random}^{n+m} - f*}{f_{random}^{n} - f*} \tag{4.5}$$

then

$$\begin{aligned} & \frac{f_{best}^{n+m} - f*}{f_{random}^{n+m} - f*} \leq \frac{f_{best}^{n} - f*}{f_{random}^{n} - f*} \\ \implies \quad & \log_{10}\left(\frac{f_{best}^{n+m} - f*}{f_{random}^{n+m} - f*}\right) \leq \log_{10}\left(\frac{f_{best}^{n} - f*}{f_{random}^{n} - f*}\right) \\ \implies \quad & -\log_{10}\left(\frac{f_{best}^{n+m} - f*}{f_{random}^{n+m} - f*}\right) \geq -\log_{10}\left(\frac{f_{best}^{n} - f*}{f_{random}^{n} - f*}\right) \\ \implies \quad & I_{\mathcal{A}}^{n+m} \geq I_{\mathcal{A}}^{n}. \end{aligned}$$

The implication is that improvement is non-decreasing with budget. Which might make sense intuitively, however, the normalization in the improvement criterion presents a challenge for this intuitive idea. The main challenge is with assumption (4.5). This assumption is that the amount by which algorithm $\mathcal{A}$ approaches f^* over the additional m function calls is greater than the amount by which the random sample approaches f^* over the additional m function calls. For algorithms such as NM and sNOMADr, where there is

a possibility of converging to a local minimum, such an assumption cannot be made.

The previous assumption that

$$f^{n+m}_{best} \leq f^n_{best}$$

is certainly true if the the first n sampled points are identical, but it is not true in general. The improvement gained by an algorithm may be affected by the starting point or randomness and the statement,

$$f^{n+m}_{best} \leq f^n_{best},$$

may not be true if the values are a result of two distinct algorithm runs.

What has been demonstrated here, is that the assumptions required for a general theoretical statement about the relationship between improvement and function call budget are unwarranted. In the following section, the presence and strength of this relationship will be analyzed using experimental data and a statistical approach.

The other variable available at the beginning of the optimization process is the input dimension, d. The attempt to develop a relationship between improvement and n revealed some challenges due to the improvement being normalized. It is likely difficult to determine the type and amount of effect d will have on a given algorithm and the random sampling technique. Therefore theoretical analysis of this relationship will not be discussed in this thesis. The following section provides an empirical analysis of the relationship between improvement and the available variables d and n.

4.2 Estimating $t_{\mathcal{A}}$ and $I^n_{\mathcal{A}}$

To estimate the time and improvement functions, a number of test functions with varying input dimensions and function call budgets were optimized using the algorithms from Chapter 3. Regression was then performed on the resulting data to approximate $t_{\mathcal{A}}$ and $I^n_{\mathcal{A}}$. The test functions used are from [MS05] and [MGH81]. As was mentioned in the previous section, the function type could have an effect on the improvement function. Therefore the functions chosen for this experiment exhibit a variety of types. There are convex functions, multi-modal functions, functions with multiple minimizers, and highly erratic functions which have the appearance of noise. The functions in the test set have input dimensions varying from 2 to 30. A list of test function names and dimensions appears in Table 4.1. Note that some

of the functions, such as the Ackley and Rastrigin, are not dimension specific and can therefore be tested at multiple dimensions. Other functions, such as the Branin and drop-wave, are dimension specific and are therefore used at only one dimension. The descriptions of the key characteristics (relative to optimization) of each of the functions can be found in the third column of the table. For clarification, 'flat bowl' functions have a bowl shape and exhibit a very small slope in a relatively large neighbourhood of the minimizer. The variably dimensioned function has a very large slope in some directions and very small slope in other directions. The Extended Powell and Rosenbrock functions each have a single minimizer but have characteristics which are not easily categorized. The variety of test problems used is intended to minimize the effect of the test set on the results of this section.

Functions were optimized based on function calls ranging from 10 to 500. Every function was optimized 20 times by each algorithm. Each of the 20 repeats used a different starting point. The purpose of this approach was to take into account the effect of the starting point. For each optimization problem values necessary for computing the improvement criterion were stored and the algorithm was timed. The experiment was performed on a dedicated cluster so that the time values would be more reliable.

Once the data is available regression can be performed. The choice of regression technique for this data is linear regression. The main reason behind this choice is that linear regression provides a clean equation for the regression surface and can be used for statistical inference, [Far02]. Other optimization techniques such as spline regression or kernel regression, provide complicated regression equations. In the following subsections, the regression surfaces for $t_{\mathcal{A}}$ and $I^n_{\mathcal{A}}$ are developed. Since both $t_{\mathcal{A}}$ and $I^n_{\mathcal{A}}$ will be approximated by functions of d and n, $\hat{t}_{\mathcal{A}}(d, n)$ will be used to denote the estimate of the time function for algorithm $\mathcal{A}$ and $\hat{I}_{\mathcal{A}}(d, n)$ will be used to denote the estimate of the improvement criterion.

4.2.1 Time functions

It will be necessary to obtain a time function for each of the five algorithms from Chapter 3. A deeper analysis of the regression process will be shown for the EGO algorithm, while only the results will be presented for the remaining four algorithms, since the process is analogous.

Analysis of the data for the EGO algorithm begins with a three dimensional plot of the data shown in the left image of Figure 4.1.

Note that the time (measured in seconds) increases with both dimension and function call budget. The data appears to be "cut-off" somewhere above

Table 4.1: A list of the test functions used. The second column indicates the dimensions for which each function was used and the third column provides a brief description of relative function characteristics.

Function Name	Dimension(s)	Description
Ackley	3,7,10,14, 22,24,28,30	Multiple local minima
Branin	2	Multiple global minima
Broyden banded	2,3,4,6, 7,8,10,14, 22,24,28,30	Flat bowl
Broyden tri-diagnoal	2,3,4,6, 7,8,10,14, 22,24,28,30	Bowl
Discrete integral	2,4,5,6, 8,9,10,14, 18,22,26,30	Flat bowl
Drop-wave	2	Multiple local minima
Extended Powell	4,8	Other
Goldstein-Price	2	Multiple local minima
Linear full rank	2,4,5,6, 8,9,10,14, 18,22,26,30	Bowl
Linear rank one	2,4,5,6, 8,9,10,14, 18,22,26,30	Valley
Rastrigin	5,9,10,14, 18,22,26,30	Multiple local minima
Rosenbrock	4	Other
Rotated hyper-ellipsoid	3,7,10,14, 22,24,28,30	Bowl
Shekel 10	4	Multiple local minima
Six-hump camel	2	Multiple local/global minima
Sum of powers	3,5,7,9, 10,14,18,22, 24,26,28,30	Flat bowl
Sum of squares	3,7,14,22, 24,28,30	Bowl
Trigonometric	2,4,5,6, 8,9,10,14, 18,22,26,30	Multiple local minima
Variably dimensioned	2,4,5,6, 8,9,10,14, 18,22,26,30	Steep/shallow valley

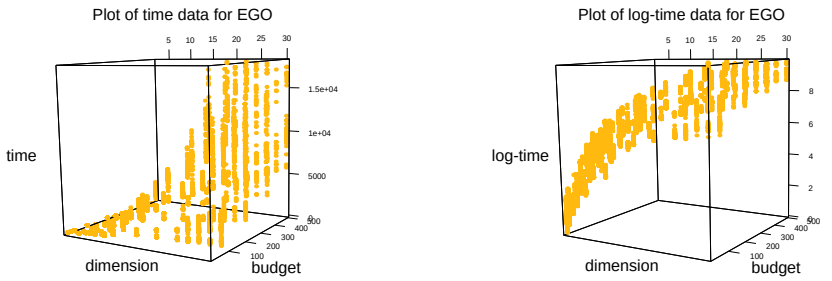

Figure 4.1: Plot of time used by the EGO algorithm as a function of dimension and function call budget (left). Plot of time used by the EGO algorithm after a log transformation (right).

15,000s. This is due to a 5 hour (18,000s) limit imposed on the algorithms throughout the experiment. As dimension and budget increase the spread in the data also increases. This is not a desirable quality for linear regression since it violates one of the assumptions of the process, namely that the variance is constant. A log transformation of the data may lead to a dataset with constant variance. Indeed, this appears to be the case as can be seen in the right image of Figure 4.1. This plot seems to indicate a logarithmic growth in both d and n. Therefore linear regression is performed with the model

$$\log(t_{EGO}) = \beta_0 + \beta_1 \log(d) + \beta_2 \log(n) + \beta_3 \log(d) \log(n), \tag{4.6}$$

where $\beta_i \in \mathbb{R}$ are coefficients to be determined by regression for each $i = 0, 1, 2, 3$. The third term is an "interaction" term and its necessity will be determined by the statistical inference provided by the regression.

Figure 4.2 shows two diagnostic plots from the linear regression of EGO time.

On the left is a plot of the residuals against the fitted values. Ideally, the regression surface should pass through the "middle" of the data at all points and the plot should have close to half of the points above 0 and the rest of the points below 0. This is the case with this dataset. For larger fitted values, there is some deviation from this centrality and this is probably due to the sharp "cut-off" at 18,000s. The red curve is the centre of the residuals and in a good linear model should be constant and close to 0. In this particular plot, the red curve is satisfactory.

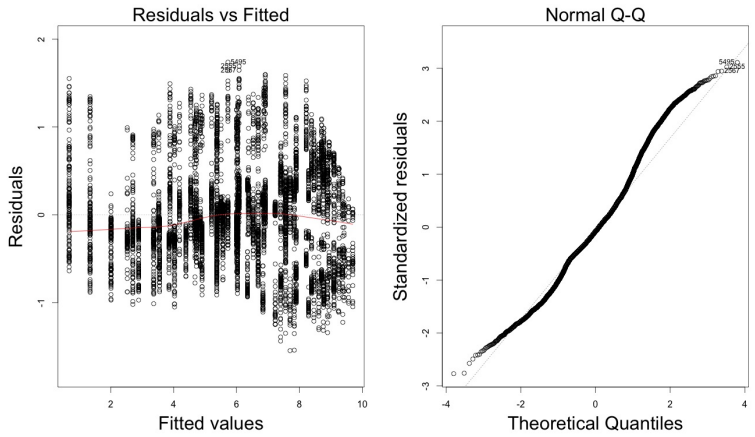

Figure 4.2: Residual plot (left) and Q-Q plot (right) from the linear regression of EGO time.

The plot on the right is a Normal Quantile-Quantile (Q-Q) plot. It is a plot of the standardized residuals against the quantiles of the standard normal distribution. In linear regression, it is assumed that the variance is normally distributed. The purpose of this plot is to verify this assumption. If the residuals are normally distributed about the regression surface then the Q-Q plot should be close to the identity line. In this instance the Q-Q plot does indeed adhere to the identity line and the normality assumption is validated. The normality assumption makes valid the use of the t and F statistics for inference. These diagnostic plots indicate that the model from (4.6) is worth considering.

Prior to analyzing the equation produced by linear regression, it should be noted that the F statistic for the model was smaller than 2E-16. This value indicates the probability of observing an F statistic greater than the one observed under the assumption that all of the coefficients from (4.6) are 0. In this instance the probability of such an event is significantly small bringing into question the assumption of all coefficients taking on a value of 0. In other words there is at least one valid explanatory term in the model (4.6). The least squares estimates of the coefficients from (4.6) are shown in Table 4.2.

The table also shows the p-values for the t statistic for each coefficient. This p-value can be interpreted as the probability of observing a t statistic greater than or equal to the one observed under the assumption that the corresponding coefficient has a value of 0. Note that the p-values for the

Table 4.2: Summary of linear regression results for the time function of the EGO algorithm.

	Estimate	Std. Error	t statistic	p value
β_0	-2.9001E+00	7.6448E-02	-3.7935E+01	0
β_1	1.9749E+00	3.9860E-02	4.9547E+01	0
β_2	9.6312E-01	1.6824E-02	5.7246E+01	0
β_3	-4.9195E-03	8.1360E-03	-6.0466E-01	0.5454

first three coefficients are extremely small, indicating, very strongly that the coefficients for these terms are unlikely to be 0. The coefficient β_3 on the other hand is simply not small enough to support the inclusion of the interaction term $\log(d)\log(n)$. This term can therefore be removed from the model and linear regression performed again, with the results found in Table 4.3.

Table 4.3: Summary of linear regression results for the time function of the EGO algorithm after removal of the interaction term.

	Estimate	Std. Error	t statistic	p value
β_0	-2.8585E+00	3.3279E-02	-8.5894E+01	0
β_1	1.9515E+00	9.1106E-03	2.1420E+02	0
β_2	9.5401E-01	7.4847E-03	1.2746E+02	0

This time each of the coefficients is reported as 0. Therefore there is no reason to believe that any of the remaining terms have no effect on the response variable.

Using the results from linear regression, an estimate of t_{EGO} is

$$\log\left(\hat{t}_{EGO}(d,n)\right) = -2.859 + 1.952\log(d) + 0.954\log(n),$$

which can be simplified to

$$\hat{t}_{EGO}(d,n) = \exp\left(-2.859 + 1.952\log(d) + 0.954\log(n)\right). \tag{4.7}$$

Figure 4.3 shows a plot of the regression surface and the data.

The regression was performed using only 15 of the 20 random restarts for each algorithm and function. These 15 points are referred to as the *training* data, since they are used to create the model function. Data from the remaining 5 runs is used to assess the quality of the prediction from the linear regression models. This subset of the data is referred to as the *test*

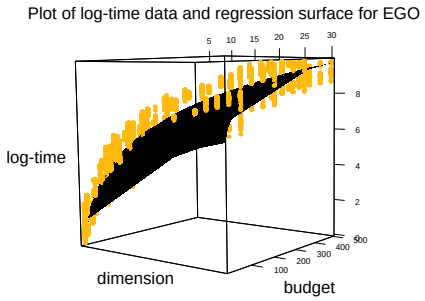

Figure 4.3: Plot of the log transformed time data and the regression surface for the EGO algorithm.

data, since it is used to test the model developed by the training data. This process is called *model-validation*. The idea is that a regression function is designed to pass through the "middle" of the data. The data obtained in this section contains multiple time values for each (d, n) pair that was tested. A good regression function might be expected to lie near the mean of the data at each tested point (d, n). Now consider the test data which was not used to develop the regression model. Two qualities are desirable when assessing the model using test data. First, it is desirable that its values are not very different from the training data. This implies a well-behaved dataset. Secondly, the test data should not be very different from the regression function. These two ideas can be combined into a model-validation criterion. Before this criterion is presented, however, some notation is required. Let $\mathcal{P}$ be the set of elements (d, n) such that a test problem for the pair (d, n) is used in this thesis. Let $N_{\mathcal{P}}$ denote the number of elements in $\mathcal{P}$. Then the *normalized error (NE)* is

$$NE(\hat{t}_{\mathcal{A}}) := \frac{\sum_{i=1}^{N_{\mathcal{P}}} \sum_{j=1}^{N_i} \left(t^{test}_{\mathcal{A}}(d_i, n_i) - \hat{t}_{\mathcal{A}}(d_i, n_i)\right)^2}{\sum_{i=1}^{N_{\mathcal{P}}} \sum_{j=1}^{N_i} \left(t^{test}_{\mathcal{A}}(d_i, n_i) - \bar{t}_{\mathcal{A}}(d_i, n_i)\right)^2}, \tag{4.8}$$

where $t^{test}_{\mathcal{A}}(d_i, n_i)$ is a time value for a test problem with dimension d_i and budget n_i, $\hat{t}_{\mathcal{A}}(d_i, n_i)$ is the estimated time value at (d_i, n_i) based on the training data, $\bar{t}_{\mathcal{A}}(d_i, n_i)$ is the average time value over all training data for the specific $(d_i, n_i$ pair, and N_i is the number of observations at (d_i, n_i) in the test data. For NE, a value near 1 indicates that the sum of the squared

residuals from the test data is similar to the sum of the squared distances from the test data to the average values of the training data.

For the EGO algorithm time function the NE is 1.18. This value is not too different from 1, however, it indicates that the sum of the distances squared from the test data to the estimated function is greater than the sum of the distances squared from the test data to the average values of the training data.

The process used to produce (4.7) was performed on the remaining five algorithms with the resulting equations, including that of EGO, shown in Table 4.4.

$$\hat{t}_{GA}(d,n) = \exp\left(-3.474 + (8.254\text{E-}2)\log(d)\log(n)\right)$$

$$\hat{t}_{NM}(d,n) = \exp(-6.411 - 0.521\log(d) + 0.503\log(n) + \\ +0.178\log(d)\log(n))$$

$$\hat{t}_{sNOMADr}(d,n) = \exp(-6.345 - (8.906\text{E-}2)\log(d) + 1.009\log(n) + \\ +0.161\log(d)\log(n))$$

$$\hat{t}_{EGO}(d,n) = \exp\left(-2.859 + 1.952\log(d) + 0.954\log(n)\right)$$

$$\hat{t}_{KGCP}(d,n) = \exp(-4.900 - 0.103\log(d) + 2.453\log(n) + \\ +0.130\log(d)\log(n))$$

Table 4.4: Estimated $t_{\mathcal{A}}$ functions for each of the five algorithms.

The regression plots in Figure 4.4 show an increase in $\log(t_{\mathcal{A}})$ with respect to both d and n for each algorithm. Notable differences do exist however in the amount of increase exhibited. The genetic and NM algorithms require the least amount of time, while EGO and KGCP require substantially more time as dimension and function call budget increase. The time required by sNOMADr is somewhere in between these two groups.

Table 4.5 shows the NE and three measures of variance for each algorithm.

To analyze the variance present in the time data for each algorithm three measures of variance are used. The first is the sample variance of the entire

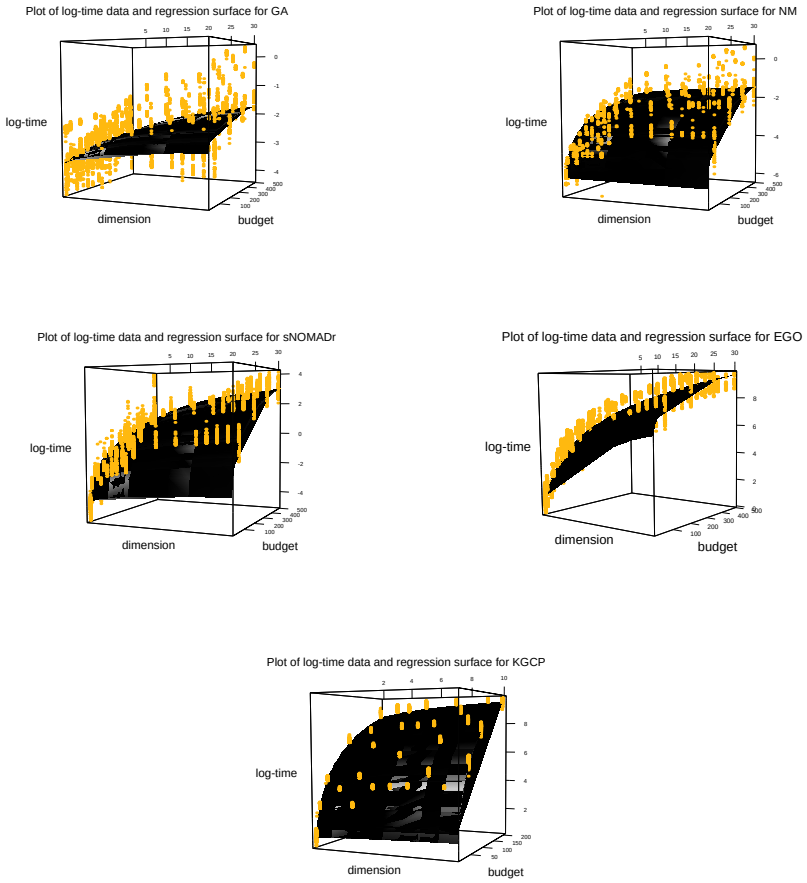

Figure 4.4: Plots of the regression functions for the algorithm time of GA (top left), NM (top right), sNOMADr (middle left), EGO (middle right), and KGCP (bottom).

Table 4.5: The NE value for each $\hat{t}_{\mathcal{A}}$ as well as the sample variance, average variance over all (d, n) pairs, and the average variance over all (d, n, f) triples.

Algorithm	NE	S^2	$S^2_{d,n}$	$S^2_{d,n,f}$
GA	1.71	0.449	0.155	0.0351
NM	1.39	1.46	0.292	0.0558
sNOMADr	1.36	3.14	0.156	0.0694
EGO	1.18	5.16	0.260	0.172
KGCP	1.11	5.70	0.0835	0.0688

dataset,

$$S^2 := \frac{\sum_{i=1}^{N_{\mathcal{P}}} \sum_{j=1}^{N_i} (t_{\mathcal{A},j}(d_i, n_i) - \bar{t}_{\mathcal{A}})^2}{\sum_{i=1}^{N_{\mathcal{P}}} N_i - 1}, \tag{4.9}$$

where $\bar{t}_{\mathcal{A}}$ is the average observed time for algorithm $\mathcal{A}$ over all test problems and reruns. This measure of variance is a reflection of the range of time data for a specific algorithm. The S^2 column in Table 4.5 contains the sample variance for each algorithm. Larger variance is exhibited by the more expensive algorithms such as EGO and KGCP. The fastest of the algorithms, GA, has a small variance.

The second measure of variance is

$$S^2_{d,n} := \frac{1}{N_{\mathcal{P}}} \sum_{i=1}^{N_{\mathcal{P}}} \frac{\sum_{j=1}^{N_i} (t_{\mathcal{A},j}(d_i, n_i) - \bar{t}_{\mathcal{A}}(d_i, n_i))^2}{N_i - 1}, \tag{4.10}$$

where $\bar{t}_{\mathcal{A}}(d_i, n_i)$ is the average over all time observations for algorithm $\mathcal{A}$ and a specific (d, n) pair. Equation (4.10) measures the average variability over all (d, n) pairs. If the difference between S^2 and $S^2_{d,n}$ is small then the time data is likely close to constant. This is the case for the GA algorithm where the ratio $S^2/S^2_{d,n}$ is close to 3. The plot of the time used by the GA algorithm in Figure 4.4 demonstrates the near constant nature of the time used by the algorithm as a function of d and n. For the KGCP on the other hand the ratio $S^2/S^2_{d,n}$ is close to 68, thus indicating that the time used by the KGCP is far from constant with respect to d and n. Smaller values of $S^2_{d,n}$ indicate less variability at a given (d, n) pair which in turn implies that other variables such as function type and starting point have a smaller effect on the time used by the algorithm. By contrast, larger values of $S^2_{d,n}$ indicate larger variability at a given (d, n) pair and this in turn implies that other variables may have a significant effect on the time used by the algorithm.

The KGCP algorithm exhibits the smallest $S^2_{d,n}$ value of all the algorithms and is therefore least affected by variables other than d and n. On the other hand, the EGO and NM algorithms exhibit the largest $S^2_{d,n}$ and their time therefore is most affected by other variables.

The final measure of variance is the average sample variance over all 135 test problems,

$$S^2_{d,n,f} := \frac{1}{135}\sum_{i=1}^{135}\frac{\sum_{j=1}^{20}(t_{\mathcal{A},j}(d_i,n_i)-\bar{t}_{\mathcal{A}}(d_i,n_i))^2}{20-1}, \tag{4.11}$$

where there are 20 observations for each test problem and $\bar{t}_{\mathcal{A}}(d_i,n_i)$ is the average observed time over those observations. This variance takes into consideration not only the dimension and function call budget but also the specific function being optimized. The greatest variability according to this measure can be found for the EGO algorithm. This means that even if the exact same function is optimized multiple times, the observed time will vary more than for the other algorithms. The ratio $S^2_{d,n}/S^2_{d,n,f}$ can be loosely be interpreted as the amount by which the time used by an algorithm is affected by the type of function being optimized. For the GA and NM algorithms this ratio is greater than 4 and certainly greater than that of the other algorithms. For EGO and KGCP this ratio is less than 1.5 indicating that the function being optimized has a minimal effect on the time used by the algorithms. The larger values for GA and NM indicate a greater effect of function type on the time used by the algorithms.

Overall, the time data indicates that the BGO algorithms require more time for optimization but are less affected by function type. The GA and NM algorithms can perform optimization much more quickly but are also more affected by function type. It should be noted that the KGCP algorithm was not able to complete optimization for some higher budgets and dimensions and as such the time data used to develop the regression functions uses a subset of the problems from Table 4.1.

4.2.2 Improvement Functions

As was the case in the time function regression, the complete regression process will only be shown for one algorithm, NM, while only the results will be shown for the remaining algorithms. Figure 4.5 shows a three dimensional plot of the improvement data for the NM algorithm.

The majority of the data is positive indicating that the NM algorithm outperformed the random sample almost every time. The amount by which

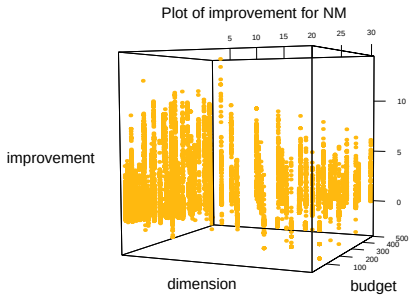

Figure 4.5: Plot of improvement as a function of d and n for the NM algorithm.

NM outperforms the random sample, however, varies quite a bit. NM can in some instances provide as many as 10 more digits of accuracy than a random sample but it can also provide no more digits of accuracy than a random sample for the same dimension and function call budget. Thus it is clear that the performance of NM varies due to factors other than d and n.

There does appear to be a trend, in that there is a greater difference in the performance of NM and the random sample at smaller dimensions and larger budgets. The non-constant variance present in this dataset poses a challenge for linear regression. It is also difficult to determine the type of relationship between the response and explanatory variables. In such an instance it is appropriate to view the diagnostic plots for several models and select the one that provides the best diagnostics. The model,

$$I_{NM}(d, n) = \beta_0 + \beta_1 \log(d), \tag{4.12}$$

provides good diagnostic plots which can be seen in Figure 4.6.

The plot of residuals against fitted values indicates that the regression surface lies near the middle of the data. At two locations, $d = 24$ and $d = 28$, the variance is notably smaller than anywhere else in the dataset. This is because certain types of functions were not found in these dimensions. As it turns out the omitted functions provide greater variance to the data. This situation reinforces the idea that function type has an effect on improvement. Nonetheless, the residual plot does not indicate any violation of the assumptions necessary for linear regression. The Normal Q-Q plot, on the other hand, does indicate non-trivial deviation from normality. This can

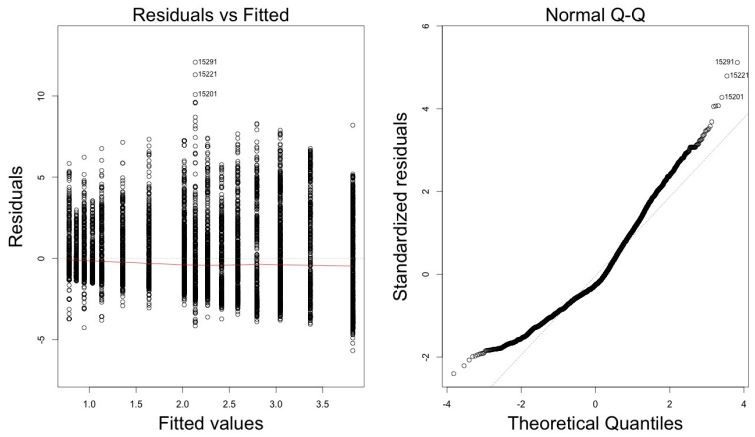

Figure 4.6: Residual plot (left) and Q-Q plot (right) from the linear regression of NM improvement.

pose a problem in linear regression, although it may not be very crippling in this instance. One reason is that the large datasets such as this one are robust to small deviations from normality. The second reason is that a violation of the normality assumption in linear regression impedes the statistical inference based on the t and F statistics, but it does not affect the prediction of the regression model. With these remarks in mind, the linear regression process continues with a summary of the results found in Table 4.6.

Table 4.6: Linear regression results for the improvement of the NM algorithm.

	Estimate	Std. Error	t statistic	p value
β_0	4.6000E+00	7.4084E-02	6.2091E+01	0
β_1	-1.1221E+00	3.1687E-02	-3.5411E+01	0

The p-values from the t statistic and F statistic (less than 2E-16) strongly indicate that there is a significant relationship between the response and explanatory variables. However, the earlier discussion indicates that these values should not be trusted entirely. Taking the coefficients from Table 4.6 provides the estimate of the improvement function for the NM algorithm

$$\hat{I}_{NM}(d, n) = 4.600 - 1.122 \log(d). \tag{4.13}$$

A plot of the regression surface and the data can be seen in Figure 4.7.

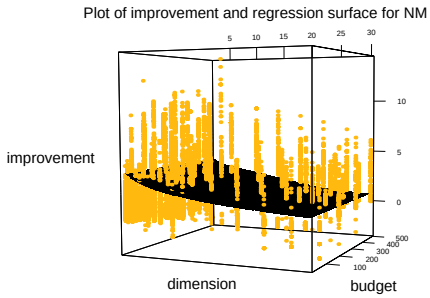

Figure 4.7: Plot of improvement data and regression surface for the NM algorithm.

The regression functions for each of the algorithms can be found in Table 4.7.

Table 4.7: Estimated I_a functions for each of the five algorithms.

$$\hat{I}_{GA}(d,n) = 1.389 - (1.543\text{E-}2)d + (1.397\text{E-}3)n$$

$$\hat{I}_{NM}(d,n) = 4.600 - 1.122\log(d)$$

$$\hat{I}_{sNOMADr}(d,n) = -9.980 + 0.280d + (7.061\text{E-}3)n + 2.664\log(d) + \\ +4.352\log(n) - 1.772\log(d)\log(n)$$

$$\hat{I}_{EGO}(d,n) = 0.976 + (3.396\text{E-}2)d$$

$$\hat{I}_{KGCP}(d,n) = 2.687 - 0.450\log(d) + (4.100\text{E-}3)n$$

Note that for the NM and EGO algorithms, the regression functions do not indicate that the function call budget has an effect on improvement. This may appear to be somewhat puzzling, however, recall that improvement is normalized by the random sampling technique. Thus it is the amount by which NM and EGO outperform random sampling that is not expected to improve with increased function call budgets. The regression functions

for the GA, NM and KGCP algorithms indicate that an increase in input dimension leads to a decrease in improvement. This is not the case for the EGO algorithm where the estimated relationship between dimension and improvement is positive. This variation in type of relationships helps explain why it was challenging to develop theoretical relationships between improvement and d and n. The nature of the relationships appears to be specific to each algorithm. For the sNOMADr algorithm the relationships are not as clear due to the $\log(d)\log(n)$ term.

Plots of the improvement data with the regression functions for each algorithm can be seen in Figure 4.8.

Model validation for the improvement functions is performed using the the NE criterion from (4.8) adapted to the improvement function. The NE value for the model (4.13) is 1.42. This value is not significantly different from 1 indicating that the predictive power of the model $\hat{I}_{NM}$ is acceptable. Table 4.8 shows the NE value for each algorithm as well as the three measures of variance discussed in Section 4.2.1.

Table 4.8: The NE value for each $\hat{I}_{\mathcal{A}}$ as well as the sample variance, average variance over all (d, n) pairs, and the average variance over all (d, n, f) triples.

Algorithm	NE	S^2	$S^2_{d,n}$	$S^2_{d,n,f}$
GA	1.03	1.74	1.59	0.718
NM	1.42	6.50	3.80	1.54
sNOMADr	1.09	14.5	6.19	2.22
EGO	1.04	2.12	1.93	1.12
KGCP	1.11	2.8	2.17	0.702

With the exception of the NM algorithm, the NE for the model $\hat{I}_{\mathcal{A}}$ is very close to 1 for every algorithm. This is an indication of good predictive power for the GA, sNOMADr, EGO, and KGCP improvement models. For all of the algorithms, the lower bound on improvement is nearly the same. However, the best improvement values are quite different among the algorithms. The sNOMADr and NM algorithms in particular can obtain a very high value of improvement. The sample variance, S^2, is therefore quite large for sNOMADr and NM and much smaller for the GA, EGO and KGCP algorithms. The ratio $S^2/S^2_{d,n}$ is close to 1 for the GA, EGO, and KGCP algorithms implying that there is a more or less constant trend in improvement with respect to d and n for these algorithms. For the sNOMADr and NM algorithms the ratio is slightly larger and thus the relationship between

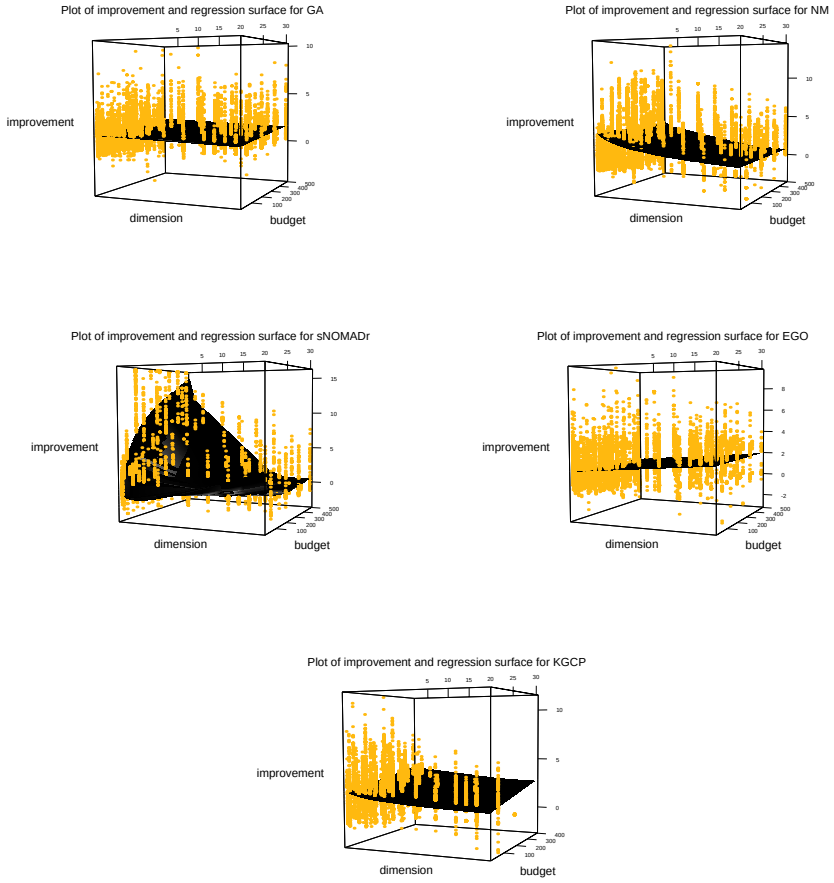

Figure 4.8: Plot of the regression surface for the improvement of GA (top left), NM (top right), sNOMADr (middle left), EGO (middle right), and KGCP (bottom).

improvement and d and n is less constant as can be seen in the images in Figure 4.8. The relatively large values of $S^2_{d,n}$ indicate the high variability in the improvement gained by each algorithm for a particular (d, n) pair. Such a high variability will certainly effect the validity of the regression function for improvement. The large variability also indicates that the improvement is likely affected by other variables. This variability is highest for the NM and sNOMADr algorithms. While the variability decreases when function type is considered in $S^2_{d,n,f}$, the values are still relatively large. The ratio $S^2_{d,n}/S^2_{d,n,f}$ is greatest for the KGCP algorithm implying that the specific function being optimized has a notable effect on the amount of improvement gained. This ratio is smallest for the EGO algorithm. The variance in improvement gained when optimizing a specific function is greatest for the sNOMADr algorithm and smallest for the GA and KGCP algorithms. The sNOMADr algorithm is inherently local and as such may converge to locally optimal points. This of course implies that starting point will have an effect on improvement. However, the use of VNS combats the local nature and may lead to larger values of improvement when the algorithm is able to avoid a locally optimal point. The combination of local and global search in this algorithm likely leads to the large variability in the improvement it can obtain. The large variability of the improvement in the NM algorithm even for a specific function can be attributed to the local nature of this algorithm and the effect of starting point. The EGO algorithm also exhibits a larger variance in improvement gained and this may be attributed to the quality of the GP model. If the GP model is accurate and the parameter estimates are accurate, the EGO algorithm may perform quite well, however, if the GP is not an adequate representation of the objective function, the EGO algorithm may perform poorly.

The large variance noted in the improvement plot of the NM algorithm is also seen in the other four algorithms. Several remarks can be made regarding the linear regression functions. The plot of the regression surface for GA is almost constant. This implies that the behaviour of GA is similar to that of the random sample which is intuitive since the genetic algorithm is essentially a "clever" random search. The plot for sNOMADr indicates a very notable difference between the improvement gained by sNOMADr and the improvement gained by random search. Out of all the algorithms sNOMADr is expected to provide the strongest performance for dimensions smaller than 5 and function call budgets greater than 400. The regression surface for KGCP indicates that at smaller dimensions and greater budgets, KGCP will perform much better than random sampling. There are also some

notable gaps in the data for the KGCP algorithm. At higher dimensions and larger budgets, KGCP was not able to complete optimizing in the allotted 10 hour time limit. As a result, the function $\hat{I}_{KGCP}$ should be used carefully in the region where $d > 14$ and $n > 200$. The plot for EGO shows a unique trend among this group of algorithms. The performance of EGO compared to that of random sampling is greater at *higher* dimensions.

In the following chapter, the regression functions developed in these sections will be used for the rubric developed in Section 4.1.

Chapter 5

Using the Rubric

Chapter 4 saw the development of a rubric to aid algorithm selection in black-box optimization. In Section 4.1 a framework for the rubric was developed and in Section 4.2 the time and improvement functions were estimated for each algorithm. The rubric is designed to be used in the following way. Starting with a time budget, T, the average running time of the objective function, t_f, and the input dimension d, the time equations from Table 4.4 can be used to obtain the function call budget, $n_{\mathcal{A}}$, for each of the 5 algorithms. Then the improvement functions from Table 4.7 can be used to compute the estimated improvement that can be obtained by each algorithm. Finally the algorithm with the largest estimated improvement is suggested for the optimization problem.

The purpose of this chapter is two-fold. First, the results of the rubric will be analyzed with the intent of answering the questions from Section 1.1. Secondly, the validity and accuracy of the rubric will be tested. In Section 4.2 several concerns were brought up regarding the estimated time and improvement functions. In both cases, there appears to be an amount of variance unaccounted for by the variables d and n. This is particularly noticeable in the improvement functions. For the improvement data, there is generally a large amount of variance. Both of these issues may cause the rubric to be less reliable.

Before these analyses are presented, an adjusted improvement criterion is required.

5.1 The Adjusted Improvement Criterion

Note that the function call budget may not be the same for each algorithm. For example, consider a scenario with the following information:

$$T = 3600s \quad t_f = 20s \quad d = 5.$$

Recall equation (4.2.1)

$$T = nt_f + t_{\mathcal{A}}(d, n).$$

Substituting the known information this becomes

$$3600 = 20n + t_{\mathcal{A}}(5, n).$$

For the GA algorithm this equation is

$$3600 = 20n_{GA} + \exp\left(-3.474 + (8.254\text{E-}2)\log(5)\log(n_{GA})\right)$$

and for the KGCP algorithm it is

$$\begin{aligned} 3600 = 20n_{KGCP} + \exp(&-4.900 - 0.103\log(5) + 2.453\log(n_{KGCP}) + \\ &+ 0.130\log(5)\log(n_{KGCP})). \end{aligned}$$

These equations can be solved and rounded down to obtain $n_{GA} \approx 179$ and $n_{KGCP} \approx 104$. The estimated improvement for these two algorithms, according to the equations in Table 4.7, is

$$\hat{I}_{GA}(5, 179) = 1.389 - (1.543\text{E-}2)5 + (1.397\text{E-}3)179 = 1.56$$

and

$$\hat{I}_{KGCP}(5, 104) = 2.687 - 0.450\log(5) + (4.100\text{E-}3)104 = 2.39.$$

It appears that the KGCP algorithm is preferred over the GA algorithm for this optimization scenario. However, recall that the improvement criterion from (4.4) is normalized by the random sample having the same budget. In this instance, GA is compared to a random sample with 179 points while the KGCP is compared to a random sample with 104 points. It is likely that the random sample with 179 points will outperform the random sample with 104 points. This will lead to the improvement criterion for GA to be smaller than it really is. What is needed is a comparison of each algorithm's improvement to a random sample with a fixed function call budget for the given optimization scenario. The random sample technique requires a very small amount of time to make decisions about which points to sample. It can, therefore, be assumed that $t_r(d, n) = 0$. Then the function call budget permitted for the random sample under a given time budget is

$$n_r = \left\lfloor \frac{T}{t_f} \right\rfloor.$$

The improvement for algorithm $\mathcal{A}$ after n sampled points, adjusted for the sampling budget of the random sampling technique, is

$$AI^n_{\mathcal{A}} = -\log_{10}\left(\frac{f(x^{n_{\mathcal{A}}}_{\text{best}}) - f^*}{f(x^{n_r}_r) - f^*}\right). \tag{5.1}$$

The improvement functions developed in the previous chapter now need to be adapted for this criterion. Equation (5.1) can be expressed in terms of the improvement criterion, $I^n_{\mathcal{A}}$, in the following way

$$\begin{aligned}
AI^n_{\mathcal{A}} &= -\log_{10}\left(\frac{f(x^{n_{\mathcal{A}}}_{\text{best}}) - f^*}{f(x^{n_r}_r) - f^*}\right) \\
&= -\log_{10}\left(\frac{f(x^{n_{\mathcal{A}}}_{\text{best}}) - f^*}{f(x^{n_r}_r) - f^*}\frac{f(x^{n_{\mathcal{A}}}_r) - f^*}{f(x^{n_{\mathcal{A}}}_r) - f^*}\right) \\
&= -\log_{10}\left(\frac{f(x^{n_{\mathcal{A}}}_{\text{best}}) - f^*}{f(x^{n_{\mathcal{A}}}_r) - f^*}\frac{f(x^{n_{\mathcal{A}}}_r) - f^*}{f(x^{n_r}_r) - f^*}\right) \\
&= -\log_{10}\left(\frac{f(x^{n_{\mathcal{A}}}_{\text{best}}) - f^*}{f(x^{n_{\mathcal{A}}}_r) - f^*}\right) - \log_{10}\left(\frac{f(x^{n_{\mathcal{A}}}_r) - f^*}{f(x^{n_r}_r) - f^*}\right) \\
&= I^n_{\mathcal{A}} - \log_{10}\left(\frac{f(x^{n_{\mathcal{A}}}_r) - f^*}{f(x^{n_r}_r) - f^*}\right).
\end{aligned}$$

The adjusted improvement can now be estimated as a function of d and n as

$$\hat{A}I_{\mathcal{A}}(d,n) = \hat{I}_{\mathcal{A}}(d,n) - \log_{10}\left(\frac{f(x^{n_{\mathcal{A}}}_r) - f^*}{f(x^{n_r}_r) - f^*}\right). \tag{5.2}$$

the first term of this expression is modelled in Section 4.2.2 and the equations $\hat{I}_{\mathcal{A}}(d,n)$ for each algorithm can be found in Table 4.7. The second term in this expression is unknown. The presence of f and f^* in this expression present a computational challenge. The approach used here is to assume that

$$\frac{f(x^{n_{\mathcal{A}}}_r) - f^*}{f(x^{n_r}_r) - f^*} \approx \frac{\|x^{n_{\mathcal{A}}}_r - x^*\|}{\|x^{n_r}_r - x^*\|}.$$

Substituting into the second term of (5.2) yields

$$-\log_{10}\left(\frac{\|x^{n_{\mathcal{A}}}_r - x^*\|}{\|x^{n_r}_r - x^*\|}\right).$$

The assumption made earlier allows the removal of function values from the above expression, thus simplifying the computation. Note that this new term is a measure of how much closer a random sample can get to x^* if $n_{\mathcal{A}}$ function calls are allowed as opposed to n_r function calls. Knowledge of x^* is not crucial here and can be replaced by a randomly selected point $\bar{x} \in X$. Then

$$-\log_{10}\left(\frac{\|x^{n_{\mathcal{A}}}_r - \bar{x}\|}{\|x^{n_r}_r - \bar{x}\|}\right)$$

can be quickly approximated using a *Monte Carlo* method. This method evaluates the above expression a large number of times randomly and then computes the average value. Thus (5.2) can be computed.

5.2 Case Study: Small Time Budgets

In the first case study a relatively small budget of $T = 3600s$ (1 hour) with $t_f \in \{5, 10, 20, 60\}$ and $d \in \{2, 8, 15, 25\}$. The results of the rubric can be seen in Table 5.1.

Table 5.1: Results of the rubric for a small time budget of 1 hour and varying t_f and d values. The best performing algorithm for each t_f, d pair can be seen in bold.

t_f (s)	d	GA		NM		sNOMADr		EGO		KGCP	
		$n_{\mathcal{A}}$	$AI_{\mathcal{A}}$	$n_{\mathcal{A}}$	$AI_{\mathcal{A}}$	$n_{\mathcal{A}}$	$AI_{\mathcal{A}}$	$n_{\mathcal{A}}$	$AI_{\mathcal{A}}$	$n_{\mathcal{A}}$	$AI_{\mathcal{A}}$
5	2	719	2.4	719	3.8	719	**18.1**	697	1	160	2.7
10	2	359	1.9	359	3.8	359	**13.3**	354	1	144	2.8
20	2	179	1.6	179	3.8	179	**9.9**	178	1.1	116	2.8
60	2	59	1.4	59	3.8	59	**5.6**	59	1.1	56	2.6
5	8	719	2.3	719	2.3	717	**7.2**	480	1.2	123	2.1
10	8	359	1.8	359	2.3	359	**4.3**	286	1.2	114	2.1
20	8	179	1.5	179	2.3	179	**2.5**	159	1.2	98	2.1
60	8	59	1.4	59	**2.3**	59	0.9	57	1.3	54	2
5	15	719	2.2	719	1.6	716	**3.6**	261	1.5	110	1.9
10	15	359	1.7	359	1.6	359	1.3	190	1.5	103	**1.8**
20	15	179	1.4	179	1.6	179	0.4	123	1.5	91	**1.8**
60	15	59	1.2	59	1.6	59	0	51	1.5	53	**1.7**
5	25	719	2	719	1	713	1.7	121	**1.8**	101	1.6
10	25	359	1.5	359	1	358	0.2	103	**1.8**	96	1.6
20	25	179	1.3	179	1	179	-0.2	79	**1.8**	85	1.6
60	25	59	1.1	59	1	59	0.5	41	**1.8**	52	1.4

For low dimensions, $d \leq 8$, the improvement gained by sNOMADr is expected to be significantly greater than that of the other algorithms. The difference in performance is more noticeabe at smaller dimensions and higher budgets. As dimension increases, the performance of sNOMADr drops quite drastically, so that at dimensions greater than 15, this algorithm is expected to provide less improvement than any of the other algorithms. For higher dimensions, $d \geq 15$, the BGO algorithms are expected to provide the most improvement. The difference between the two BGO algorithms is that the improvement of the EGO algorithm is expected to increase as d increases while for the KGCP algorithm the improvement is expected to decrease. Thus at dimension 25, the EGO algorithm outperforms KGCP, while at dimension 15, KGCP is the preferred algorithm according to the rubric. At lower dimensions, NM competes with sNOMADr with its expected improvement being greater than that of sNOMADr in one instance. The improvement expected to be gained by GA is never greater than that of the other

algorithms although it is a close competitor in some instances.

The claims of the rubric can now be tested against real data. The algorithms were used to optimize functions from two scenarios found in Table 5.1: $(t_f, d) = (20, 2)$ and $(t_f, d) = (60, 25)$. Each algorithm optimized three functions: Rastrigin, sum of powers, and Rosenbrock. Three repeats were used, thus creating nine data points for each algorithm under a specific optimization scenario. The results are used to test both the time and improvement predictions of the rubric. Figure 5.1 shows two box plots of the percent difference between the time required by each algorithm and the 3600s time budget.

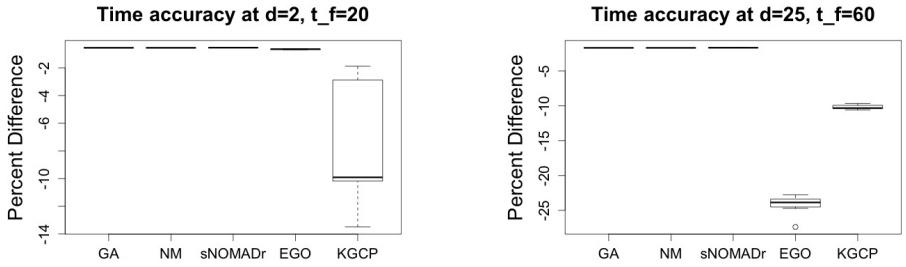

Figure 5.1: Box plots of the percent difference between the time used by each algorithm and the time budget of 3600s.

The time used by the GA, NM, and sNOMADr algorithms is within 1% of the time budget. For the KGCP algorithm, this difference is closer to 10% in both plots. For the EGO algorithm, the difference is less than 1% at dimension 2 but close to 25% at dimension 25. Being more expensive, the KGCP and EGO algorithms require more time for each iteration. Using up another iteration would likely take more time than is allotted and so the algorithms quit sooner. The time required by the EGO algorithm to complete an iteration increases with dimension. This explains the large difference between the time used by the EGO algorithm at dimension 2 and at dimension 25. Note that the test functions did not require 20 seconds to run. The values in the box plot were obtained by measuring the time required by the algorithm and then adding $20n_{\mathcal{A}}$ to each value. It should also be noted that the time budget was not exceeded by any one of the algorithms.

Figure 5.2 shows two box plots of improvement measured for each algorithm tested for functions where $(d, t_f) = (2, 20)$ and $(d, t_f) = (25, 60)$.

For the box plot on the left, the measured improvement reflects the

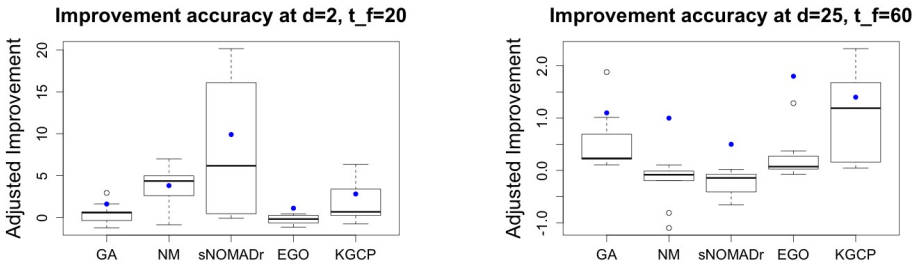

Figure 5.2: Box plots of the measured improvement gained by each algorithm and the predicted improvement shown in blue.

predictions of the rubric for the most part. One notable deviation is in the sNOMADr algorithm, where the measured improvement is smaller than that predicted by the rubric. The overall trend, however, is quite accurate. The median improvement values for NM and sNOMADr are very similar, but the higher upper bound for the sNOMADr algorithm supports the choice of the rubric. In the box plot on the left, the measured improvement for each of the algorithms was lower than that predicted by the rubric. The most notable difference is in the EGO algorithm, which was preferred by the rubric, but its measured improvement is certainly not greater than that of the other algorithms. The KGCP algorithm is the algorithm with the best measured improvement and the most accurately predicted by the rubric. Note that the dataset from which the time and improvement functions were estimated in Chapter 4 does not contain any functions with dimension 25 and function call budget as low as 59. Using the rubric in this instance is done with the understanding that extrapolation is being used. This may in part explain the disparity between the rubric and measured improvement.

5.3 Case Study: Large Time Budgets

In this section, a case study of optimization problems with large time budgets and objective functions with long running times is considered. The time budget allotted here is one week (604800s). Objective functions are assumed to have average running times between 20 minutes and 2 hours. The same dimensions considered in the previous sections are considered here as well. Table 5.2 contains the improvement and budget as predicted by the rubric.

The large time budget and average running time of the function make

Table 5.2: Results of the rubric for a large time budget of 1 week and varying t_f and d values. The best performing algorithm for each t_f, d pair can be seen in bold.

t_f (s)	d	GA		NM		sNOMADr		EGO		KGCP	
		$n_{\mathcal{A}}$	$AI_{\mathcal{A}}$	$n_{\mathcal{A}}$	$AI_{\mathcal{A}}$	$n_{\mathcal{A}}$	$AI_{\mathcal{A}}$	$n_{\mathcal{A}}$	$AI_{\mathcal{A}}$	$n_{\mathcal{A}}$	$AI_{\mathcal{A}}$
3000	2	201	1.6	201	3.8	201	**10.4**	201	1.1	199	3.2
5000	2	120	1.5	120	3.8	120	**8.2**	120	1	120	2.9
8000	2	75	1.5	75	3.8	75	**6.4**	75	1	75	2.7
10000	2	60	1.4	60	3.8	60	**5.6**	60	1	60	2.6
3000	8	201	1.5	201	2.3	201	**2.8**	201	1.2	197	2.6
5000	8	120	1.4	120	**2.3**	120	1.8	120	1.3	120	2.2
8000	8	75	1.4	75	**2.3**	75	1.2	75	1.2	75	2.1
10000	8	60	1.3	60	**2.3**	60	1	60	1.3	60	2
3000	15	201	1.4	201	1.6	201	0.5	201	1.5	196	**2.3**
5000	15	120	1.3	120	1.6	120	0.1	120	1.5	120	**2**
8000	15	75	1.3	75	1.6	75	0	75	1.5	75	**1.8**
10000	15	60	1.2	60	1.6	60	0	60	1.5	60	**1.7**
3000	25	201	1.3	201	1	201	-0.2	199	1.8	194	**2**
5000	25	120	1.2	120	1	120	0	120	**1.8**	119	1.7
8000	25	75	1.1	75	1	75	0.3	75	**1.8**	75	1.5
10000	25	60	1.1	60	1	60	0.5	60	**1.8**	60	1.5

trivial the time used by the algorithm. Thus all algorithms are more or less on the same playing field with regard to function call budgets. A general pattern can be found in the algorithms recommended by the rubric. For smaller dimensions, algorithms in the DFO category are expected to perform better, while for higher dimensions, algorithms from the BGO family are preferred. The trend in preferred algorithms is similar for large time budgets and small time budgets.

To test the rubric, a procedure identical to the one described for small time budgets in Section 5.2, was used. Figure 5.3 shows box plots of the percent difference between the time used by each algorithm and the time budget.

In the box plots, the time used by each of the algorithms differs from the time budget by approximately 1% or less. The effect of the expensive computations required by the BGO algorithms is less prevalent in these box plots than in Figure 5.1 due to the larger time budget. In the box plot on the right the time used by the BGO algorithms exceeds the time budget, albeit not by much.

Figure 5.4 shows a box plot of measured improvement for optimization problems with $(d, t_f) = (8, 3000)$ and $(d, t_f) = (15, 3000)$.

In both of these box plots, the measured improvement is lower than that

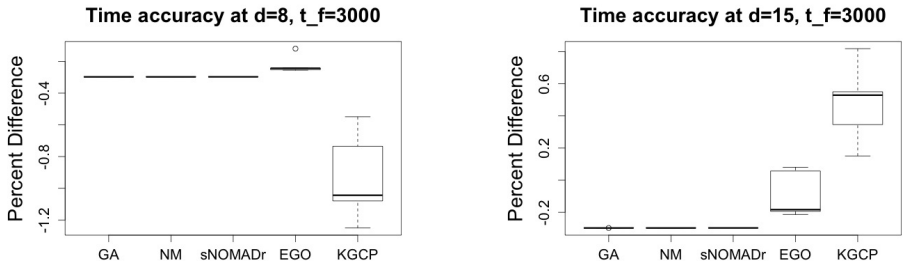

Figure 5.3: Box plots of the percent difference between the time used by each algorithm and the time budget of 604800s.

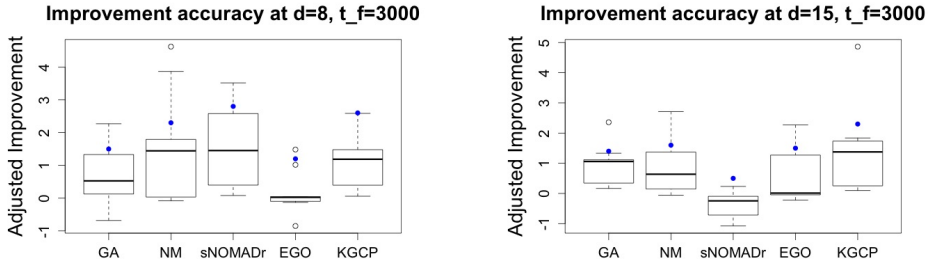

Figure 5.4: Box plots of the measured improvement gained by each algorithm and the predicted improvement shown in blue.

predicted by the rubric. A very noticeable difference is for the EGO algorithm in the left box plot, where the measured improvement is significantly smaller than the predicted improvement with very little variance. A clear-cut best algorithm cannot be determined from either of the box plots. There is a large amount of overlap in the results between the five algorithms.

With the exception of the NM algorithm in left box plot of Figure 5.1, the rubric predicted a larger improvement than was measured in the data. As was noted in Section 4.2.2, function type seems to affect improvement. This could explain the optimistic predictions of the rubric with respect to improvement. The functions used in the test could have properties that lead to below average measures of improvement.

5.4 Analysis of Rubric

Using the rubric for the two case studies in Sections 5.2 and 5.3 revealed some trends in preferred algorithms with respect to input dimension and function call budget. However, testing the claims of the rubric against real data revealed some concerns regarding its reliability. In this section, the trends and validity of the rubric will be discussed.

The basic trend seen in the rubric is that DFO algorithms are preferred at "lower" dimensions and BGO algorithms are preferred at "higher" dimensions. The definition of lower and higher dimensions is not exact based on the data from Table 5.1 and Table 5.2, however, it appears that the cut off is somewhere between $d = 8$ and $d = 15$. A possible conclusion is that dimension is not the only determining factor. Since in the tables, functions optimized at higher dimensions were not given an appropriately large budget. It may be that the ratio n/d is more important than the dimension.

In the tables of the previous sections, BGO algorithms were preferred when

$$\frac{n}{d} \in (2, 13).$$

Figure 5.5 shows box plots of measured improvement values from the dataset used in Section 4.2 for $d \in [2, 30]$ and $n \leq 10d$ for each of the algorithms.

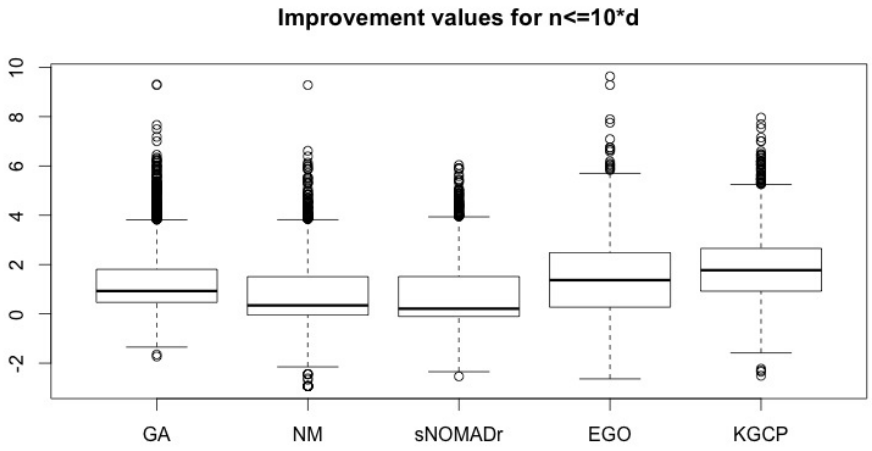

Figure 5.5: Box plot of the improvement gained by each of the algorithms for the dataset described in Chapter 4. Only data where $n \leq 10d$ are considered.

The median values for the BGO algorithms are greater than those of the competing algorithms, but the difference is not that significant, considering the large variances and the notable outliers. In Figure 5.6, box plots are shown for dimensions 10 and smaller.

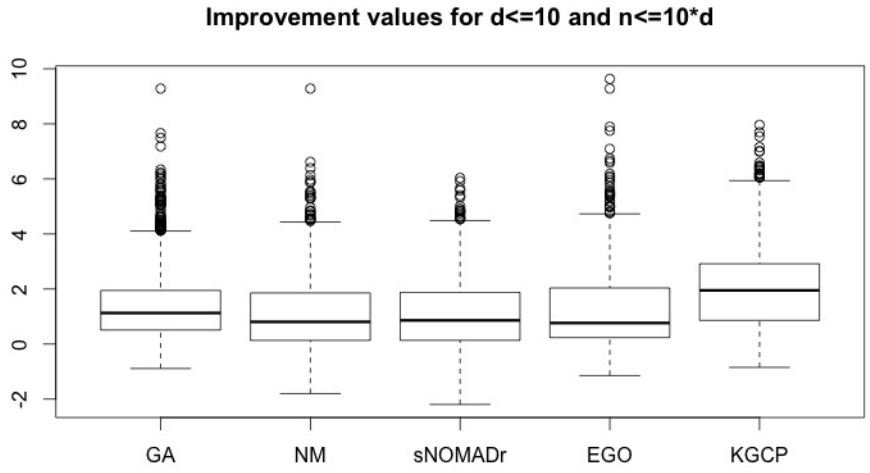

Figure 5.6: Box plot of the improvement gained by each of the algorithms on real functions where $d \leq 10$ and $n \leq 10d$.

The difference in median values for this subset of the data is even less significant. The median performance of KGCP is still slightly greater than for the other algorithms, but for EGO this is no longer true. Figure 5.7 shows box plots for dimensions 10 and greater.

For this subset of the data, the median performance of BGO algorithms is greater but, once again, the large variances may render the differences insignificant.

Regarding time, the amount required by BGO algorithms is far greater than that of the other algorithms. Therefore BGO algorithms are allowed a smaller function call budget when the running time of the objective function is small. However, if the average running time of the objective function is large, the function call budgets are similar for all of the algorithms. Thus, the disadvantage of fewer function calls faced by BGO algorithms is eliminated. In Table 5.1 and Table 5.2, when the time required to run the function is 60 seconds or greater, the difference in function call budgets is

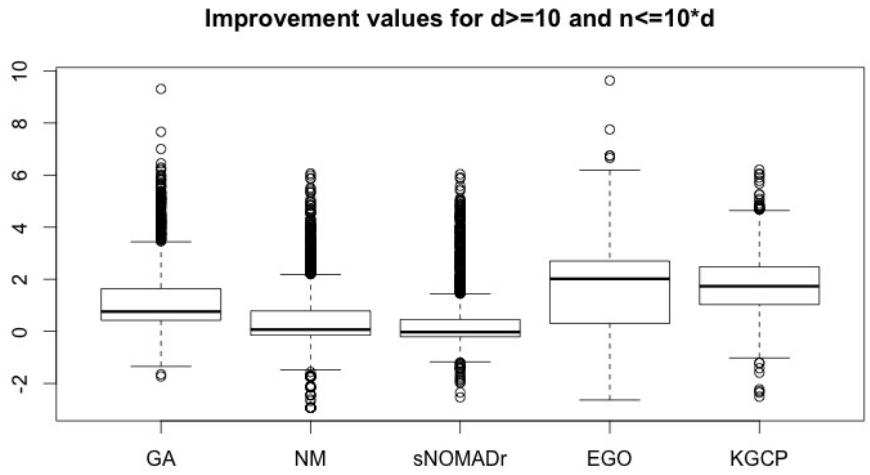

Figure 5.7: Box plot of the improvement gained by each of the algorithms on real functions where $d \geq 10$ and $n \leq 10d$.

minor. The difference in function call budgets is not always a limitation for the BGO algorithms since there are examples, in Table 5.1, where the improvement in KGCP is greater according to the rubric than the other algorithms in spite of a much lower function call budget.

Sections 5.2 and 5.3 present specific case studies and the discussion presented in this section is based on these case studies. While other case studies could be considered it is likely that the results will be similar to those observed in this chapter. Namely the time prediction will be quite accurate especially when T is large enough, the trends regarding the DFO and BGO improvements will be similar, and the large variance in improvement will be present.

Chapter 6

Conclusion

The rubric constructed in this thesis estimates the amount of function calls that can be obtained under a given time budget and it also predicts the improvement that could be obtained within the time budget. How accurate are these estimates and is this rubric reliable?

The assessment of time in Section 4.2.1 showed clear trends, strong relationships, and good fits for most of the algorithms. The case studies in Chapter 5 showed that, with respect to time, the rubric did not deviate much from the real data. These observations suggest that the predictions of function call budgets from the rubric are quite reliable. For the BGO algorithms at small time budgets it is inconclusive whether the prediction of function call budgets is reliable or not. The large difference between the time budget and the time used by the BGO algorithms could be due to the expensive nature of these algorithms or due to the poor estimates by the rubric.

With regard to improvement, the rubric is much less reliable. In Section 4.2.2 it was noted that the improvement data for each algorithm exhibited large variance and in Chapter 5, there was a notable difference between the improvement according to the rubric and the measured improvement. In some instances the algorithm suggested by the rubric was clearly outperformed by other algorithms in the measured improvement. The large variance present in the improvement data makes it difficult to determine the best choice for an algorithm. The rubric may be more reliable in instances where the difference in improvement values, as suggested by the rubric, is large.

The large variance present in the improvement values is likely due to factors other than those considered: d and n. In Section 4.2.1 and Section 4.2.2 it was shown that for both time and improvement, variance decreased when measured for data from specific functions. Function type is a factor not considered in this analysis, but appears to have a considerable effect on improvement and a lesser effect on time. Figure 6.1 shows the improvement values for the KGCP algorithm for a specific dimension and budget.

A distinct break in the data can be seen at a value of 1. This break

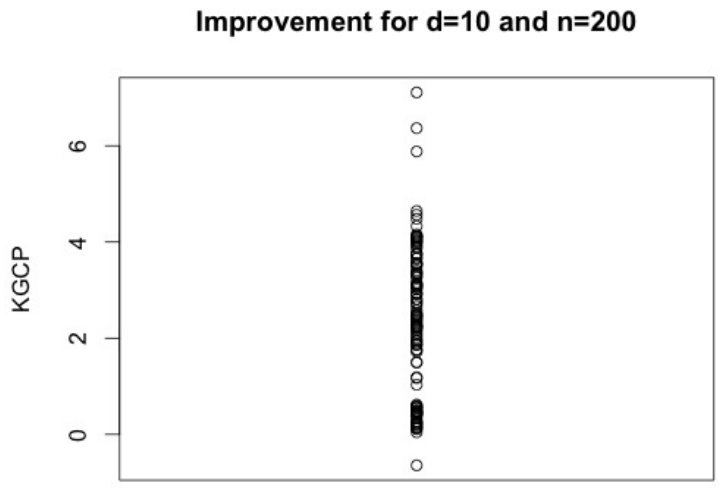

Figure 6.1: Box plot of the improvement gained by the KGCP algorithm for $d = 10$ and $n = 200$.

is determined solely by function type. Two functions that were more challenging for this solver reported improvement values below the break, while results for the remaining function types are found above the break. This phenomenon is not unique to the KGCP algorithm, dimension 10, or the function value budget of 200. Similar results can be found throughout the experimental data. In Table 4.8 the variance in improvement for the NM and sNOMADr algorithms was larger than for the other algorithms even when function type was taken into account. The improvement of these local optimization algorithms is likely affected by the starting point. The conclusion is that the variables d and n are insufficient for describing the improvement in function value gained by a particular algorithm.

This research was conducted using specific implementations of the algorithms from Chapter 3. For improvement, and more so for time, the results will almost surely differ with implementations. For example, the time used by KGCP and EGO should be comparable, theoretically, due to the use of the expensive GP in both algorithms. However, in practice, at higher dimensions and budgets the time required by the KGCP is almost twice as

large as that of EGO. This is likely due to approaches in implementation. Each of the five algorithms were used with more or less default parameters. An experienced user of each algorithm may have knowledge of preferred parameter settings and thus improve the performance of the algorithm. It is, however, unlikely that this improvement will be significant or affect the general trends observed in this thesis.

The data behind the results in this thesis considers a particular range of dimensions and function call budgets. The results of the rubric outside of this domain cannot be fully trusted. Although, large deviations from the observed trends are unlikely near the boundaries of the test domain. Farther away from the test domain, the results may not follow the relationships exhibited in this thesis.

The somewhat inconclusive results of this experiment suggest some directions for further work. An analysis where function type is considered may increase the accuracy of the rubric. Optimization problem classification tools have been available for some time in the Dr. Ampl meta solver, [FGK87], and could be used in a rubric of this sort. For optimizers regularly working on black-box functions in specific applications, such an analysis may be useful, since the black-box functions may be of a similar type.

In Chapter 5, it was noted that the large variance made it challenging to determine a clear-cut winner. An analysis taking into account worst case performance and best case performance may assist in "tie-breaking" scenarios.

Bibliography

[ABLD08] C. Audet, V. Bechard, and S. Le Digabel. Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. *Journal of Global Optimization*, 41:299–318, 2008. → pages 13

[AD06] C. Audet and Jr. J. E. Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, 17(1):188–217, 2006. → pages 13

[BBC+02] R. A. Berk, P. Bickel, K. Campbell, R. Fovell, S. Keller-McNulty, E. Kelly, R. Linn, B. Park, A. Perelson, N. Rouphail, et al. Workshop on statistical approaches for the evaluation of complex computer models. *Statistical Science*, pages 173–192, 2002. → pages 1

[Bul11] A. D. Bull. Convergence rates of efficient global optimization algorithms. *J. Mach. Learn. Res.*, 12:2879–2904, November 2011. → pages 15, 17

[CKAEY14] M. Campbell-Kelly, W. Aspray, N. Ensmenger, and J. R. Yost. *Computer: a history of the information machine*. Westview Press, Boulder, Colorado, 2014. → pages 1

[CMMY91] C. Currin, T. Mitchell, M. Morris, and D. Ylvisaker. Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association*, 86(416):953–963, 1991. → pages 6

[DD02] K. R. Davidson and A. P. Donsig. *Real analysis with real applications*. Prentice Hall, 2002. → pages 3

[DT91] Jr. J. E. Dennis and V. Torczon. Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4):448–474, 1991. → pages 12

[EAVH91] A.E. Eiben, E.H.L. Aarts, and K.M. Van Hee. Global convergence of genetic algorithms: A markov chain analysis. In H. P. Schwefel and R. Mnner, editors, *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 3–12. Springer Berlin Heidelberg, 1991. → pages 11

[Far02] J. J. Faraway. Practical regression and anova using r., 2002. → pages 23

[FGK87] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A mathematical programming language*. AT&T Bell Laboratories Murray Hill, NJ 07974, 1987. → pages 53

[FPD09] P. Frazier, W. Powell, and S. Dayanik. The knowledge-gradient policy for correlated normal beliefs. *INFORMS journal on Computing*, 21(4):599–613, 2009. → pages 15, 16

[GPR+13] D. Ginsbourger, V. Picheny, O. Roustant, C. Chevalier, and T. Wagner. Package 'diceoptim'. 2013. → pages 15, 17

[HJ61] R. Hooke and T. A. Jeeves. " direct search" solution of numerical and statistical problems. *Journal of the ACM*, 8(2):212–229, April 1961. → pages 12

[HM03] P. Hansen and N. Mladenovic. Variable neighbourhood search. *Handbook of Metaheuristics, Dordrecht, Kluwer Academic Publishers*, 2003. → pages 13

[Hol75] J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, Oxford, England, 1975. → pages 10, 11

[JSW98] 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. → pages 1, 2, 7, 14, 15

[Kel99] C. T. Kelley. *Iterative methods for optimization*, volume 18. Siam, 1999. → pages 12

[LRWE98] J. C. Lagarias, J. A. Reeds, M. H. Wright, and Wright P. E. Convergence properties of the nelder-mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9(1):112–147, 1998. → pages 12

[LSW09] J. L. Loeppky, J. Sacks, and W. J. Welch. Choosing the sample size of a computer experiment: A practical guide. *Technometrics*, 51(4):366–376, 2009. → pages 5

[McK98] K. I. M. McKinnon. Convergence of the nelder-mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158, 1998. → pages 12

[MGH81] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software (TOMS)*, 7(1):17–41, 1981. → pages 22

[MS05] M. Molga and C. Smutnicki. Test functions for optimization needs. 2005. → pages 22

[NM65] J.A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7(4):308–313, 1965. → pages 11

[RN14] J. S. Racine and Z. Nie. *crs: Categorical Regression Splines*, 2014. R package version 0.15-23. → pages 13

[Scr13] L. Scrucca. GA: A package for genetic algorithms in R. *Journal of Statistical Software*, 53(4):1–37, 2013. → pages 11

[SFP11] W. Scott, P. Frazier, and W. Powell. The correlated knowledge gradient for simulation optimization of continuous parameters using gaussian process regression. *SIAM Journal on Optimization*, 21(3):996–1026, 2011. → pages 2, 7, 15, 16

[SWN03] T. J. Santner, B. J. Williams, and W. Notz. *The design and analysis of computer experiments*. Springer Science & Business Media, 2003. → pages 1, 5, 7, 8

[Tor97] V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on optimization*, 7(1):1–25, 1997. → pages 13

[VB11] R. Varadhan and H. W. Borchers. Package 'dfoptim'. 2011. → pages 12

[Wri96] M. H. Wright. Direct search methods: Once scorned, now respectable. *Pitman Research Notes in Mathematics Series*, pages 191–208, 1996. → pages 12

[WS07] G. G. Wang and S. Shan. Review of metamodeling techniques in support of engineering design optimization. *Journal of Mechanical Design*, 129(4):370–380, 2007. → pages 2