

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING



**Bayesian Optimisation With
Dimension Scheduler:
Application to Microalgae Metabolism**

Author: Doniyor Ulmasov

Supervisors: Ruth Misener

Marc Deisenroth

Submitted in partial fulfilment of the requirements for the
MSc Degree in Computational Management Science
of Imperial College London

September 2015



This page intentionally left blank

ABSTRACT

Bayesian Optimisation (BO) is a data-efficient, global black-box optimisation method optimising an expensive-to-evaluate fitness function; BO uses Gaussian Processes (GPs) to describe a posterior distribution over fitness functions from available experiments. Similar to experimental design, an acquisition function is applied to the GP posterior distribution over fitness functions to suggest the next (optimal) experiment. Dynamic models of biological processes allow us to test biological hypotheses without running costly real-world experiments. But model construction requires estimating biological parameters (e.g. reaction rate kinetics) from costly experiments. BO efficiently estimates the parameters and thereby reduces the number of model simulations. We focus on parameter estimation for a dynamic model of microalgae metabolism [1]. In biological parameter estimation, Bayesian Optimisation (BO) is challenging because the parameters interact nonlinearly and the broad parameter bounds result in a huge search space. Due to the high problem dimensionality (in this context, 10 parameters), balancing exploration versus exploitation becomes more intricate and traditional Bayesian methods do not scale well. Therefore, we introduce a new Dimension Scheduling Algorithm (DSA) to deal with high dimensional models. The DSA optimises the fitness function only along a limited set of dimensions at each iteration. In each iteration, a random set of dimensions is selected to be optimised. This reduces the necessary computation time, and allows the dimension scheduling method to find good solutions faster than the traditional method. The increased computational speed stems from the reduced number of data points per each GP and the reduced input dimensions in the GP; GPs scale linearly in the number of dimensions but cubically in the number data points. Additionally, considering a limited number of dimensions at each node allows us to easily parallelise the algorithm.

Compared to commercial parameter estimation for biological models and a traditional Bayesian Optimisation algorithm, our approach achieves strong performance in significantly fewer experiments and a reduced computation time. We also design and provide a graphical user interface (GUI), which allows untrained users to optimise any model that can be invoked through a command line. The GUI is built on top of a modular Bayesian Optimisation library, `pybo` [2], which includes most common acquisition functions and kernels. The framework removes the barrier of programming language by providing the user with a straightforward user interface to set BO parameters, observe the optimisation as the code runs, and examine the GP after the experiment has been completed.



This page intentionally left blank

ACKNOWLEDGEMENT

I would like to thank my supervisors, Dr Ruth Misener and Dr Marc Deisenroth for their advices, interesting discussions and guidance provided throughout the project. Additionally, I would like to thank Dr Benoit Chachuat and Dr Caroline Baroukh for their collaboration and input provided for into the project.



This page intentionally left blank

TABLE OF CONTENTS

1	INTRODUCTION	1
1.1	MOTIVATION AND OBJECTIVE.....	1
1.2	CONTRIBUTION	1
1.3	REPORT OUTLINE.....	2
2	BACKGROUND AND RELATED WORK.....	3
2.1	ALGAE MODEL	3
2.2	BAYESIAN OPTIMIZATION.....	4
2.2.1	GAUSSIAN PROCESS.....	5
2.2.2	ACQUISITION FUNCTIONS	7
2.3	PYBO.....	8
2.4	RELATED WORK	9
3	EXPERIMENTS.....	10
3.1	INITIAL EXPERIMENTS.....	10
3.2	EFFECTS OF THE INPUT NORMALIZATION	13
3.3	LONG RUN TESTS	17
3.4	DIMENSION FIXATION, DIMENSION DETECTION.....	19
3.5	MISCELLANEOUS NOTES:.....	21
4	BAYESIAN OPTIMIZATION WITH DIMENSION SCHEDULING ALGORITHM.....	22
4.1	INTRODUCTION	22
4.2	ALGORITHM.....	23
4.3	EXAMPLE RUN.....	26
4.4	EXPERIMENTS SET-UP AND RESULTS.....	27
4.5	LIMITATIONS AND FURTHER IMPROVEMENT.....	29
4.6	CONCLUSION	30
5	BAYESIAN OPTIMIZATION FRAMEWORK.....	31
5.1	ANALYSIS.....	31
5.1.1	IMPLEMENTATION CHOICES.....	32
5.1.2	GRAPHICAL USER INTERFACE (GUI) LIBRARY.....	32
5.2	DESIGN AND IMPLEMENTATION.....	33
5.2.1	FRAMEWORK STRUCTURE:.....	33

5.2.2	USER EXPERIENCE (UX) DESIGN:.....	35
5.2.3	IMPLEMENTATION OPTIMIZATION	39
5.2.4	INSTALLATION.....	40
5.3	TESTING	41
5.3.1	CLI TESTING	41
5.3.2	GUI TESTING.....	43
5.3.3	OPERATING SYSTEMS.....	44
5.3.4	USER TESTING	44
5.4	EVALUATION AND FURTHER WORK.....	45
6	CONCLUSION.....	46
	APPENDIX.....	47
	A1: INITIAL EXPERIMENT RESULTS.....	47
	A2: FILE FORMAT SPECIFICATION	48
	A3: USER GUIDE.....	50
	BIBLIOGRAPHY.....	52

1 INTRODUCTION

1.1 MOTIVATION AND OBJECTIVE

Bayesian Optimisation (BO) is a data-efficient, global black-box optimisation method optimising an expensive-to-evaluate fitness function; BO tries to minimize number of queries made to the objective function by balancing exploration and exploitation over a surrogate function with a utility function [3]. As a result, the technique is commonly applied over the black-box function associated with high cost, computationally or/and physically. In our case, we diverge from the traditional use of BO and address the problem of computation time with many function observations at high dimensions. We will explore the problems faced with scaling BO in the third chapter, and present our solution to address the problem in the fourth chapter.

The technique has been applied to optimize neural network structures [4], control wind turbine's parameters to maximize the energy output [5] and material design[6]. We scope our problem over dynamic models of microalgae metabolism [1]. The goal is not only to find the maximum or minimum of a given model, but also reveal underlying features of the biological models, which could be extrapolated to other biological models or used to evaluate the given models.

1.2 CONTRIBUTION

We introduce a new Dimension Scheduling Algorithm (DSA) to deal with high dimensional models. The DSA optimises the fitness function only along a limited set of dimensions at each iteration. In each iteration, a random set of dimensions is selected to be optimised. This reduces the necessary computation time, and allows the dimension scheduling method to find good solutions faster than the traditional method. The increased computational speed stems from the reduced number of data points per each GP and the reduced input dimensions in the GP; GPs scale linearly in the number of dimensions but cubically in the number data points. Additionally, considering a limited number of dimensions at each node allows us to easily parallelise the algorithm. Compared to commercial parameter estimation for biological models and a traditional Bayesian Optimisation algorithm, our approach achieves strong performance in significantly fewer experiments and a reduced computation time. We also design and provide a graphical user interface (GUI), which allows untrained users to optimise any model that can be invoked through a command line. The GUI is built on top of a modular Bayesian Optimisation library, pybo [2], which includes most common acquisition functions and kernels. The framework removes the barrier of programming language by providing the user with a straightforward user interface to set BO parameters, observe the optimisation as the code runs, and examine the GP after the experiment has been completed.

1.3 REPORT OUTLINE

Here is the outline of the project:

- ❖ In Chapter 2 we provide brief overview of the biological model used in our work. We give provide an introduction into Bayesian Optimization, and identify all the main components of the technique. Followed by an analysis of latest Bayesian Optimization techniques and current challenges of the method.
- ❖ In Chapter 3 we outline all the experiments carried out, with outcomes of each experiment and further analysis of the results. The chapter includes sections on effects of normalization on the performance of the Bayesian Optimization library, with detailed analysis of the call stack to identify the bottleneck of the technique.
- ❖ In Chapter 4 we introduce the Bayesian Optimization with Dimension Scheduling Algorithm. We provide the pseudo-code of the algorithm, with experimental results of the algorithm on the models described in the second chapter. We discuss the limitations of the algorithm, follow with possible improvements of the algorithm.
- ❖ In Chapter 5, we cover the development cycle of the Bayesian Optimization Framework. We discuss the initial analysis of the problem area, followed with design, implementation and testing of the Framework.
- ❖ In Chapter 6, we summarize our work completed with a future outlook.

2 BACKGROUND AND RELATED WORK

We start by looking at specifics of the biological models, which we will be using throughout this work. We introduce Bayesian Optimization, and the related research completed in the field to deal with problems faced with the traditional Bayesian Optimization methods.

2.1 ALGAE MODEL

The thesis studies the applications of Bayesian Optimization methods on dynamic models of biological processes. Models of biological processes allow us to test hypotheses about the biological system without running costly real-world experiments. However, training the models requires to find parameters that explain behaviours of the true biological process, thus, requiring to conduct costly experiments at training time. To minimize the number of experiments, we use Bayesian optimization as a tool for data-efficient global black-box optimization.

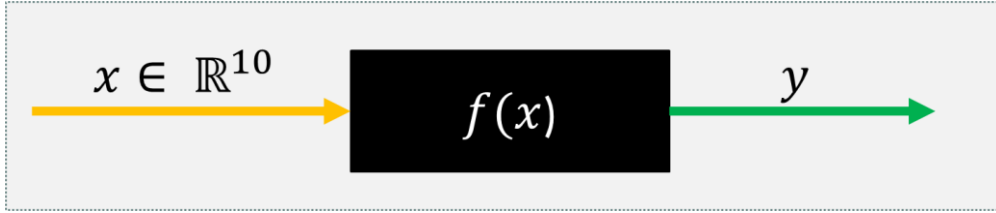


Figure 2.1.1: 10 dimensional black box function with an output of y

We focus on parameter estimation for a dynamic models of microalgae metabolism [1]; the forcing function is based on light exposure and nitrate input and experimental data which includes data collected for measurable outputs including: lipids, carbohydrates, carbon organic biomass, nitrogen organic biomass and chlorophyll [1]. The two models simulate Algae yield, with 10 input parameters with varied bounds; the bounds are shown on the right, Figure 2.1.2; and diagram of the black-box function is shown on top, Figure 2.1.1. The objective of the optimization is to minimize the objective value.

Bounds of the Models	
[0., 0.5]	[0., 10.]
[0., 400.]	[0., 100.]
[0., 25.]	[0., 100.]
[0., 100.]	[0., 1000.]
[0., 10.]	[0., 100.]
[0., 10000.]	[0., 10.]
[0., 1.]	[0., 10.]
[0., 1000.]	[0., 1000.]
[0., 10.]	[0., 10.]
[0., 25000.]	[0., 10.]
M1	M1SN

Figure 2.1.2 Bounds of the models

The dynamic models provided can be evaluated relatively quickly, but the size of the bounds and number of dimensions creates an enormous search area making random sampling methods futile. Commercial optimization software, gPROMS, have achieved lowest objective value of 22.45 after ≈ 24 hours of computational time. To achieve the objective value,

gPROMS requires an initial starting point, which does not guarantee an optimal result. The software may or may not provide with a result, nor does it report points of explorations. In other words, the only data the user receives back from the software is the best point achieved.

2.2 BAYESIAN OPTIMIZATION

Bayesian Optimization (BO) is a global black-box optimization technique and our objective is to find

$$\begin{aligned} \mathbf{x}^* &= \operatorname{argmax} f(\mathbf{x}) \\ \mathbf{x} &\in B, \text{ where } B \text{ is the feasible set, } B \subset \mathbb{R}^d \end{aligned}$$

The problem of minimization is addressed by transforming the function,

$$\begin{aligned} \mathbf{x}^* &= \operatorname{argmax} g(\mathbf{x}) \\ g(\mathbf{x}) &= -f(\mathbf{x}) \\ \mathbf{x} &\in B, \text{ where } B \text{ is the feasible set, } B \subset \mathbb{R}^d \end{aligned}$$

We also assume $f(\mathbf{x})$ is Lipschitz-continuous, i.e. there exists some constant C , usually unknown, such that for all $\mathbf{x}_1, \mathbf{x}_2 \in B$

$$\|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| \leq C \|\mathbf{x}_1 - \mathbf{x}_2\|$$

The black-box function does not yield us any information on convexity of the problem or the gradient at any given point. We are limited only to querying the function at point \mathbf{x} and receiving an output \mathbf{y} . We bound our problem with a hyperrectangle with d dimensions specified by the feasible set B .

The idea behind BO is to use the prior and the evidence collected from the observed points to maximize over the posterior to decide next point of evaluation; with each new step getting us closer to the global optimum of the objective function; thus reducing the total number of function queries required to achieve the optimal result. The general algorithm can be formulated the following way:

Algorithm 1: General Bayesian Optimization

- 1: Sample the function n times, and update the GP with the sampled data
 - 2: **while** termination condition is not met **do**:
 - 3: Find $\mathbf{x}_{n+1} = \operatorname{argmax} a(\mathbf{x}_{n+1}, \mathbf{GP}_n)$
 - 4: Sample the objective function $\mathbf{y}_{n+1} = f(\mathbf{x}_{n+1}) + \varepsilon_{n+1}$
 - 5: Update the GP with $(\mathbf{x}_{n+1}, \mathbf{y}_{n+1})$
 - 6: Increment n by 1
-

The first step is to generate a sample data to be able to calculate the posterior of the Gaussian Process (GP), alternately, we can bootstrap the GP with pre-generated data. We use GPs as a surrogate function, which we maximize with a utility function $a(\bullet)$. In literature the utility function is often referred as the acquisition function or the policy or infill function; we interchangeably use the terms acquisition function and policy to refer to the utility function in BO. Common acquisition functions are covered in the section 2.2.2, and GPs are discussed in the section 2.2.1. The objective function is evaluated with the next point of evaluation derived by maximizing the acquisition function. If our objective function is noisy, we assume the noise is Gaussian where $\varepsilon_{n+1} \sim \mathcal{N}(0, \sigma_{noise}^2)$. In our case, the dynamic biological models are noise-free. We update our GP with the new observation, and repeat the cycle till we meet the termination condition. The termination condition is arbitrary, more commonly the termination condition “while $n < t$ ” is used, where t is the total number of function observations we want and n is the current number of observations.

Alternative approaches to global optimizations have been studied before, but the methods require large amount of observations to achieve the optimal result [3]. The alternatives include: interval optimization, branch and bound methods, stochastic approximations and reinforcement learning [7]. In a scenario with a costly function evaluations, we may not be able to afford to sample the function as many times required by these alternative methods [8]. Even though we are working with a relatively cheap function evaluations, we are interested in applying BO to achieve optimal results as efficiently as possible.

2.2.1 GAUSSIAN PROCESS

The Gaussian Process (GP) form the important component of the BO, as previously mentioned we use GPs as a surrogate function in our BO algorithm. GPs are an extension of the multivariate Gaussian distribution, with an infinite dimension stochastic process for which any finite combination of dimensions will be a Gaussian distribution [3]. The marginalization property of the Gaussian distribution allows us to compute marginal and conditional probabilities in a closed form. A GP is fully defined by its mean and covariance functions:

$$f(x) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

The surrogate function is a probabilistic model of the function, where for any given \mathbf{x} point, the GP returns the mean and variance of a normal distribution over the probable values of $f(\bullet)$ at \mathbf{x} . The mean of the GP is commonly set to zero where $m(\mathbf{x}) = 0$. Alternatively, the mean function can be specified from the initial sampled data from the function space.

Kernels

The kernels of a GP incorporate smoothness assumptions made of the black function. The common kernels used in BO with GPs are the Matérn family kernels [9][10] and Squared Exponential kernel. The SE kernel is infinitely differentiable, therefore it makes the most naïve and smoothest assumptions of the underlying function. The Matérn kernels contains a smoothness parameters, which determines the differentiability assumptions of the function. The mathematic definition of the kernels is presented below.

$$\text{Squared Exponential Kernel: } k(x_i, x_j) = \exp(-\frac{1}{2} \|x_i - x_j\|^2)$$

$$\text{Matérn Kernel: } k(x_i, x_j) = \frac{1}{2^{\varsigma-1}\Gamma(\varsigma)} (2\sqrt{\varsigma}\|x_i - x_j\|^\varsigma) H_\varsigma(2\sqrt{\varsigma}\|x_i - x_j\|^\varsigma)$$

where $\Gamma(\cdot)$ and $H_\varsigma(\cdot)$ are the Gamma function and the Bessel function of order ς , and ς is the smoothness parameter [3]

Alternative kernels have been developed, and some for specific use cases, such as incorporating cost function into the kernel [4].

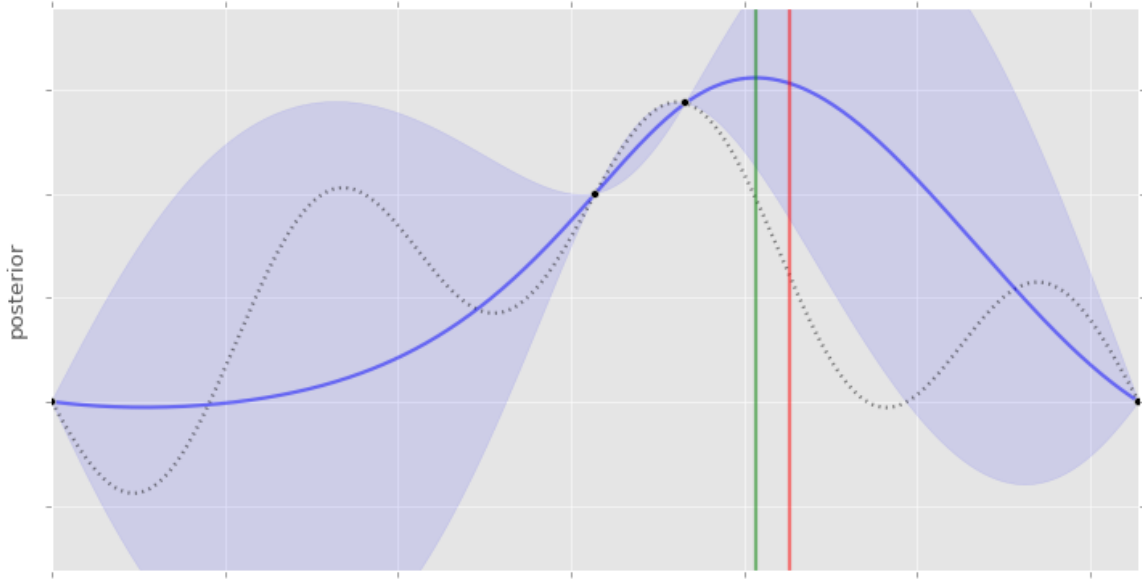


Figure 2.2.1 An example of GP with four data points. The shaded blue area represents the possible objective values at point x .

The diagram above provides an intuitive overview of the Gaussian Processes. The diagram shows the true function with a dotted line, and the mean of the GP with a solid blue line. The edges of the shaded area are the μ minus or plus the σ . For example, at a point indicated by a red line, we expect an objective value higher than our current best known value.

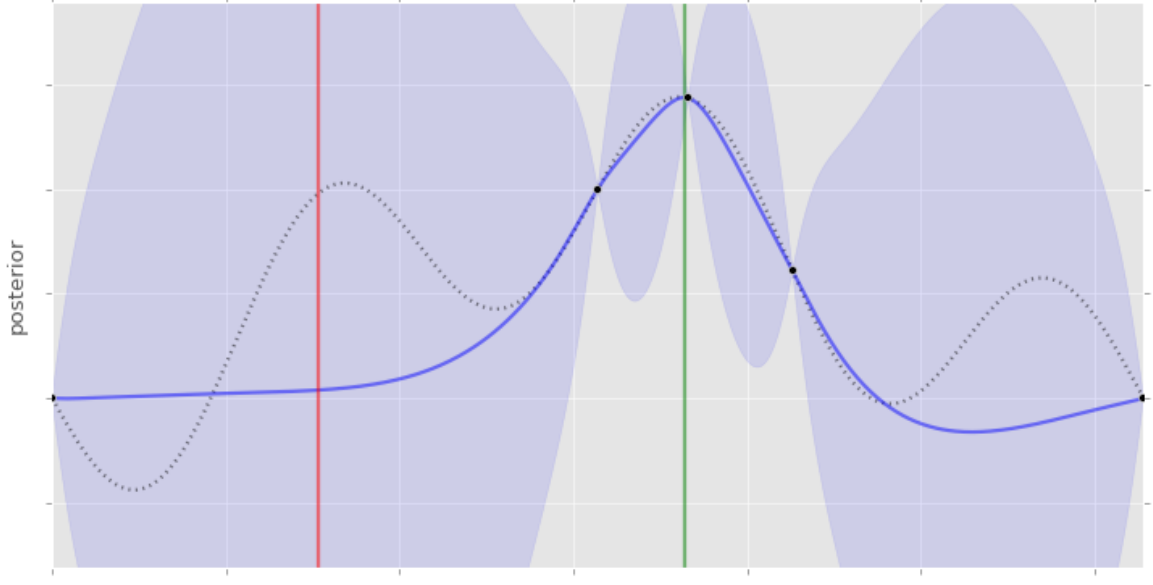


Figure 2.2.2 An example of GP with five data points. The shaded blue area represents the possible objective values at point x .

Upon evaluation of the point, we update our posterior to reflect our latest knowledge about the underlying true function. The red lines in the two diagrams are showing the next point of evaluation generated by the Expected Improvement policy specified below.

2.2.2 ACQUISITION FUNCTIONS

The acquisition functions are used to determine next point of evaluation based from the posterior of the GP. Therefore, each acquisition function uses the mean (μ) and variance (σ) derived from the GP. The three most common acquisition functions are the Probability Improvement, Expected Improvement and Upper Confidence Bound. The formulas for each acquisition function are presented below [3].

Probability Improvement:

$$\begin{aligned} PI(x) &= P(f(x) \geq f(x+) + \varepsilon) \\ &= \Phi\left(\frac{\mu(x) - f(x+) - \varepsilon}{\sigma(x)}\right) \end{aligned}$$

where $\Phi(\cdot)$ is the normal cumulative distribution function,

$$x+ = \arg \max x_i \in x_{1:t} f(x_i)$$

and ε is the trade-off parameter

Expected Improvement:

$$EI(x) \begin{cases} (\mu(x) - f(x+) - \varepsilon)\Phi(Z) + \sigma(x)\phi(Z) & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases}$$

$$\text{Where } Z = \begin{cases} \frac{\mu(x) - f(x) - \varepsilon}{\sigma(x)} & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases}$$

where $\varphi(\cdot)$ and $\Phi(\cdot)$ denote the PDF and CDF of the standard normal distribution respectively

Upper Confidence Bound

$$UCB(x) = \mu(x) + \beta\sigma(x)$$

Where β is the exploration vs exploitation parameter

Each acquisition function contains an exploration vs exploitation parameters. The parameter can be scheduled according to the current iteration or other arbitrary rules. There does not exist a specific rule of thumb with regards the choice of the acquisition function. Like the kernels, acquisition functions can be tailored to the problem area [11].

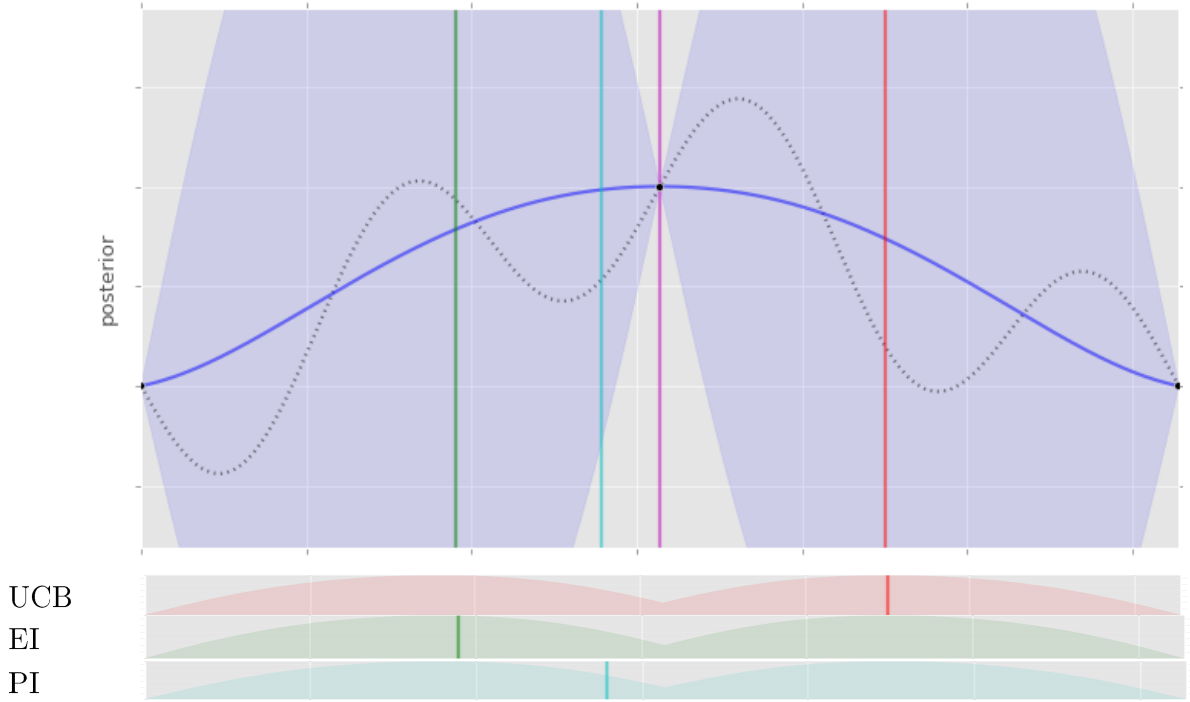


Figure 2.2.3 Next point of evaluation generated via different policies

The diagram above illustrates how the different policies pick a different point of evaluation depending on the posterior. In red it is the UCB, which is sampling along the highest variance area of the GP, similarly the EI in green, is sampling on the opposite side of the curve. On the other hand the PI policy, in cyan colour, is sampling closer to known points, where the variance is lower.

2.3 PYBO

The primary tool used throughout the thesis is a Python 2.7 based modular Bayesian optimization library pybo [2]. Pybo contains all the kernels and the policies outlined in the

previous section. Furthermore, the modular structure of the library allows for addition of custom components or modification of the Bayesian optimization algorithm without requiring changes in other components.

The default policies present in pybo, contain an exploration parameters and for the PI and EI policies and the hyper-parameters are integrated out. In UCB policy the exploration parameter is scheduled based on the current iteration value and the total iterations set for the algorithm. In addition to the policies from the section 2.2.2, we also have a choice of Thompson sampling policy. The Thompson policy uses a finite approximation to the kernel matrix with Fourier components [2].

In addition to the choice of kernels and policies, the library contains two different solvers for the acquisition functions. The two solvers are DIRECT solver and LBFGS solver, details of the solvers in covered in the Sections 3.2.

2.4 RELATED WORK

Increased usage of computational models, and machine learning in general has sparked the interest in Bayesian Optimization techniques. BO and different variants of the algorithm have been applied to robotics [12], adjust wind turbine parameters for maximum yield [5]. However, there have not been development in application of BO to dynamic models of biological processes. Traditionally, BO has been applied to functions with an expensive evaluation costs, which is not necessarily the case with our models.

In all applications, there is a common theme. The BO methods perform excellently when the number of dimensions is not high. The BO methods do not scale well, and there have been two interesting approaches tackling the problem of dimensionality.

The most recent development is the Additive BO method (ABO)[11]. ABO makes an assumption of the underlying black-box function. The function is additive, as shown below:

$$f(x) = f^1(x^1) + f^2(x^2) + \dots + f^M(x^M).$$

Under the additive function assumption the method divides the dimensions space into independent decompositions of dimensions where:

$$\begin{aligned} x^i \cap x^j &= \emptyset \\ \text{where } x^j &\in X^j = [0,1]d^j \\ &\text{are lowerdimensional components.} \end{aligned}$$

As a result, each dimension decomposition can be optimized separately. The overall benefit of the method is faster converges to an (near-) optimal objective value in less iteration compared to the traditional BO method and other variants of BO.

3 EXPERIMENTS

Throughout the work number of experiments were carried out to explore how Bayesian Optimization techniques perform on the provided Algae models (M1 and M1SN). Each experiment was carried out to observe different aspects of Bayesian Optimization. The following section explores effects of different kernels, policies, and normalization of the input, and dimension fixation.

3.1 INITIAL EXPERIMENTS

The first set of experiments performed aimed to observe: how normalization aids the performance of the Bayesian Optimization library; how policies effect the output of the functions; how kernels effect the output of the functions; combination of which kernels and policies provides us with lowest objective value?

For the experiment total of 4 models were tested. The models consisted of the Algae models from Chapter 2.1, M1 and M1SN, and versions of the same models with normalized input values, M1N and M1SNN respectively. The normalized models' bounds are scaled down between 0 and 1 for all 10 dimensions of the models. Normalization is a common technique to bring all the values under a common scale, for easier comparison of the parameter values.

The Bayesian Optimization parameters used for the experiment included; all the common kernels, 3 variants of the Matérn kernels and Squared Exponential kernel; and Expected Improvement (EI), Probability Improvement (PI), Thompson and Upper Confidence Bound (UCB) policies. Each run of the experiment tested a combination of a policy and a kernel for 100 iteration, with 30 random Sobol samples for initialization and LFBSG solver. The importance of the solver is discussed in the next section.

Each experiment recorded the objective value achieved at each iteration and the time (in seconds) between each iteration of the BO. Based on the output, we calculate the average objective value, median objective value and average computation time per iteration. Total of 64 experiments were carried out, and only a subset of the results is presented on the next page, Table 3.1.1 and Table 3.1.2, and in the Appendix A2 the full data summary table is presented. The experiment with the kernel Matérn 5 and Thompson policy on the M1N was prematurely terminated due to computation instability of the calculation. The covariance matrix calculated by from the data resulted in non-positive definite matrix resulting in a failure of Chomsky Decomposition. The issue could be fixed by increasing the noise parameter for the GP, but for consistency purpose were kept same for all experiments.

Results:

Total of 64 experiments runs were carried out, a subset of experiment results are presented in the tables below. The results of all 64 experiments can be found in the Appendix.

Table Keys:

A.R. = Average Result

A.T. = Average Iteration Time

B.R. = Best Objective Value Result

T. = Time per Iteration Bar Graph

M.R. = Median Result

Expected Improvement

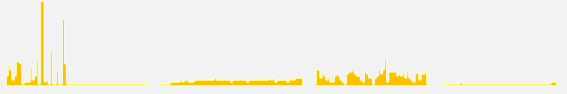
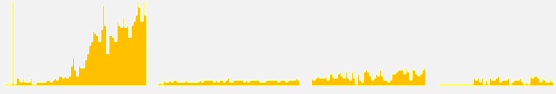
	MATERN1				MATERN3			
	M1	M1N	M1SN	M1SNN	M1	M1N	M1SN	M1SNN
A.R.	10547760.5	11503438.8	76768.24	1083.773	13751871.3	11583633.4	277110.7	94265.95
B.R.	201.537197	264.816842	98.08365	81.67155	171.243597	168.042621	63.50432	54.83634
M.R.	5753246.88	6170110.22	303.0726	164.9722	9175492.33	5967380.36	571.7213	627.2734
A.T.	5.70040353	3.10867061	5.865699	2.556584	14.6504596	3.0706828	5.107278	2.863471
T.								

Figure 3.1.1 Experiments with EI policy

Probability Improvement



	MATERN5				Squared Exponential			
	M1	M1N	M1SN	M1SNN	M1	M1N	M1SN	M1SNN
A.R.	13154416.3	3709609	298175.9	212514.1	8562599	1605591	203151.2	901.8844
B.R.	163.187297	73.05804	57.56085	32.70784	132.6882	71.41848	48.27869	81.67155
M.R.	9175492.33	10591.88	566.7448	197.3143	2319529	11523.23	485.5674	149.8152
A.T.	20.3816455	2.813308	4.576696	2.833679	13.48185	2.527455	6.475325	2.358551
T.								

Figure 3.1.2 Experiments with PI policy

The tables are colour coded, the green colour indicates good results relative to all other experiments, and red colour indicates the opposite. The best hyperparameter combination in terms of the objective value was the Matérn 5 kernel with Probability Improvement policy on M1SNN model. Each experiment ran for only 100 iterations, therefore the objective value is not reflective of the performance in the long run. At the same time, running 1000 iterations with all possible combinations is a computationally expensive and a time consuming process. Therefore, best performing hyper parameters are tested in the long run, and explored in the further section of this Chapter.

The significant finding in this set of experiments was the effects of the input normalization. A common pattern can be spotted by viewing the time per iteration graphs below the tables. The pattern is consistent, not only in the subset presented, but across all of the experiments. Experiments with normalized models on average required less to complete an iteration, and the growth of the iteration times is slower. The effects are more pronounced with the M1 model versus the M1N model. M1 model is significantly slower on average per iteration time, and each subsequent iteration increases in the computation time. The table below, presents an average of iteration times for across all experiments for all four models. The M1 model, on average is 6.25x times slower than a normalized variant. Similarly, the M1SN model is on average 1.96x times slower than a normalized variant. The effects of the normalization are explored in detail in the next section.

Average and Variance of the A.T. for all Experiments

	M1	M1N	M1SN	M1SNN
Average	17.8762	2.866584	5.144304	3.323937
Variance	147.524	0.162416	3.770371	1.921226

The time per iteration is bound to grow as we add observations to the Gaussian Process. The calculation of the posterior is a $O(n^3)$ computation [13], where n is number of observations. Therefore, the computation time per iteration increases regardless of the bounds of the model. In our case, the number of iterations is not high enough to reflect the consequences of the $O(n^3)$ computation on the normalized models. The performance degradation of the original models and detailed breakdown of the computation cost is presented in the next two experiment.

3.2 EFFECTS OF THE INPUT NORMALIZATION

The initial experiments revealed significant reduced computational time per iteration of the Bayesian Optimization when the models' inputs were normalized between 0 and 1. Further experiments were carried out to pin-point the cause of the performance differences between the normalized and original models, and performance comparison of two different solvers. The pybo library contains Direct solver and LBFGS solver. The same models from the previous section was used to perform the experiments: M1, M1SN, M1N, and M1SNN models. The policy and kernel were fixed to the Squared Exponential kernel and Expected Improvement policy. To ensure fairness of the experiments, all experiments used the same initial data. As a result, the only different variables in the experiments were the solvers and the models and normalization of the inputs.

To locate the cause of the performance differences, each run was profiled with cProfile [14] module from Python 2.7. The module records number of method calls made, and computational duration of each call. Total of 8 experiments were in each run, 4 with Direct solver and another 4 with the LBFGS solver. Total of were 6 runs were performed, and averaged results of the experiments are presented below.

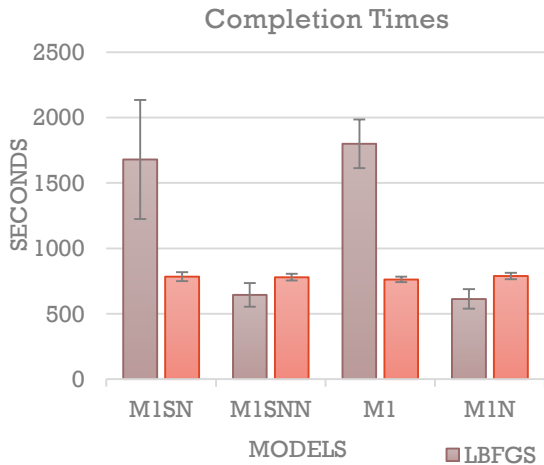


Figure 3.2.1 Total time (in seconds) passed for each run with different models and solvers with Standard Error bars

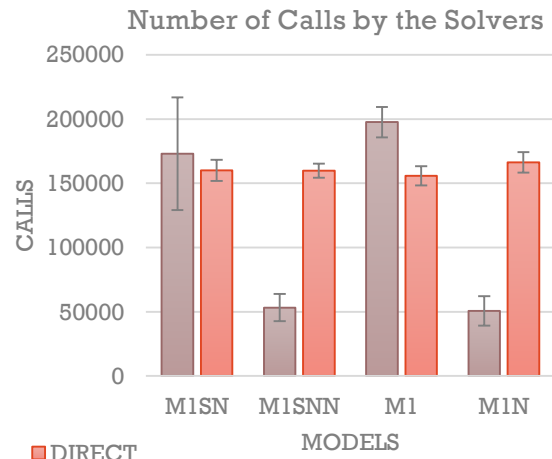


Figure 3.2.2 Total number of calls made by each experiment with different models and solvers with Standard Error bars

Note: The profiling module has a significant overhead, therefore these times are not representative of the actual performance times of the algorithm without the profiler.

The data from the profiling makes it clear, the gains in normalization are evident only when the LBFGS solver is used. The computation time for the LBFGS solver with the M1N model is on average 2.9x times shorter compared to the M1 model; similarly the M1SNN model is 2.60x times faster than the M1SN model with the LBFGS solver. The Bayesian Optimization library performs more calls with the LBFGS solver, thus increasing the total computation time. To narrow down the cause of the increased calls and times the profiling data was processed with a modified visualization

library SnakeViz [15]. The visual graphics generated by SnakeViz are presented in the Figure 3.2.3 for LBFGS solver and Figure 3.2.4 for the Direct Solver.

SnakeViz Icicle Graphs:

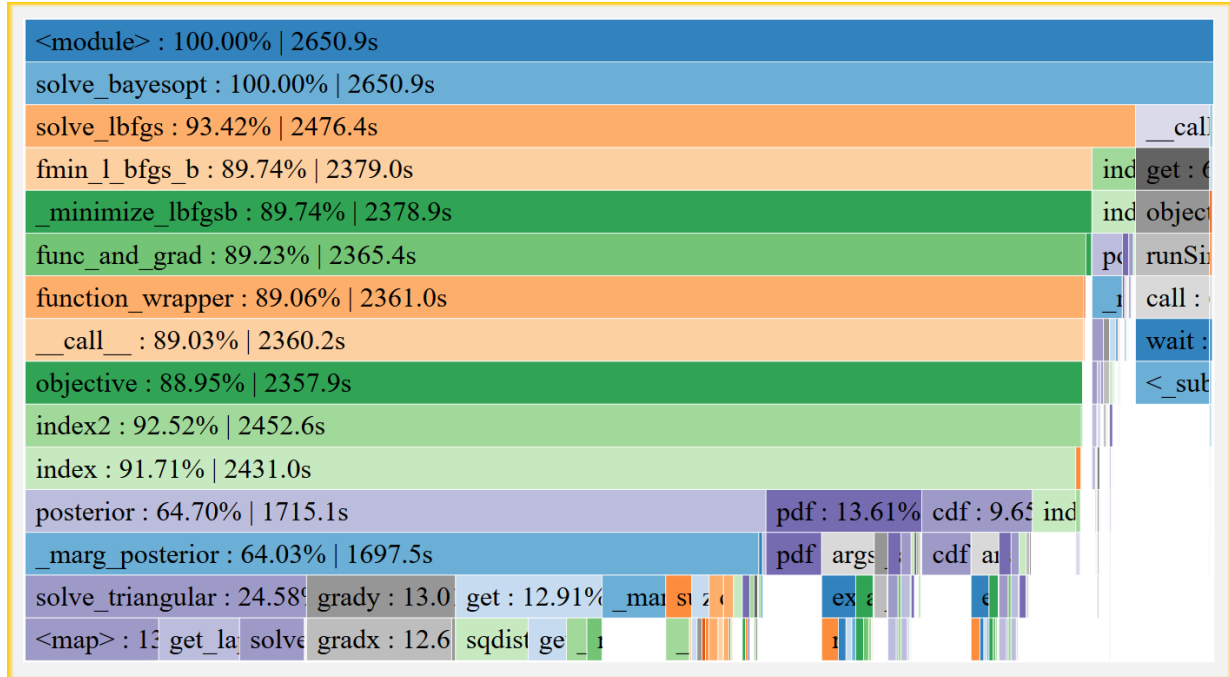


Figure 3.2.3 Icicle Graph of the Bayesian Optimization with the LBFGS solver and the M1 model

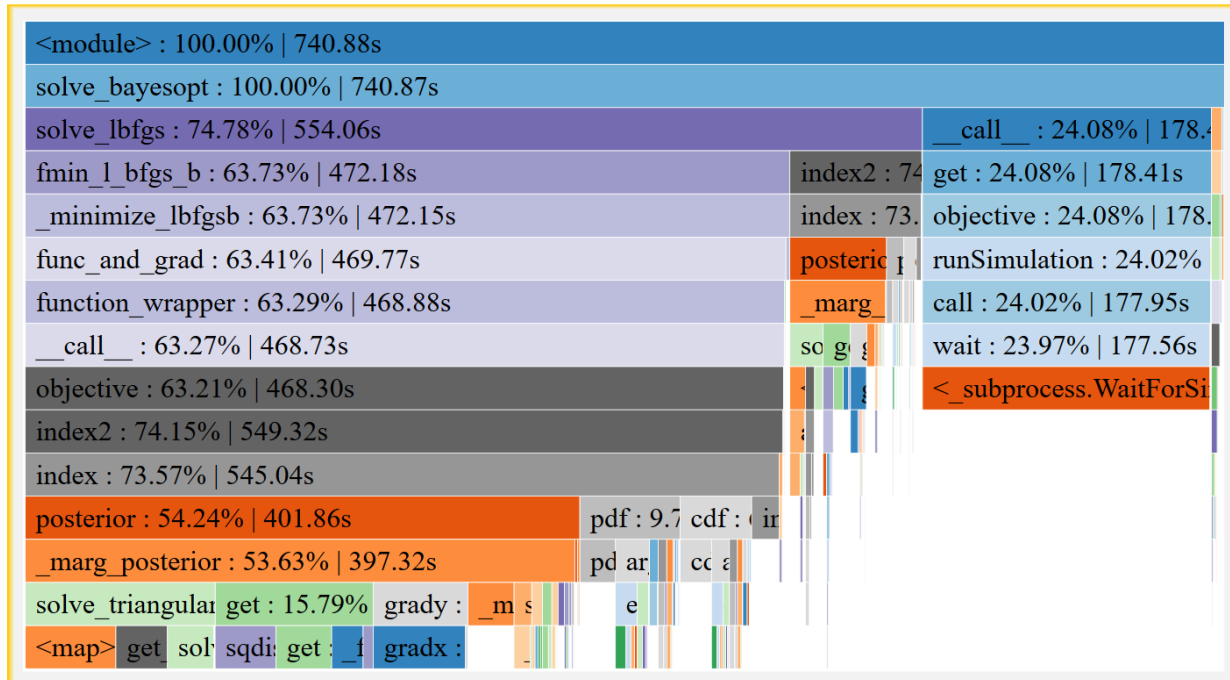


Figure 3.2.4 Icicle Graph of the Bayesian Optimization with the LBFGS solver and the M1N model

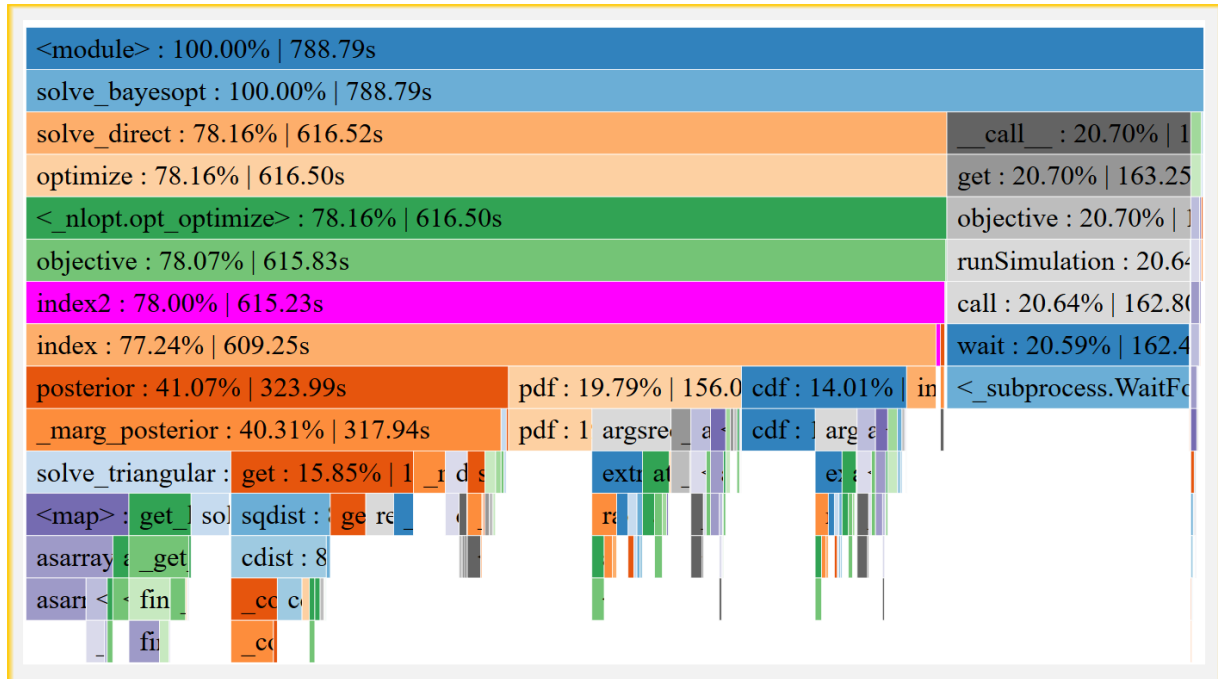


Figure 3.2.5 Icicle Graph of the Bayesian Optimization with the Direct solver and the M1 model

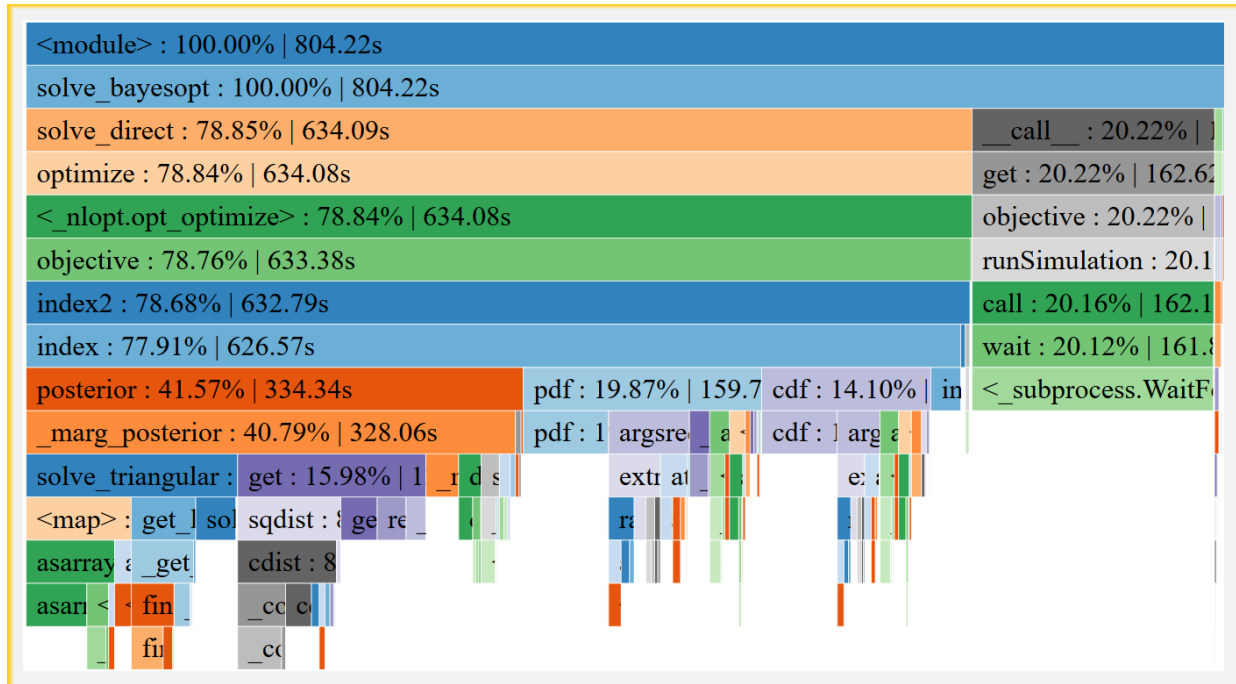


Figure 3.2.6 Icicle Graph of the Bayesian Optimization with the Direct solver and the M1N model

The figures generated by SnazeViz present the call structure of the code, percentage of the computation time consumed by the call and the total time accumulated by the call. The call stack for the two solvers differ, since both of the solvers call to different external libraries. Nlopt library for the Direct solver and scipy for the LBFGS solver.

Direct Solver

Direct solver is based on the DIviding RECTangles algorithm for global optimization, which is a deterministic and a systematic algorithm. The algorithm works by dividing the search domain into smaller and smaller hyperrectangles. As a result the performance of the algorithm is predictable, and consistent [16]. The figures 3.2.1 and 3.2.2 in addition to the averages of the results show the standard error of the data. The Direct solver provides more consistent result with smaller standard error compared to the LBFGS solver. The performance is 1%-2% slower under normalized models; the slow down partly stems from the scaling up and down the input between the models.

LBFGS Solver

LBFGS is part of a quasi-Newton family of algorithms, by bringing all the bounds between 0 and 1, we are not only scaling down the bounds, but also bringing them to the same magnitude, and leading the performance difference we observe in Figures 3.2.1 and 3.2.2 between the normalized and original models [17]. Unlike the Direct solver the algorithm is not systematic, but it is deterministic. The first step of the algorithm is finding the random initial points from which onwards it applies the rest of the algorithm. As a result, the performance of the algorithm is not as consistent as the Direct solver. The standard error bars are significantly larger with the LBFGS solver. Although, the solver may perform faster on a normalized model than a Direct solver, but this is not always the case.

3.3 LONG RUN TESTS

The previous experiments were performed for short runs, to get the basic idea of how different components of Bayesian Optimization effect each other. In the experiments performed in the section 3.1, the best performing set of hyper parameters was the Matérn 5 kernel with the Probability Improvement policy for the M1SNN model. The experiment was set up for only 100 iterations, and the best objective value achieved was not satisfactory. At 10 dimensions, the GP does not have enough data to be able to accurately predict the result. The more dimensions we have, the more data is required for the GP to provide meaningful mean and variance over a given data set. Therefore, the same set of hyper parameters were utilized for the long run experiment.

The experiment was set up on a remote server, and ran for total of 9 days, 1 hour and 43 minutes. Total of 2057 iterations were completed and each iteration recorded the objective value, input parameters and the time between iterations. Unfortunately, the variance of the prediction prior evaluation was not recorded. A reduction the variance would provide a progress indication of the accuracy of the GP.

The results of the experiment is presented in the graphs on the next page, Figures 3.3.1-3.3.3. The long run experiment has shown the performance degradation, from real world data, of the GP as the number of observations increases. In the experiment from section 3.4, we have covered the call stack solver and number of calls performed by the solver to the GP. The calculation of the posterior of GP is one of the major computational costs, with a computational complexity of $\mathcal{O}(n^3)$. On the figure 3.3.1, the graph shows the number of seconds between each iteration and a n^3 trendline. The experimental data closely matches with the trendline with exception of few outliers. As number of data points increase in the GP, the longer each iteration takes to be completed. At the same time, this exposes the weakness of the GP with large data set. The iteration time after 2000th iteration is over 1000 seconds per iteration, in the same amount time we can randomly sample our biological processes dynamic model over 300 times. As a result, even though in BO we have a guarantee of convergence to an optimal point given enough time and observations. Due to the computation cost of the posterior calculation, there exists a fine line between the computation time per iteration and the computation time of the objective function where we have to decide if it is desirable to use BO when the objective function computation time is fraction of the iteration time.

In terms of the objective value achieved, the best objective value is near optimal. The optimal result being 22.45 and best achieved objective value being 27.24 on 1157th iteration. The BO was unable to improve the objective value further for next 900 iterations. The overall cost in terms of the computation time is too high to consider such use cases viable, especially for models with higher dimensionality. Since with increase in dimensions, the GP would require more observation points to produce reliable predictions.

Long Run Experiment Results

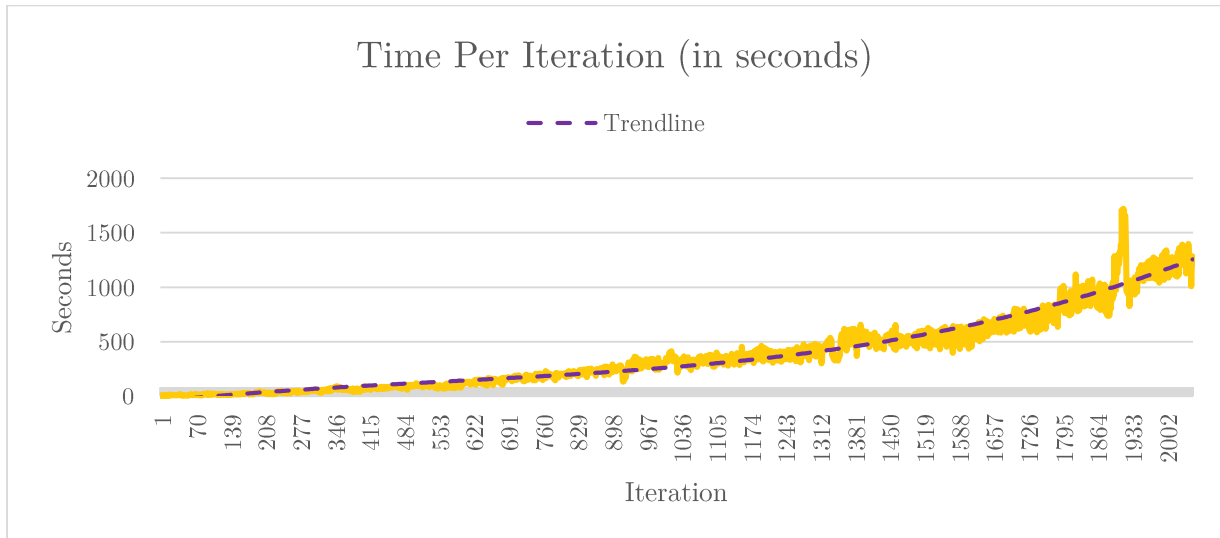


Figure 3.3.1 Long run experiment: iteration time in seconds with a fitted n^3 trendline

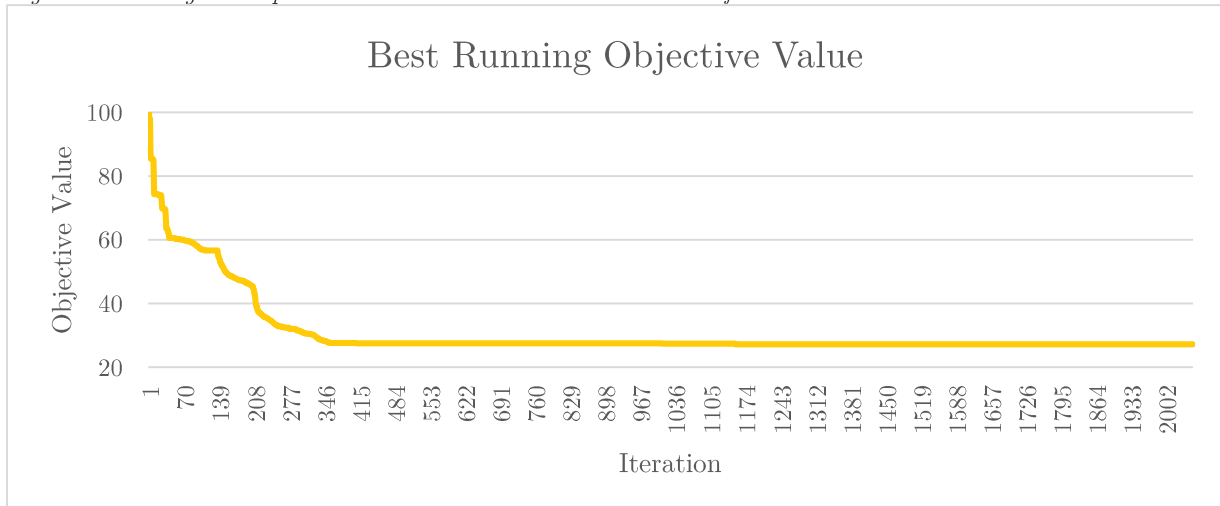


Figure 3.3.2 Long Run Experiment: Best Running Objective Value Graph

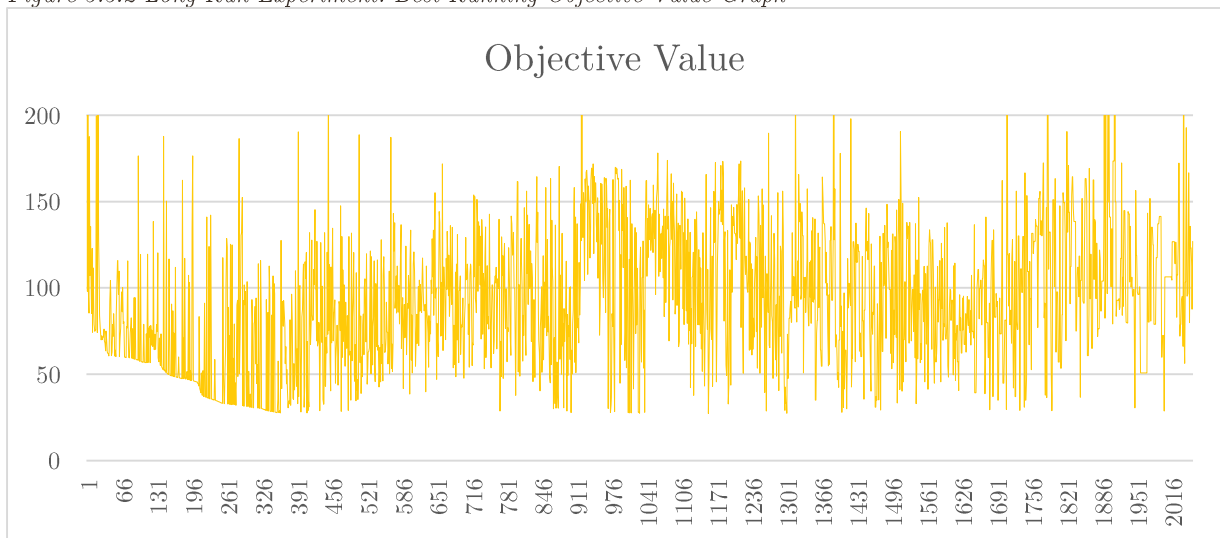


Figure 3.3.3 Long Run Experiment: Objective Value Graph

3.4 DIMENSION FIXATION, DIMENSION DETECTION

The following set of experiments were directed at exploring if the model can be optimized only by using a subset of dimensions, and how to choose the correct dimensions. The idea is, not all dimensions carry the same level of importance in the models. As a result, we might be able to reduce the computation time by reducing number of dimensions and improve the objective value at the same time. The reduction of computation would come from reduced number of dimensions in the GP, which in turn would require less observation points to improve the accuracy of the predictions of the GP.

The structure of the code was modified to enable fixation of n dimensions to a specific value. The GP was limited only to the first c dimensions of the observed points. From the initialized data, we set \mathbf{x}_b to the arguments with best objective value from the sampled data. Similarly, the solver would solve for only first c dimensions. The next point of evaluation received from the acquisition function is combined with \mathbf{x}_b to get back to the full dimensional input point. The combined input point is evaluated and added to the GP. The updated BO algorithm is presented below.

Algorithm 2: Bayesian Optimization with Fixed Dimensions

- 1: Sample the function n times, and update the GP $\{1 \dots c\}$ with the sampled data X, Y
 - 2: Set $\mathbf{x}_b = \operatorname{argmax} f(X)$
 - 3: **while** termination condition is not met **do**:
 - 4: Find $\mathbf{x}_{n+1} = \operatorname{argmax} a(\mathbf{x}_{n+1}, \text{GP}_n\{1 \dots c\})$
 - 5: Sample the objective function $\mathbf{y}_{n+1} = f\left(\mathbf{x}_b \xleftarrow{\{1 \dots c\}} \mathbf{x}_{n+1}\right)$
 - 6: Update the GP $\{1 \dots c\}$ with $(\mathbf{x}_{n+1}, \mathbf{y}_{n+1})$
 - 7: Increment n by 1
-

The experiment was carried out for 100 iterations with c equal to 2, 3, 4, 5, 6, 7, 8, 9 and full 10 dimensions. The M1SN model was used with Squared Exponential kernel and EI policy. The same initial data was used for all of the experiment.

Results:

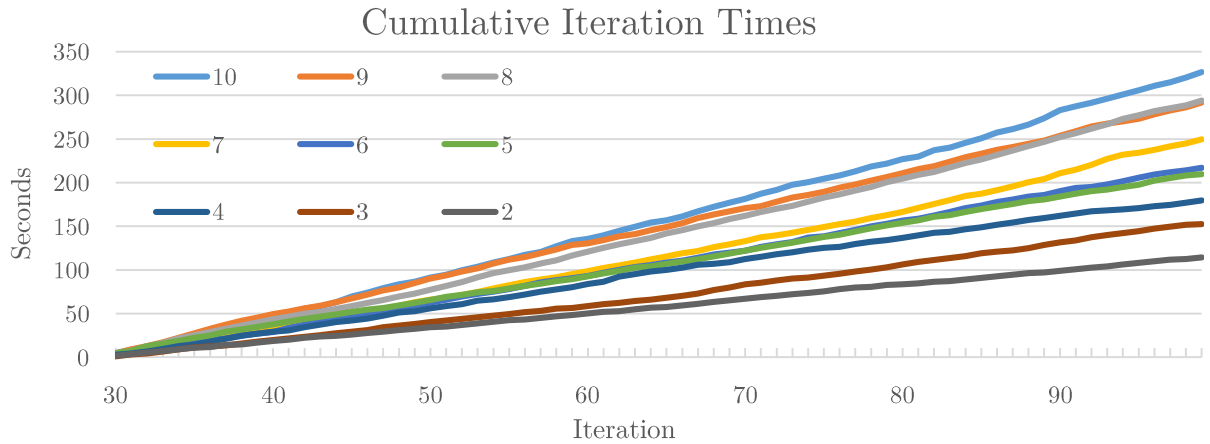


Figure 3.4.1 Cumulative Iteration times with different number of unfrozen dimensions

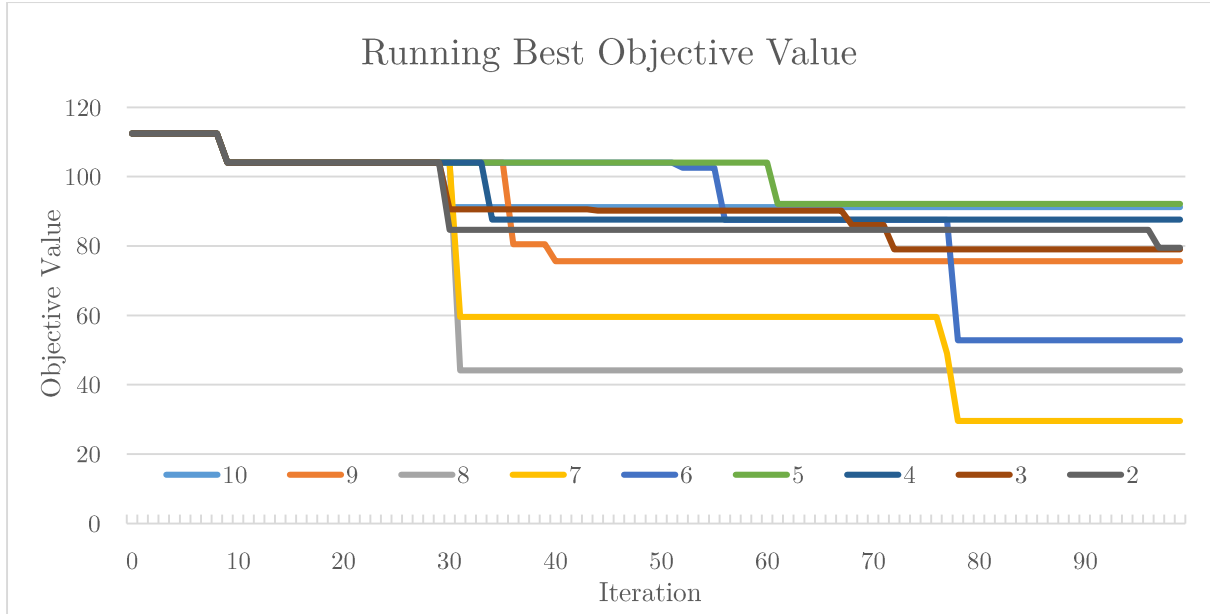


Figure 3.4.2 Running best objective value with different unfrozen dimensions

The computation time with different number of dimensions shows a linear growth. The fastest overall being at 2 dimensions and slowest at 10 dimensions. There is a clear computation time performance improvement when operating on lower dimensions.

The running best objective value graph, Figure 3.4.2, clearly shows possibility of improving the objective value by operating only on a subset of dimensions. The lowest objective value was achieved only by operation at 7 dimensions. More interestingly, the objective value improved even when the optimization was limited to only two dimensions.

The results of the dimension fixation showed the possibility of improvement of the objective even if we operate on lower dimensions. The problem is, which dimensions should be optimized along. We are dealing with a black-box function, and assuming we do not have an expert to guide us, how do we determine the most important dimensions?

To observe the characteristics of the inputs for the M1SN model, a simple technique was performed to analyse the dimensions. 2000 random input points were generated for the model, and evaluated. The points were filtered out, where any point with an objective value of higher than 100 was throw-out. These are arbitrary rules, performed on rule of thumb guidance. Out of 2000 random points, only 87 points remain in the final “Filtered” data set. Another data set was collected from all the experiments performed throughout the duration of the project, with total of 93797 data points in the “Horde” data set.

The data sets were processed with standard Principle Component Analysis technique (PCA) to analyse the dimensions. PCA is a common dimensionality reduction technique applied in the pre-processing phase in machine learning algorithms. The procedure is aimed at transforming the points into linearly uncorrelated variables. In our case, we are interested in the variance of the dimensions, and we assume the higher the variance, the more important the dimension is.

Data Set	Dimension									
	1	2	3	4	5	6	7	8	9	10
Horde(93797)	0.16059	0.1287	0.1147	0.1096	0.0644	0.0985	0.0908	0.0726	0.0781	0.0821
Filter(87)	0.16969	0.0358	0.1289	0.1229	0.1152	0.0721	0.0774	0.0816	0.1026	0.0937

Based on the numbers obtained via applying PCA, we can assume that dimensions carry varied importance. If we assume, the Horde dataset is more representative of the dimension variance due to the larger data set, the Filter dataset approximates the large dataset with fewer points with few exceptions (Dimensions 2 and 5). The downside of both approaches is the requirement of the data prior BO is applied. Therefore, both approaches come at a costs which we are trying to minimize.

3.5 MISCELLANEOUS NOTES:

- The experiments with the mean from the data achieved lower objective values quicker compared to the GPs with mean zero. However, in the long term stopped improving in a similar manner as mean zero GPs.
- A quick experiment with a similar set-up to Additive BO performance in terms of the computation time was magnitudes times slower, due to the fact that in additive BO all GPs contain all the observations (for their limited dimensions set). The experiment was terminated due to the long computation time. The experiment did not use the original code, and did not use the custom policy used by the original code, therefore is omitted from this report.

4 BAYESIAN OPTIMIZATION WITH DIMENSION SCHEDULING ALGORITHM

The Bayesian Optimization with Dimension Scheduling Algorithm (DSA) was devised based on the findings from the Chapter 3. In this chapter we summarize the problems faced with traditional BO, introduce DSA and how DSA addresses the problems faced with BO, present experimental data and discuss advantages and the limitations of the DSA.

4.1 INTRODUCTION

BO is commonly used for optimizing costly black-box functions in terms of the computation or even a physical experiments. In our case, our objective function is relatively computationally cheap, but at the same time complex enough to render common global black-box optimization methods inefficient. Due to the complexity of the objective function the traditional BO requires many samples, but still magnitudes less than other methods (i.e. Direct or LBFGS, covered in the Chapter 3).

One of the common problems encountered with BO is the algorithm performance severely degrades in high dimensions and/or with large data amount of observations. As we have covered in the previous chapter, the GP scales linearly in terms of dimensions and in cubic scale in terms of the observation points. Although GP scale linearly in terms of the dimensions, the GP at higher dimensions requires more observation points for more accurate predictions. The latest research dealing with problem of dimensionality work under strong assumptions; an additive BO works under an assumption the underlying black-box function is an additive function [11]; REMBO assumes the low dimensional problem is hidden in a highly dimensional problem [18]. In cases where these assumptions do not hold, the performance of these methods is underwhelming. The dynamical biological model is an example of such function.

To address the problem with high dimensionality and many observation points, we introduce Bayesian Optimization with Dimension Scheduling Algorithm (DSA). The DSA is tailored to our problem, and diverges from the traditional Bayesian Optimization use case. The DSA distributes the observation points across many GPs, with each GP containing observation points with a subset of dimensions. The total number of GPs by default is dependent on the total number of dimensions and subset size for each GP. Each new iteration, a new subset of dimensions is selected from a given probability distribution and optimized along. As a result the DSA benefits from a faster computational performance and achieves better objective values compared to the BO.

The following sections present a more detailed breakdown of the algorithm with an example run, followed by experimental results comparing the DSA to traditional BO.

4.2 ALGORITHM

Algorithm 3: Bayesian Optimization with Dimension Scheduler

- 1: Sample the function n times, update the all the GPs with sampled data X, Y
 - 2: Set $\mathbf{x}_b = \operatorname{argmax} f(X)$ and $y_b = \max f(X)$
 - 3: **while** termination condition is not met **do**:
 - 4: Update P from the observations *(Optional)*
 - 5: Randomly select dimension set Z from the dimensions probability distribution P .
 - 6: Find $\mathbf{x}_{n+1}^Z = \operatorname{argmax} a(\mathbf{x}_{n+1}^Z, \mathbf{GP}_Z)$
 - 7: Sample the objective function $y_{n+1} = f\left(\mathbf{x}_b \stackrel{Z}{\leftarrow} \mathbf{x}_{n+1}^Z\right)$
 - 8: Update the \mathbf{GP}_Z with $(\mathbf{x}_{n+1}^Z, y_{n+1})$
 - 9: If $y_{n+1} > y_b$ then $\mathbf{x}_b = \mathbf{x}_b \stackrel{Z}{\leftarrow} \mathbf{x}_{n+1}^Z$ and $y_b = y_{n+1}$
 - 10: Increment n by 1
-

The first step of the algorithm is similar with all other BO methods, to sample initial observation points from the function. Based from the sampled observations (X, Y) , we set $\mathbf{x}_b = \operatorname{argmax} f(X)$ and $y_b = \max f(X)$. The \mathbf{x}_b stores ours best known arguments for the highest objective value, and y_b stores the best objective value. We start the optimization process by optionally updating our probability distribution from the observed data. In our implementation we use PCA, and use the eigenvalues as the probability of the dimension. We pick a random dimension set Z from the dimensions probability P . The size of the set is a parameter set by the user. For the subset Z we optimize \mathbf{GP}_Z for the next point for evaluation with our acquisition function $a(\bullet)$. We replace dimensions Z in \mathbf{x}_b and evaluate the updated point. The y_{n+1} and \mathbf{x}_{n+1}^Z are added only to the \mathbf{GP}_Z , and if the y_{n+1} is a better objective value than y_b , we update \mathbf{x}_b and y_b with the data from the previous iteration.

Practical Considerations

Probability Distribution

The algorithm requires a probability distribution to pick randomly the dimensions at each iteration. The probability distribution can be supplied by an expert, who can reliably assign probabilities to each dimension. Unfortunately, in most cases we do not have such service at hand. We have covered two methods to generate a probability distribution from the data in the Section 3.4, but both of the methods require some form of pre-processing.

An alternative method applied in the implementation of the DSA is an online updating probability distribution. The algorithm starts with a uniform distribution, and every 50 iteration updates the probability distribution by applying PCA over the observation points. The frequency of updating the probability distribution is an arbitrary choice. The online PCA method arrives to an approximately close results as

Horde and Filter methods covered in the Section 3.4, as shown on the table below, Table 4.2.1. The results are for an experiment run with 1000 iterations, and the probability distribution shown is from the last iteration of the experiment.

Data	Dimension									
	0	1	2	3	4	5	6	7	8	9
Horde(93797)	0.1605	0.1286	0.1146	0.1095	0.0643	0.0985	0.0907	0.0726	0.0780	0.0820
Filter(87)	0.1696	0.0358	0.1289	0.1228	0.1152	0.0721	0.0774	0.0815	0.1026	0.0936
Online(1000)	0.1635	0.0650	0.1177	0.0787	0.1088	0.1047	0.0973	0.0841	0.0908	0.0889

Table 4.2.1 Probability Distribution derived with different data sets

Z: Subset Size

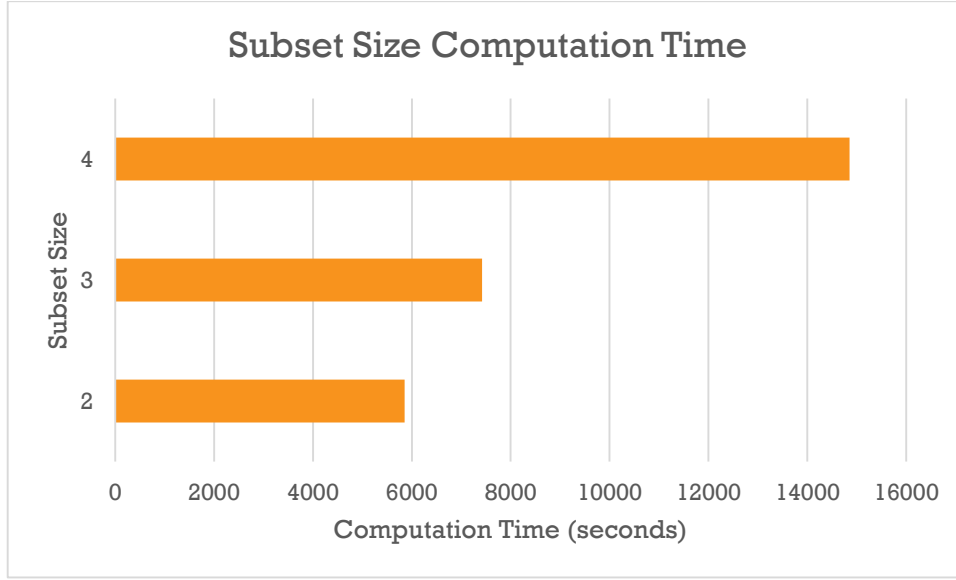


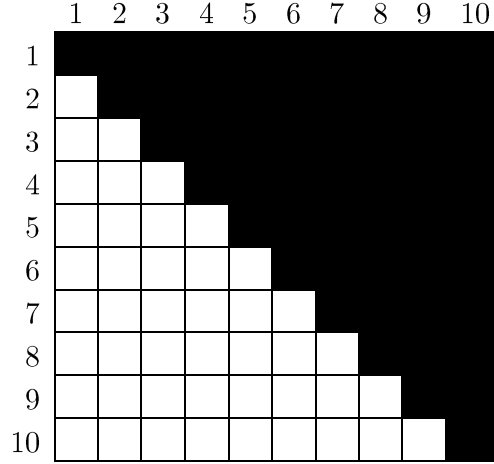
Figure 4.2.1 Computation Time Graph for different subset sizes. Averaged result of 5 experiment runs with different subset sizes over the M1SN model. Each experiment set for 1000 iterations with EI policy and SE kernel.

Every iteration, the algorithm selects a subset dimensions Z to optimize along. The size of the subset has a significant impact on the performance of the algorithm, as shown on the Figure 4.2.1 above. A larger subset size, means our GP operates on a higher dimension, which adds linear growth. In addition to the GP, the solver of the acquisition function maximizes on higher dimensions. The combination of the two, result in a slower computation time, without significant gains in the objective value.

Performance Analysis

The performance increase stems from the reduced number of dimensions for the GP and the solver, and reduced number of observations per GP. As discussed in the previous chapter, computational complexity of posterior calculation is $\mathcal{O}(n^3)$, by distribution the observation points across multiple GPs we are dividing the computational cost by number of GPs. The growth of the algorithm still remains at $\mathcal{O}(n^3)$, but it is divided by total number of

GPs used and the probability distribution. Total number of GPs used by the algorithm is equivalent to the total number of permutations of the set Z . The total number of permutations of the set Z is based on the sizes of the subsets and dimensionality of the problem. The diagram below represents all the permutations used by the algorithm at subset size 2 on a 10 dimensional problem, where the white cells are the permutations used by the algorithm.



Number of observations per GP is dependent on number of initial observation points, current iteration and the probability distribution. The formula below calculates the average case scenario for subset Z , it does not take into consideration online probability distribution updates.

$$i + (n - i) \prod_d^Z P(d)$$

Implementation Details:

The implementation of the DSA differs from the simplified pseudo code presented. For more efficient execution the GPs are created lazily. In other words, the instances of GP_Z are created only when the set Z is selected. When the dimensions set Z is selected, a new GP_Z is created, and the prior of the GP is set from all the observation points gathered throughout the experiment. As noted in the previous chapter, by setting the prior from the data we reduce the training time of the GP.

4.3 EXAMPLE RUN

Below is an example of the DSA running on a real 3 dimension minimization problem:

Intial Sampled Data:

$$X = \begin{bmatrix} 2.61 & 1.16 & 5.351 \\ 6.36 & 0.91 & 6.60 \\ 1.36 & 1.91 & 1.60 \\ \vdots & & \end{bmatrix} \quad Y = \begin{bmatrix} 572.12 \\ 1014.29 \\ 33.65 \\ \vdots \end{bmatrix} \quad B = \begin{bmatrix} [0,10] \\ [0,2] \\ [0,10] \end{bmatrix} \quad P = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}$$

Add (X, Y) to all GP_Z

Set Best objective and its arguments

$$x_b = [1.36 \quad 1.91 \quad 1.60] \text{ and } y_b = 33.65$$

Pick random dimensions set Z

$$Z = [1 \ 3] \therefore \text{fix dimension 2 to 1.91}$$

Find $x_{n+1}^Z = \operatorname{argmax} a(x_{n+1}^Z, GP_Z)$

$$x_{n+1}^Z = [0 \ 0]$$

Evaluate $y_{n+1} = f(x_b \leftarrow_Z x_{n+1}^Z)$

$$33.65 = f([0 \ 1.91 \ 0])$$

Add $([0 \ 0], 33.65)$ to GP_Z , and since $y_{n+1} < y_b$ is not true y_b and x_b are not updated.

Pick random dimensions set Z

$$Z = [2 \ 3] \therefore \text{fix dimension 1 to 1.36}$$

Find $x_{n+1}^Z = \operatorname{argmax} a(x_{n+1}^Z, GP_Z)$

$$x_{n+1}^Z = [1.37 \ 0.81]$$

Evaluate $y_{n+1} = f(x_b \leftarrow_Z x_{n+1}^Z)$

$$21.72 = f([1.36 \ 1.37 \ 0.81])$$

Add $([1.37 \ 0.81], 21.72)$ to GP_Z , and since $y_{n+1} < y_b$ we update the variables to:

$$x_b = [1.36 \ 1.37 \ 0.81] \text{ and } y_b = 21.72$$

\vdots

Repeat the cycle till the termination condition is met

4.4 EXPERIMENTS SET-UP AND RESULTS

For the comparison experiments between traditional BO and DSA, 10 different models were tested with 4 experiment runs per each model. Each experiment ran for 300 iterations, with SE kernel and EI policy. The models used in the experiment were variants of the algae models specified in the second chapter, with some models at higher dimensions. The table below summarizes model details and results of the 4 runs.

Results: Objective Value

		Experiments							
		Best Achieved Objective Values							
Model	d	BO:1	DSA:1	BO:2	DSA:2	BO:3	DSA:3	BO:4	DSA:4
M19	10	58.95	39.40	51.09	30.94	47.10	24.79	58.31	32.13
M19o	10	80.93	136.70	87.78	F	34.98	F	106.66	F
M26	11	56.14	30.47	F	27.14	65.63	41.53	110.87	26.18
M29G	11	46.56	37.58	7301	37.98	48.88	25.78	79.92	30.95
M29C	11	60.40	76.41	48.74	37.26	75.97	29.97	50.03	45.41
M31	11	47.80	31.55	51.68	25.77	52.12	31.38	66.02	38.95
M32C	12	62.19	23.20	61.32	36.30	52.44	32.17	60.56	32.71
M32G	12	57.42	27.38	47.80	41.86	42.47	35.91	64.41	27.07
M33	12	52.44	24.61	56.24	30.47	43.86	34.00	58.50	30.31
M35	12	44.63	33.54	49.76	29.78	54.25	28.68	42.62	28.02

Table 4.4.1 Objective Values Table

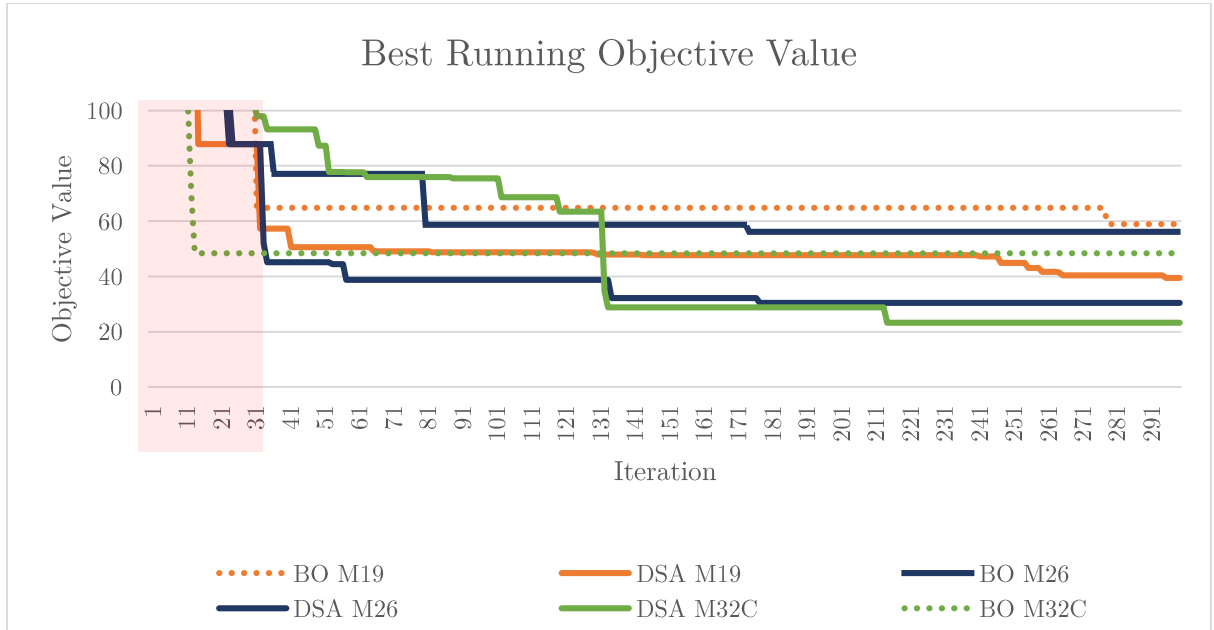


Figure 4.4.1 Graph of Best Running Objective value from the experiment run 2

Note: Some experiments have failed due to the error within the model provided. The model M19 is specifically problematic.

In most cases the DSA outperformed traditional BO in terms of the lowest objective value. With exception of 2 instances and failed experiments due to the underlying model, DSA achieved lower objective values than the BO method. The table 4.4.1 presents the summary of the experiments in terms of the objective value and the Figure 4.4.1 present a sample of running best objective values. As in the experiments carried out in the third chapter, the traditional BO method tends to improve less frequently as the optimization process progresses. The DSA method due to the constant changes of the dimensions tends to improve the objective value more frequently and overall achieves a lower objective value. Due to the randomness present in the DSA, the performance of the algorithm does not always yield the near optimal objective value, as seen in some instances of the experiment on the table 4.4.1

Results: Completion Times

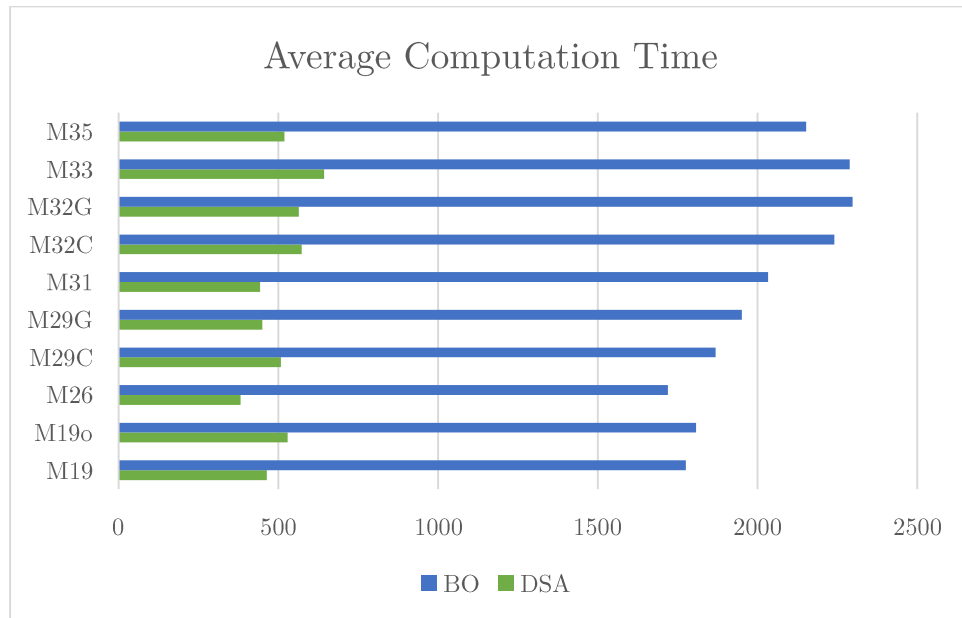


Figure 4.4.2 Average computation time of the DSA and BO on different models

In addition to achieving the lower objective value, the DSA algorithm completes each experiment on average at a fifth of computation time. The figure 4.4.2 shows the average computation time of the experiment runs with all the models. Due to the performance enhancing features covered in the previous section, DSA completed all experiment runs quicker than the BO method.

4.5 LIMITATIONS AND FURTHER IMPROVEMENT

The dimension scheduling aspect of the DSA algorithm relies on random dimension selection from a probability distribution. The randomness of the scheduler results in inconsistent results as encountered during the experiments section. A more deterministic dimension scheduling approach would provide a more consistent performance of the algorithm.

Unlike traditional BO, the DSA does not have proof of convergence. At no given point does any GP in the DSA contain all the data points to converge the variance to zero. As a result, DSA achieves only near-optimal result with a possibility of an optimal result. The issue of convergence could be addressed by a hybrid BO solution with the function space sampled for near optimal solutions with DSA, and further optimized with the traditional or other BO methods.

Another solution considered for an improvement of the performance of the DSA algorithm is to schedule the dimensions on the GP level. As we have covered in the previous sections, the main performance bottleneck is at the solver level. The solver requires more posterior calls at higher dimensions, by limiting the solver to a subset of the dimensions we can applying the same convergence proof to this solution. By using a GP with all dimensions and observations points, we would be sacrificing the performance gains achieved by distributing the observation points across many GPs. The performance of the GP could be aided by caching part of the posterior calculations, since only subset of dimension's values are change. The proposed solution would require further work to test if the statement holds true.

The DSA address only a subset of problems with BO under specific conditions. The algorithm faces similar challenges when we increase number of dimensions. If $d \gg n$, the GPs' accuracy would suffer since the observation points would be spread thinly over many GPs. In the current implementation, number of GPs is equivalent to number of permutations of the subset Z are used by the DSA, however not all GPs perform equally well. The GPs could be eliminated over time by calculating the marginal likelihood of the GPs. Thus optimizing only along dimensions where the GP_Z has a high marginal likelihood. Alternatively, we may simply limit number of GPs in our DSA optimization process.

As indicated in the introductory statement, DSA is designed for our specific use case. The DSA algorithm does not take into consideration the regret accumulated over the optimization process. Cumulative regret is defined by $R_T = \sum_n^N f(x^*) - (f x_n)$, and plays an important role in bandit problems such as reinforcement learning and online advertising. Alternative BO variant would be more suitable for such applications[19].

One of the advantages of the Bayesian Optimization with Dimension Scheduler over traditional Bayesian Optimization is easy parallelization the code for increased performance on multicore systems. The traditional Bayesian Optimization method can be parallelized to a

certain extent. The Gaussian Process has been successfully parallelized[13] with gains in performance. The solvers can be parallelized by dividing the search space between different processes. These approaches provide compartmental parallelization, whilst the whole process is still sequential.

The parallelization process of the DSA is much simpler, and can use current GP and solver modules. The solution lies in distribution of the GPs across many processes, and a manager process to communicate between all the child-processes and the objective value function. The manager would contain the \mathbf{x}_b and \mathbf{y}_b variables, and assign different iteration points to each process. Each process would contain a GP and a solver, and based on the iteration number the process receives (if we have exploration parameter scheduler), the process maximizes the acquisition function for the GP in the process and returns the solution to the manager. The manager would evaluate the solution and return it to the process to update the GP and start a new iteration. A graphical representation of the parallelized DSA is presented on the next page, Figure 4.6.1.

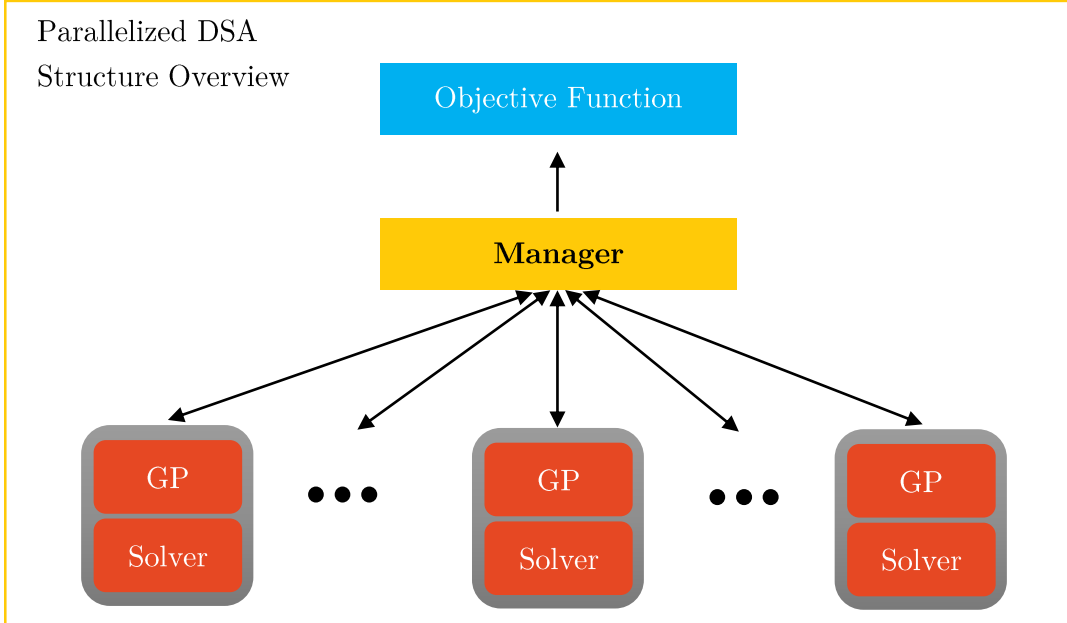


Figure 4.5.1 Parallelized Implementation of DSA structure overview

4.6 CONCLUSION

We have presented the DSA algorithm and successfully applied it to our problem area. The DSA addresses the problem of performance degradation in BO in terms of the computation time required with many data points, by distributing the observations across many GPs and reducing the dimensionality of the input for each GP, the DSA outperforms the traditional BO algorithm. With further work specified in the prior section, the performance of the algorithm in terms of the computation time and accuracy can be improved further.

5 BAYESIAN OPTIMIZATION FRAMEWORK

The main objective of the Framework is to provide the users with ability to use Bayesian Optimization methods without any prerequisite knowledge of Bayesian Optimization or python coding skills. This is achieved by providing the users with a clear and easy to use Graphical User Interface, and command line interface to set and invoke Bayesian Optimization methods over any given model which can be called via a command line.

The following section covers the development of the Bayesian Optimization Framework. The sections starts analysis of the users, the requirements of the framework; followed by the reason behind design choices of the framework and implementation of the framework. The testing section covers the user testing and unit-testing performed to ensure the software performs as expected. The final section discusses the further possible improvements and concludes the software development section.

5.1 ANALYSIS

The development of the software started with early sketches of the user interface. To analyse how a user may interact with the software, and how they may use it. The initial designs for the interface divided users based on their experience and knowledge of the Bayesian Optimization techniques. Thus the first screen presented the user with an option to select their level during the first interaction with the software. Depending on the selection made by the user, different set of parameters would be presented to the user. The initial sketch is presented below (Figure 5.1.1).

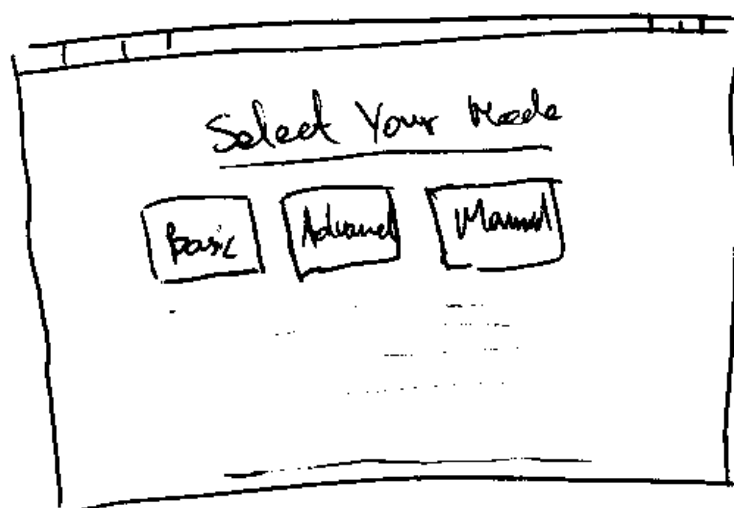


Figure 5.1.1 The greeting page would let select their expertise level

Prior to any further development of the framework, potential users of the framework, Dr Caroline B. and Dr Benoit C. from Imperial College Chemical Engineering Department, were presented with the initial sketches and questioned on how they would like to interact with the Bayesian Optimization framework. The feedback from Dr Caroline B. and Dr Benoit C., formed the bases of further development. The main request on their behalf was ability to invoke the Bayesian Optimization through a command line without any interaction with the graphical user interface. The command line interface would allow the optimization to be carried out remotely on a server without use of any Remote Desktop utilities.

The graphical user interface aspect of the framework would still play an important role for easy parameter settings, local execution and visual feedback of the Bayesian Optimization. The development of the framework was an iterative approach with features required analysed during the development process.

5.1.1 IMPLEMENTATION CHOICES

The core of the project was developed on top of the modular Bayesian Optimization library pybo[2]. The pybo library is written in Python 2.7 and has dependencies on pyg, numpy, scipy and matplotlib modules. The pybo was chosen as the core of the project since it is the only library providing an easy modular approach to Bayesian Optimization. The library provides the most common acquisition functions, kernels and initializers with ability to define own custom modules to be used by the library to perform Bayesian Optimization. Alternative options included Limbo (C++11)[20], Spearmint (python)[21] and DiceOptim(R)[22] which do not have the modular flexibility of the pybo library, nor contain all the options provided by the pybo library. The core library determined the programming language and the graphical user interface to be used to build the framework.

5.1.2 GRAPHICAL USER INTERFACE (GUI) LIBRARY

Python offers variety of Graphical User Interface libraries with different capabilities and targeting different uses. To ensure all the requirements of the framework were met, the GUI library was decided based on the following requirements for the library:

- Cross Platform
 - The Graphical user interface should support all major platforms, Windows, OS X, and other Unix based operating systems such as Ubuntu and Linux Mint.
- Free or GLP Licence:
 - The library should not require any payment from the developer or the user
- Lightweight and Simple

- The user interface does not require advanced user interface features, therefore a simple library with barebones graphical user interface elements would suffice.
- Support for Python 2.7
 - Components of used by the framework, such as pybo and pygp are based on Python 2.7. Thus to avoid any compatibility issues with the modules, and reduce confusion in the source code the UI library must be compatible with Python 2.7.

The requirements for the GUI library narrowed down the choice of libraries to PyQt, PySide, Tk and wxPython. The four libraries include all the necessary widget elements to build the framework user interface, the deciding factor for the Tk library was the abundances of the library. Tk library is the default GUI library for Python, and comes with all the installations of Python since the version 2.6. As a result, it minimizes amount of external library dependencies. In addition, the Tk library is well documented, tested, with an active community of developers.

5.2 DESIGN AND IMPLEMENTATION

5.2.1 FRAMEWORK STRUCTURE:

FRAMEWORK OVERVIEW

The framework consists of the Graphical User Interface, File Parser, Bayesian Optimization Module, Command Line Wrapper, and Core modules. The modules are decoupled from, thus allowing modifying one module without effecting other modules. Each module has following purpose:

Graphical User Interface:

The module contains all the graphical user interface elements, and deals with the visualization of each stage. The GUI is invoked by the core module which determines if the user invoked the GUI or the Command Line.

File Parse:

The file parse module deals with parsing files in and out of the framework. The parser ensures the file contain all the necessary fields, and checks the values of the parameters are in the correct format. In addition, the module contains functions to convert the GUI module's parameters into a list or update the GUI module's parameters from a list.

Bayesian Optimization Module:

The following module is heavily based on the pybo's implementation of the Bayesian Optimization. The modifications allow for greater flexibility in model definition, and introduces the dimension scheduling algorithm.

Command Line Wrapper:

The command line wrapper creates an object from the parameters provided. The object upon request updates the input file, calls the module to perform the calculations, reads the output of the model and returns it to the caller of the module.

Core:

The core module, upon a call from the user scans the machine for the necessary modules. The user is notified if the required Python libraries are not installed; else launches the GUI module. Alternatively, if the user specifies to invoke without the GUI, it parses the parameter file via the File Parsing module and passes on the parameters to the Bayesian Optimization Module.

Below is the graphical representation of the modules and how they interact with each other. The framework is started by invoking the core module, which in turn calls the file parser or the GUI modules. The file parser can export the parameters to the requested directory or continue through to the command line wrapper. The GUI, once the user enters the parameters also creates a command line wrapper object. The command line wrapper object is inputted into the Bayesian Optimization module with rest of the parameters. The Bayesian Optimization module starts the process of optimization, updates the GUI if necessary with the latest values. Each iteration the Bayesian Optimization module writes the updates to the output file of the Framework.

Modules Interaction Overview

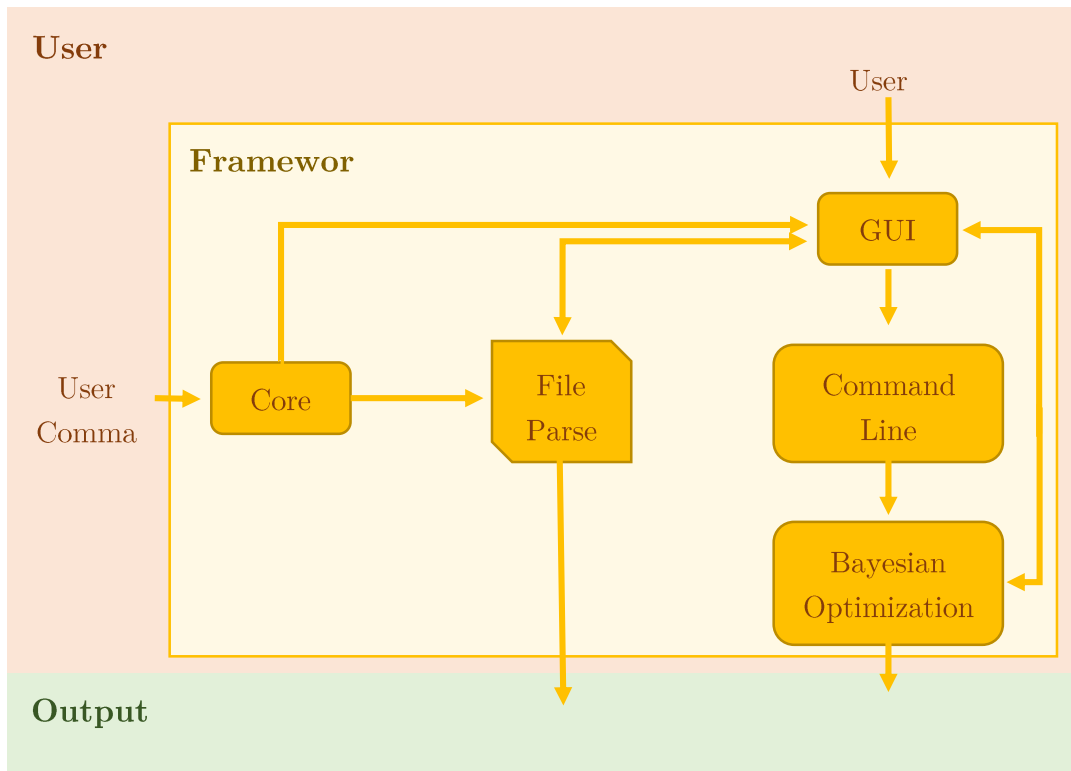


Figure 5.2.1 Framework abstract overview showing the interaction between all the modules

DEFINING A FILE FORMAT

The Framework defines a simple CSV (Comma Separated Values) as a file format which specifies the parameters of the Bayesian Optimization and model details; such as the input file, command and the output directory. The specified file format is human readable, thus allows the user to create/modify parameters file with any basic text editor; or alternatively use spreadsheet software for a more structured view. In addition, the file format is easy to parse with any programming language, with little overhead.

The file structure follows a simple pattern, each line stores only one parameter. First value is the parameter key, for example “policy”; followed by the value of the parameter, separated by a comma such as “ei” (expected improvement policy). The structure allows having multiple values for each parameter, but this is limited only to the kernel and policy parameters. The limitation is imposed by the graphical user interface, where only multiple kernel and policies can be selected. In addition, to the parameters the file format stores the data gathered during optimization or user pre-processed data. The order of the parameters does not matter, except for the data parameter. The data parameter must be defined last followed by the data where the first column is y values and starting from the 4th column are the x inputs. Full detailed breakdown of the file format specification is presented in the Appendix A2.

Alternative possible file formats included XML and JSON based file structures. Both file formats can be more expressive in their notation. For example XML based file format could specify a policy with a tag, and specify the parameters of the policy with attributes. Examples of each file format would be structure for the same parameter is shown on the right. The CSV file format is less verbose compared to the alternative file formats presented, and the benefits of the JSON and XML formats are minimal. Therefore, CSV based file format was chosen as the default file format for the framework.

Example XML snippet:

```
...
<policies>
  <policy xi=0.1>SE</policy>
</policies>
...
```

Example JSON snippet:

```
{...
  Policies: {policy: ei,
             xi: 0.1},
...}
```

Example CSV snippet:

```
...
policy, ei
eixi, 0.1
...
```

5.2.2 USER EXPERIENCE (UX) DESIGN:

GRAPHICAL USER INTERFACE (GUI)

The designing of the User Interface was an iterative process, only final iteration of the UX is presented in this subsection. The User Experience is divided into three stages: Selection, Observation and Evaluation. Each stage suits a different purposes.

Selection:

Figure 5.2.2 Selection Window

The selection stage requires the user to set the model parameters, which includes model invocation command, input file, output file, and the bounds of the parameters. The user is presented with a choice of policies, and kernels; the user may select multiple options for the framework to try out. Optionally, the user may specify the output directory for the framework. The user interface tracks the changes of the parameters, and turns on the “Optimize” button only when all the necessary parameters are set correctly.

For more advanced users, there is an option set more in-depth Bayesian optimization parameters. The “Advanced Settings” button invokes the panel with options to set the recommender, initializer, number of samples to sample by the initializer and Markov Chain Monte Carlo settings on the left side. The right hand side panel includes the Gaussian Process settings and hyper prior settings, and option to bootstrap the Bayesian Optimization processes with data. The fields in the MCMC and GP section allow python code for more dynamic settings. For example, the noise parameters can be set from the initialized data, by a following code snippet “np.std(Y)”.

Once the user is happy with their settings, and the framework has all the necessary parameters set, the user may continue to the next phase by pressing the “Optimize” button.

Observation:

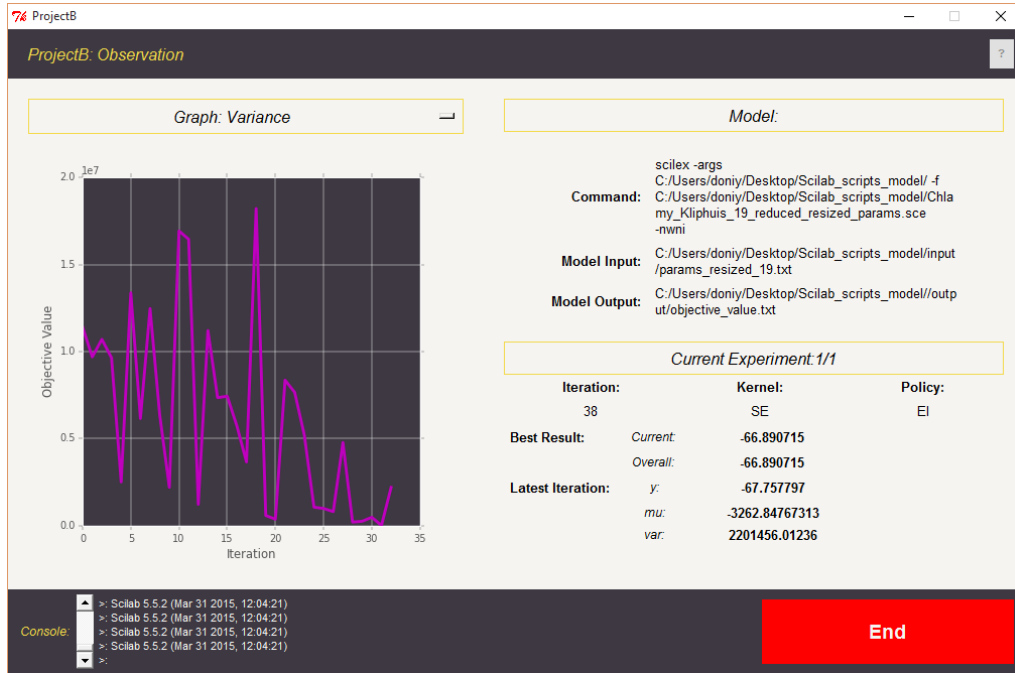


Figure 5.2.3 Observation Window

The observation stage presents the user with an overview of the current stage of the optimization process. The right hand side provides the information on the model being optimized, parameters (i.e. kernel, policy) used and current stage of the optimization process. The panel shows the best objective value in the current experiment, overall best objective value achieved and information on the latest iteration.

The left panel of the observation stage provides the user with a graph of objective values, iteration times and variance prior evaluation of the Bayesian Optimization Process. The graphs are updated after every iteration providing the user with a live overview of the progress. The user is presented with a range of graphing options, including objective value, best running objective value, variance and timing graphs.

Evaluation:

The evaluation stage is presented to the user once all the experiments are completed or terminated early. The page provides with a summary of all the experiments, providing the user with information on the completion time of the experiment in seconds and the best value achieved during the experiment. The user may optionally query the Gaussian Process created during the Bayesian Optimization process. By querying the Gaussian Process with input points, the user invokes the Framework to update the output file with the mean and variance of the requested points.

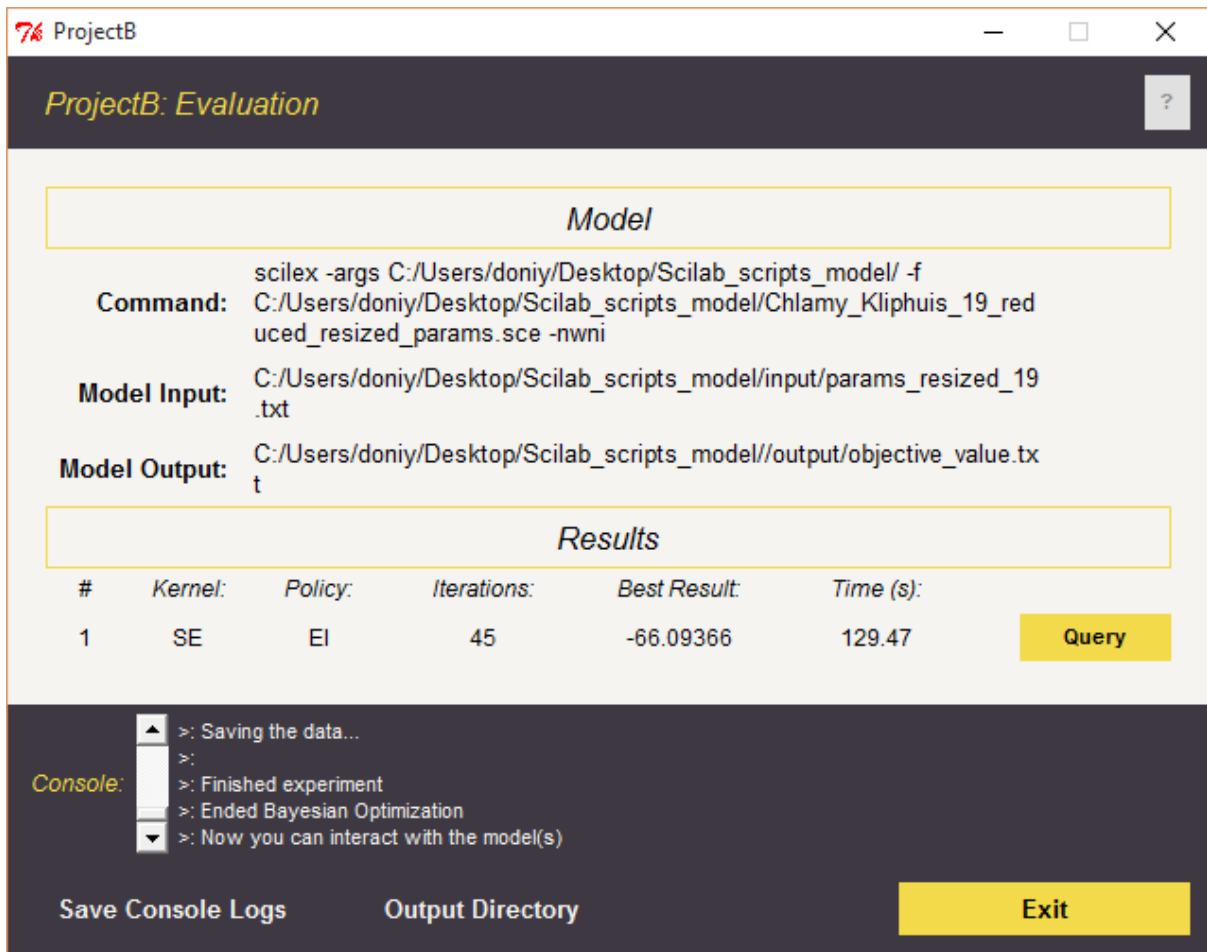


Figure 5.2.4 Evaluation window

In addition, the Evaluation page user provides the user with small handy shortcuts to export the console logs, open the output directory and exit the framework. Exiting the framework automatically closes the framework GUI and all the processes invoked during the Bayesian Optimization process.

COMMAND LINE INTERFACE (CLI)

The command line interface was designed to with common command line syntax, to reduce the learning curve for the users. The CLI is invoked by simply adding “-cli” to the framework invocation command, the interpreter ensures the command satisfies the requirements, and ensures all the necessary libraries are present. If the user does not meet one of the requirements, the user is provided with a message specifying what has cause the failure. For example, if the user does not have numpy module installed, the interpreter would request the user to install it.

The command line interface is specified as follows:

```
Framework Invocation (default GUI): python -m projectb.start

Usage: [-h] [-v] [-cli] [paramfile] [outputdir]

Positional Arguments:
  paramfile    The URI of the parameter file
  outputdir    The URI of the output directory

Optional Arguments:
  -h, --help
              shows help message and exits
  -v, --verbosity
              increases output verbosity, 1 will inform of only
              about start and finish of experiments. 2 will
              inform of each iteration of the process.
  -cli
              Use Command Line Interface, requires paramfile
```

5.2.3 IMPLEMENTATION OPTIMIZATION

Implementing the whole project a monolithic process would lead to a very inefficient code and result in a static GUI. The user would not be able to interact with the GUI whilst the Bayesian Optimization process would be running. Therefore, the framework splits some modules into a separate thread or/and process. A process is an instance of the program; each process may contain multiple threads; a process may spawn another process. Unlike, threads, processes cannot share memory spaces with each other.

Python contains a threading and multiprocessing packages for easy creation and management of threads and processes. The framework uses both packages, threading package for communicating with the Bayesian Optimization process, which uses multiprocessing package. On the next page, Figure 5.2.5, is a graphical representation of how the framework is split into processes and threads.

The GUI thread contains all three stages of the user interface. The first stage, Selection, does not interact with the communicator or Bayesian Optimization module. The communicator is spawned during the transition from the Selection stage to the Observation stage. The Communicator thread always listens for messages from the Bayesian Optimization process through Python Multiprocessing Pipes. Pipes allow communication between two processes. Each process can send data through the pipes to another pipe. The data cannot be referential to another object and limited to 32 megabytes. On the other side, the Bayesian Optimization module always checks for a termination command from the

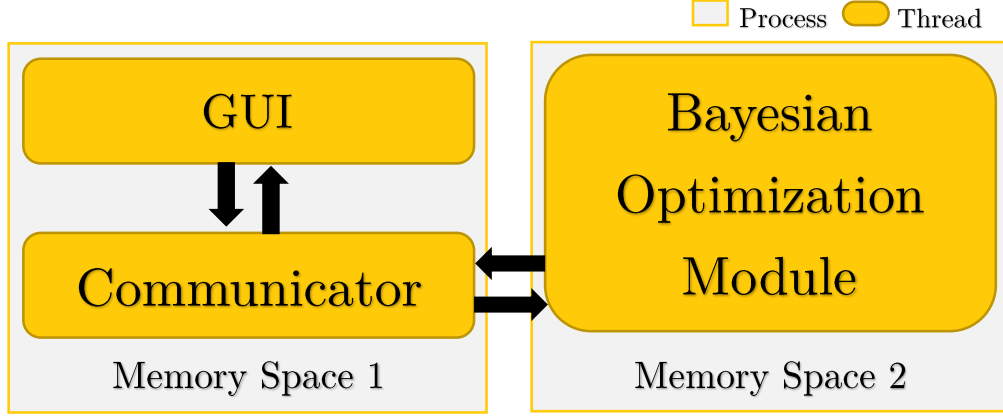


Figure 5.2.5 The overview of the threads and processes in the framework

communicator. The termination code may arrive in the middle of an iteration, the process is terminated only when the iteration is complete and a message is send out to the communicator confirming processes termination. The Bayesian Optimization process sends data back to the communicator every iteration and prior each experiment run. The communicator relays the data to the GUI thread to update the graphs and the data fields of the Observation stage.

The framework proceeds to evaluation stage once the Bayesian Optimization Module has finished or if user request premature termination. The first termination call does not kill the Bayesian Optimization Module. The module switches to a listener mode, always polling for a command from the communicator. The evaluation stage allows the user to query the models created during the Bayesian Optimization process, and if user requests a file to be processed, the GUI sends the file location and model id to the communicator. The communicator relays the information back to the Bayesian Optimization module to write the posterior to the output direction. The threads and processes are terminated fully only once the user closes the framework window.

The benefits of the threads and processes set up in the framework is two-fold. The Bayesian Optimization does not interfere with the performance of the GUI; the GUI remains responsive regardless of what is happening in the Bayesian Optimization process. Similarly, having a communicator on a separate thread, frees the GUI from polling for new data, it receives the data just in time.

5.2.4 INSTALLATION

The project is hosted on Github with detailed instructions on how to install the framework, and details on which packages are required and how to install them, also the instructions are present on the Appendix A3. The project github page is located at <https://github.com/udoniyor/projectb>

The installation process requires the user to have Python 2.7, git, and number of Python libraries prior installing the framework. The project depends on numpy, matplotlib (only for GUI, not required for CLI), pybo, pygy, mwhutils, and scipy. The majority of the dependencies stem from the pybo library which requires all the same packages installed on

the machine. As covered in the section 5.1.2, the GUI library comes bundled with all python installations, therefore does not require any extra installation steps.

All the necessarily Python libraries and the framework can be installed with only two lines of code, shown below.

```
pip install -r https://github.com/udoniyor/projectb/raw/master/requirements.txt
pip install git+https://github.com/udoniyor/projectb.git
```

The first line of code install all the external libraries from the requirements file. The second line clones the github repository and installs the framework on the machine. Possible Operation System issues, and remedies to the issues are presented on the Github page of the project.

5.3 TESTING

In the Section 5.2.1, we have covered the main modules present in the framework. The modules are dependent on each other to a certain extend. The table below covers the manual tests performed, with the command used, required outcome, actual outcome and modules responsible in the interaction.

5.3.1 CLI TESTING

#	Command	Required Outcome	Actual Outcome
C1	python -m project Modules: Core, GUI	Starts the GUI	A bug in the multiprocessing library prevented the <code>__main__.py</code> script to invoke a separate process. As a result the <code>__main__.py</code> was renamed to <code>start.py</code> . The new command to invoke the GUI is: <code>python -m projectb.start</code>
C2	python -m projectb -cli Module: Core	Error, specifying user requires to provide a settings file if command line interface is used.	As Expected

C3	<pre>python -m projectb -cli /home/udoniyor/settingsfile</pre> <p>*Settings file with a wrong parameter</p> <p>Modules: Core, File parser</p>	Error, specifying which parameter value caused an error	As Expected
C4	<pre>python -m projectb -cli /home/udoniyor/settingsfile</pre> <p>*Settings file with all correct parameters</p> <p>Modules: Core, File parser, Bayesian Optimization</p>	The framework parses the file successfully, and starts Bayesian Optimization	The parser would incorrectly parse the Boolean variables due to the incorrect “if” statement. The file parser module was updated to remedy the issue.
C5	<pre>python -m projectb -cli -v /home/udoniyor/settingsfile</pre> <p>*Settings file with all correct parameters</p> <p>Modules: Core, File parser, Bayesian Optimization</p>	The framework parses the file successfully, and prints to the console details of the experiment and starts Bayesian Optimization	The core was printing the wrong verbose level. Updated the file core to print the verbose level correctly
C6	<pre>python -m projectb -cli -vv /home/udoniyor/settingsfile</pre> <p>*Settings file with all correct parameters</p> <p>Modules: Core, File parser, Bayesian Optimization</p>	The framework parses the file successfully, and prints to the console details of the experiment and starts Bayesian Optimization, and prints details of each iteration.	As expected.
C7	<pre>python -m projectb -h</pre> <p>Module: Core</p>	Prints out the help message	As expected
C8	<pre>python -m projectb -cli /home/udoniyor/settingsfile /home/udoniyor/outputdir/</pre> <p>*Settings file with all correct parameters</p> <p>Modules: Core, File parser, Bayesian Optimzation</p>	The framework parses the file successfully, and starts Bayesian Optimization, and outputs the results to the specified directory	As expected

5.3.2 GUI TESTING

The user interface testing cannot be performed without a live user to analyse if the outcome of the actions has resulted in intended result. Therefore the GUI testing was performed manually, by using the software, and actively trying to sabotage the framework. The table below presents the issues discovered, and fixes applied to ensure the future users will not receive such errors.

#	Issue	Solution
G1	The output directory button open a file picker, not a directory picker	The selection page code was updated to present the user with directory picker.
G2	If the graph picked during initialization of the model, an error occurred due to the zero length array.	The graph options button is disabled unless there has been some data generated to present to the user.
G3	The file parser would fail to export if no data was present	The file parser was modified to output only parameters different from the default parameters.
G4	The selection page background contained different tones of grey for Policies and Objective widgets.	The background colour of all widgets was checked and updated if necessary to the correct colour
G5	Changing the graphing option in middle of an iteration can lead to an error. The framework would try to add data to the arrays while the graph was updated.	Implement simple locks preventing changes occurring while the graph is generated. Similarly, prevent graph refresh if data is being added.
G6	The normalization setting would be set to True by the GUI even if the settings file specified False.	The issue was caused by the GUI checking for the correct library, and setting the normalization true if the user did not have nlopt library. This overwrote the parameter specified by the user. The behaviour was modified to prevent such scenario.
G7	Closing the window in the Observation stage during “Saving...” the data stage resulted in an error after the window closed.	The issue was caused due the Bayesian Optimization process trying to communicate with the parent process, which does not exists anymore. The window closing function was updated to terminate the process right away if the window is closed.

5.3.3 OPERATING SYSTEMS

The installation process, Graphical User Interface and the Command Line interface were tested on the Windows 10 Pro and Linux Mint 17.2 Mate operating systems.

Installation

The installation process on both platforms did not incur any problems. Problems may occur with the installation of numpy and SciPy through the Python pip package manager. A work around is included in the instructions in the installation guide (project Github page). For Windows base system the easiest solution is to download a precompiled Python distribution with necessary Python packages, such as Anaconda Python distribution. Unix based systems include a handy package manager, allowing users to install Python packages. More details instructions and links are provided in the Github page of the Project.

Graphical User Interface

The behaviour of the user interface is consistent across both platforms. On other hand, the visual aspects of the GUI differ on the platforms. The TKinter package uses OS based widgets, as a result the buttons and frame are rendered to match rest of the user interface of the operation system. Below, Figure X and Figure X are screenshots of the GUI on Windows and Linux respectively.

Command Line Interface

The command line interface does not suffer from same problems as the GUI. The command are defined by the framework and Python. Therefore, both operating systems behave as expected.

To ensure the framework works on Unix based systems as well as Windows based systems the framework was installed on clean virtual machine instances of the operation systems. The purpose was to analyse, if the provided installation and usage tutorials are sufficient to guide the user, and if the framework performs as expected on Unix based systems, since the development was carried out on a Windows based machine.

5.3.4 USER TESTING

Once the framework was complete, the framework was tested by a potential user of the framework. The user was not given any specific instructions on how to install or use the framework. The only documentation available to the user was presented on the main GitHub page, including on how to install, troubleshoot installation problems, file format specification, and how to use the framework. This was essential part of the testing to ensure any user should be able to install the framework, and if the any trouble occurred along the

way the instructions would be updated to reflect the changes in the installation/usage process.

During the usage of the framework, C1, C5, G1, G3 bugs were uncovered by the user. The issues were promptly fixed, and the GitHub repository was updated. Once all the issues were fixed, the user was successfully use the framework to optimize different models with the framework.

5.4 EVALUATION AND FURTHER WORK

The high-level objective of the framework was to provide an easy to use Graphical User Interface and a Command Line Interface for any command line model to be optimized with Bayesian Optimization technique. We believe we have delivered on our primary goal, but there is a lot of room for improvement.

The framework currently is self-contained, and cannot interact with custom policies or kernels. In some applications, custom kernels and policies can be essential for an optimal performance. A plug-in based policies and kernels, would allow the research community to develop and share custom modules for the framework and carry on the benefits of modularity present in the underlying pybo library.

The current implementation could further be improved by switching the experiments from sequential execution to a parallelized execution. The underlying architecture of the framework supports easy parallelization, but the limitations are applied by the GUI since in the current form it can represent only one experiment at a time.

6 CONCLUSION

We have introduced Bayesian Optimization, identified the main components of the optimization process. Through experimental data we have shown the limitations and characteristics of BO. We identified the bottlenecks of the BO, and effects of the normalization of the input data on two different solvers. Based on the identified problems with Bayesian Optimization, we have developed a new novel variant of Bayesian Optimization with Dimension Scheduling Algorithm. The DSA algorithm successfully outperformed the traditional BO method and achieved better objective values in less computation time than the traditional BO method. In addition to the DSA, we have developed a graphical user interface with command line support. The framework incorporates the traditional Bayesian Optimization Algorithm and the Dimension Scheduling Algorithm with support for the most common kernels and policies. The framework has been successfully tested on real world models and with real users and has been deployed online for anybody to take advantage of the framework and the DSA.

APPENDIX

A1: INITIAL EXPERIMENT RESULTS

Expected Improvement															
MATERN1				MATERN3				MATERN5				SE			
O	ON	N	NN	O	ON	N	NN	O	ON	N	NN	O	ON	N	NN
10547760.5	11503438.8	76768.24	1083.773	13751871.3	11583633.4	277110.7	94265.95	12541457.9	11494663.3	144044.1	104192.7	9938408	10593033	624197.5	40432.85
201.537197	264.816842	98.08365	81.67155	171.243597	168.042621	63.50432	54.83634	173.293067	153.593238	60.56389	64.28592	160.9192	169.1989	74.41224	89.27149
5753246.88	6170110.22	303.0726	164.9722	9175492.33	5967380.36	571.7213	627.2734	8491820.97	5432016	313.8998	631.5574	4658371	5967380	932.7103	728.6004
5.70040353	3.10867061	5.865699	2.556584	14.6504596	3.0706828	5.107278	2.863471	21.4057797	3.14706015	6.576096	3.028224	27.07151	3.095707	7.401564	2.822609

Probability Improvement															
MATERN1				MATERN3				MATERN5				SE			
O	ON	N	NN	O	ON	N	NN	O	ON	N	NN	O	ON	N	NN
9314480	9299671	126789.6	1053.789	13670989.5	7094506	231302.7	66913.92	13154416.3	3709609	298175.9	212514.1	8562599	1605591	203151.2	901.8844
52.92281	60.2309	76.78478	81.67155	1201.37019	254.6478	84.50806	74.88411	163.187297	73.05804	57.56085	32.70784	132.6882	71.41848	48.27869	81.67155
3824685	3909096	231.382	161.3598	9175492.33	107091.6	570.5169	189.6672	9175492.33	10591.88	566.7448	197.3143	2319529	11523.23	485.5674	149.8152
5.555187	2.965332	5.666453	2.520873	12.6486247	3.076138	5.686786	2.559289	20.3816455	2.813308	4.576696	2.833679	13.48185	2.527455	6.475325	2.358551

Upper Confidence Bound															
MATERN1				MATERN3				MATERN5				SE			
O	ON	N	NN	O	ON	N	NN	O	ON	N	NN	O	ON	N	NN
5274878	1393763	94066.89	647654	1490067	380759.3	607331.5	394015.1	2746573	340166.2	153303.2	8176486	3051985	7296761	167633	32236.19
155.3905	172.054	74.72152	101.2147	155.3905	163.219	49.3135	75.76065	154.0656	163.219	50.58148	53.80269	87.75814	155.3905	81.22427	46.53252
1876.153	680.8393	150.1373	604.1808	768.4723	1025.741	1101.043	232.9128	1651.258	723.4476	1700.736	577.6972	2335.492	2139.388	1218.668	381.3321
16.15336	2.975631	6.981745	7.03117	20.90181	2.853309	3.380512	5.55881	21.13594	2.435657	4.870893	4.570615	56.15031	3.919817	8.999228	4.666705

Thompson															
MATERN1				MATERN3				MATERN5				SE			
O	ON	N	NN	O	ON	N	NN	O	ON	N	NN	O	ON	N	NN
12121234	3771630	1569.005	110430.5	9928809	2711279	3041.097	30086.69	10415057	1838029	3565.974	15895	12649058	7409581	4034.781	2935.916
195.641	189.5001	40.43447	58.70725	247.9664	94.93486	69.60402	101.4882	168.1427	164.8474	64.69766	83.66728	201.711	114.6272	83.73345	86.91412
7143471	173654.9	180.982	665.3624	4684989	232247.3	509.5654	1182.116	4605200	175957.5	1485.815	196.3712	7950653	539990.4	1073.393	354.6124
4.751753	2.427113	3.091351	2.46262	19.54725	2.340099	2.631785	2.443647	18.07866	2.379529	2.87282	2.724905	8.403991	2.729833	2.124625	2.181241

A2: FILE FORMAT SPECIFICATION

The Framework defines a simple CSV (Comma Separated Values) as a file format which specifies the parameters of the Bayesian Optimization and model details; such as the input file, command and the output directory.

The file structure follows a simple pattern, each line stores only one parameter. First value is the parameter key, for example “policies”; followed by the value of the parameter, separated by a comma such as “ei” (expected improvement policy). The structure allows having multiple values for each parameter, but this is limited only to the kernel and policy parameters. The limitation is imposed by the graphical user interface, where only multiple kernel and policies can be selected.

Another exception is the bounds parameter. The bounds can be described by stating the key "bounds", followed by a comma, lower bound, comma and upper bound. The key can be defined multiple times to specify many bounds. For example if function has three inputs with bounds between 0 and 100, it should be specified as follows:

```
...
bounds,0,100
bounds,0,100
bounds,0,100
...
```

In addition, to the parameters, the file format stores the data gathered during optimization or user pre-processed data. The order of the parameters does not matter, except for the data parameter. The data parameter must be defined last followed by the data where the second column is y values and starting from the 4th column are the x inputs. The purpose of the strict data column layout is to match the output of the framework. The framework, output is structured in a following way: time per iteration in seconds, objective value achieved at the iteration, mean calculated from the posterior for the point prior evaluation, variance calculated from the posterior for the point prior evaluation, followed by the input values separated via commas.

You do not need to specify all the keys and parameters for the framework, most have reasonable default values. Only following are required for the framework: command, modelinput, modeloutput, and bounds. By default, the framework maximizes with EI policy and SE kernel for 150 iterations with sobol initializer.

Basic Settings Specification:

Keys	Parameters
command	command line string to invoke the function
modelinput	input file for the function. One parameter per line
modeloutput	output file for the function.
bounds	lowerbound,upperbound
outputdir	directory to output the results to
policies	ei,pi,ucb,thompson
kernels	matern1,matern3,matern5,se
iter	number of iterations
objective	min/max
solver	direct*/lbfgs

```

initializer | sobol/middle/uniform
initializernum | number of samples to sample by the initializer
recommender | latent/incumbent/observed
normalize | True/False **

```

- Requires nlopt python library ** Normalize the bounds between 0 and 1. Experiments have shown normalizing the input helps with performance of the lbfgs solver. Preferably, use direct solver without normalization. *** Dimension Scheduler is a technique to improve the performance of the Bayesian Optimization.

Advanced Settings Specification:

```

dims | If dimension scheduler enabled, number of dimensions per permutation
dimscheudler | True/False ***
mcmcburn | Burn number
mcmcn | Number of GPs
eixi | Exploration parameter for the EI policy
pixi | Exploration parameter for the PI policy
ucbxi | Exploration parameter for the UCB policy
ucbdelta | Probability of that the upper bound holds
thompsonn | number of Fourier components
thompsonrng | Random seed

```

Following keys have python code snippets as parameters:

Gaussian Process Settings:
gpsf, gpmu, gpell, gpsn

Hyper-prior Settings
priorsnscale, priorsnmin, priorsfm
priorsfsigma, priorsfmin
priorella, priorellb
priormumu, priormuvar

A3: USER GUIDE

ProjectB is a graphical user interface, which allows untrained users to optimize any model that can be invoked through a command line. The GUI is built on top of a modular Bayesian Optimization library, [pybo](#), which includes most common acquisition functions and kernels and more.

Installation

The easiest way to install this package is by running

```
pip install -r
https://github.com/udoniyor/projectb/raw/master/requirements.txt
pip install git+https://github.com/udoniyor/projectb.git
```

The first line installs any dependencies of the package and the second line installs the package itself. Alternatively the repository can be cloned directly in order to make any local modifications to the code. In this case the dependencies can easily be installed by running

```
pip install -r requirements.txt
```

from the main directory.

Tips

If you are having trouble installing via pip, try installing scipy and numpy with package manager on UNIX based systems. For more details on how to install SciPy stack on your machine look [here](#)

If you are on Windows and having troubles with pip, try [Anaconda](#). It includes numpy and scipy, therefore reducing the chances of running into an error.

Usage (GUI)

To invoke the Graphical User Interface, you need to type in following command into the command line:

```
python -m projectb.start
```

This will launch the GUI of the framework. You may also specify the settings file to prefill the fields in the UI.

```
python -m projectb.start /home/user/settingsfile.projectb
```

Your settings file can have any extension name, but for clarity purposes keep it simple and relevant.

Usage (Command Line)

The framework can be executed via commandline, but you must specify a settings file. Settings file format is a CSV file, with one parameters per line. You will find the specification for the settings file in the next section. Alternatively, you may create a settings file via the GUI by exporting the settings defined in the UI. To invoke the command line, you simply add -cli to the command.

```
python -m projectb.start /home/user/settingsfile.projectb -cli
```

Optionally, you can specify the output directory during the invocation.

```
python -m projectb.start /home/user/settingsfile.projectb  
/home/user/outhere/ -cli
```

It is advised to provide an output directory, otherwise the framework will write to the directory called from.

BIBLIOGRAPHY

- [1] C. Baroukh, R. Muñoz-Tamayo, J.-P. Steyer, and O. Bernard, “DRUM: A New Framework for Metabolic Modeling under Non-Balanced Growth. Application to the Carbon Metabolism of Unicellular Microalgae,” *PLoS One*, vol. 9, no. 8, p. e104499, 2014.
- [2] M. W. Hoffman and R. Shahriari, “Modular mechanisms for Bayesian optimization,” *NIPS Work. Bayesian Optim.*, pp. 1–5, 2014.
- [3] E. Brochu, V. M. Cora, and N. de Freitas, “A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning,” 2010.
- [4] M. a Gelbart, J. Snoek, and R. P. Adams, “Bayesian Optimization with Unknown Constraints,” *{arXiv1403.5607} [cs, stat]*, pp. 1–14, 2014.
- [5] J. Park and K. H. Law, “A Bayesian optimization approach for wind farm power maximization,” vol. 9436, p. 943608, 2015.
- [6] P. I. Frazier and J. Wang, “Bayesian optimization for materials design,” pp. 1–21, 2015.
- [7] D. P. Bertsekas and J. N. Tsitsiklis, “Neuro-Dynamic Programming,” *Athena Sci.*, 1996.
- [8] Jones D.R., “A Taxonomy of Global Optimization Methods Based on Response Surfaces,” *J. Glob. Optim.*, vol. 21, no. 4, p. 39, 2001.
- [9] B. Matérn, *Spatial Variation*, vol. 2. Springer-Verlag, 1986.
- [10] M. L. Stein, *Interpolation of Spatial Data: Some Theory for Kriging.*, Springer S. Springer, 1999.
- [11] J. Schneider, “High Dimensional Bayesian Optimisation and Bandits via Additive Models,” vol. 37, 2015.
- [12] R. Calandra, J. Peters, and M. P. Deisenroth, “An Experimental Comparison of Bayesian Optimization for Bipedal Locomotion,” *Proc. IEEE Int. Conf. Robot. Autom.*, pp. 1951–1958, 2014.
- [13] M. P. Deisenroth, “Distributed Gaussian Processes,” vol. 37, 2015.
- [14] “The Python Profilers,” *Python Software Foundation.*, 2015. [Online]. Available: <https://docs.python.org/2/library/profile.html>.

- [15] M. Davis, “SnakeViz,” 2015. [Online]. Available: <https://jiffyclub.github.io/snakeviz/>.
- [16] A. Cassioli, “A Tutorial on Black-Box Optimization,” no. Lix, 2013.
- [17] A. Skajaa, “Limited Memory BFGS for Nonsmooth Optimization,” p. 50, 2010.
- [18] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. de Freitas, “Bayesian optimization in a billion dimensions via random embeddings,” *arXiv Prepr. arXiv ...*, p. 18, 2013.
- [19] X. Hao, X. Chen, H. W. Lin, and T. Murata, “Cooperative Bayesian Optimization Algorithm: A Novel Approach to Simultaneous Multiple Resources Scheduling Problem,” *2011 Second Int. Conf. Innov. Bio-inspired Comput. Appl.*, pp. 212–217, 2011.
- [20] A. Cully and J.-B. Mouret, “limbo,” 2015. [Online]. Available: <https://github.com/jbmouret/limbo>.
- [21] J. Snoek, “Spearmin,” 2014. [Online]. Available: <https://github.com/JasperSnoek/spearmin>.
- [22] O. R. D. Ginsbourger, V. Picheny, “DiceOptim: Kriging-Based Optimization for Computer Experiments,” 2015. [Online]. Available: <https://cran.r-project.org/web/packages/DiceOptim/index.html>.